Virtual University	CS609-System Programming (Solved Macq's) For Final Term TOPIC 90 to 190 Also 70 to 90 CURRENT MCQs INCLUDE	
Junaidfazal08@gmail.com Bc190202640@vu.edu.pk	FOR MORE VISIT VULMSHELP.COME	JUNAID MALIK 0304-1659294

AL-JUNAID TECH INSTITUTE

CURRENT MCQS FALL 2023

1. What does the system affinity mask indicate?
   - The active process masks
   - The Windows version
   - The number of threads
   - The number of processors on the System TOPIC 126
2. How much virtual address space is consumed by 100 threads?
   - 100MB
   - 1GB CONCEPT
   - 10GB
   - 1TB
3. What is default stack size for each thread in a memory?
   - 100MB
   - 1GB
   - 1MB TOPIC 126
   - 10MB
4. What type of Lock is an SRW LOCK?
   - Kernel Lock & Spin Lock
   - Spin Lock & Shared Lock
   - Exclusive Lock and Shared Lock TOPIC 174
   - Shard Lock Only
5. Which is Fastest Locking mechanism
   - SRW Lock
   - Mutex
   - Interlocked Function TOPIC 174
   - Critical Section
6. _____API open an Existing threads handler from a Thread ID.
   - GetThreadID()
   - GetcurrentThreadId()
   - OpenThread() TOPIC 114
   - GetCurrentThread()
7. The use of_____ensure that the main memory is updated and catch of all the processor is coherent.
   - Memory Barrier TOPIC 134
   - Volatile qualifier & Global Flag
   - Volatile qualifier
   - Global Flag
8. For Mutual exclusion to work_____ variables() must be protected by a single synchronization object across all threads.
   - Important

- Local
- Shared TOPIC 141
- Local Integer

9. What is the size of an SRW Lock?
   - 32 bits & 16 bits
   - 64 bits & 8 bits
   - 8 bits & 16 bits
   - 32 bits & 64 bits TOPIC 163
10. What is the purpose of a critical section in a program with a mutex?
    - To define a specific of code that can only be executed by one thread at a time CONCEPT
    - To allocate memory dynamically
    - To create a user interface
    - To calculate mathematical expression
11. Inherited Handlers of parent and Child processor have_____copies and own states.
    - Similar
    - Distinct TOPIC 92
    - Related
    - Connected
12. When CreateThreadpoolWork() is called then_____ is(are) executed using the process thread pool?
    - Callback Functions TOPIC 173
    - Kernel Functions
    - Kernal Routines
    - Work Object Code
13. If you want to create 4 thread i.e, thread 0, thread 1, thread 2, and thread 3, they all must be created at _____ state.
    - Suspended TOPIC 120
    - Block
    - Running
    - Ready
14. _____Function is used to return the address of the currently running fiber.
    - GetCurrentFuber() TOPIC 129
    - GetFiberDate()
    - GetFiberId()
    - GetFiberIdentity()
15. A fiber can obtain its identity by calling____API
    - GetCurrentFuber() TOPIC 129
    - GetFiberDate()
    - GetFiberId()
    - GetFiberIdentity()
16. A fiber can obtain their data by calling____API
    - GetCurrentFuber()

- GetFiberDate() TOPIC 129
- GetFiberId()
- GetFiberIdentity()

17. A fiber can be created by using_____threads
    - GetCurrentFuber()
    - CreatFiber() TOPIC 129
    - GetFiberId()
    - GetFiberIdentity()
18. A fiber can gives control_____
    - GetCurrentFuber()
    - SwitchToFiber() TOPIC 129
    - GetFiberId()
    - GetFiberIdentity()
19. Which fiber can be deleting using_____ Threads
    - GetCurrentFuber()
    - DeleteFiber() TOPIC 129
    - GetFiberId()
    - GetFiberIdentity()
20. For the efficient use of critical Section and mutex, the thread priority should be decided by the _____.
    - Program
    - OS Scheduler TOPIC 154
    - Mutex
    - Critical Section
21. If entry point of DLL is not specified, then it is an example of_____ linking.
    - Explicit
    - Implicit TOPIC 183
    - Explicit & Implicit
    - Dynamic
22. _____can be used to assign a thread to a specific processor
    - Memory Management Unit
    - Device Driver
    - Process Affinity TOPIC 179
    - Loader
23. How does a Mutex provide synchronization in a program?
    - By Locking access to shared resources TOPIC 138
    - By allocating memory dynamically
    - By creating a user interface
    - By calculating mathematical Expression
24. A process have____ID(s) and _____handle(s)
    - Unique,unique
    - Unique,multiple CINCEPTUAL

A process has one Process ID and one or more handles.

- Multiple,multiple
- Multiple,unique

25. Which of the following synchronization construct is used for semaphore
    - Enter
    - Leave
    - Wait TOPIC 96
    - Set
26. A thread can enter___ time(s) into its critical Section at different instance in thread synchronization.
    - Only one CONCEPTUAL
    - Several CONCEPTUAL
    - Exactly Ten
    - At suspended
27. How many number of Times parameter are provided to a GetProcessTime() API to compute the elapsed time of process?
    - 0 No given answer because it has 5 parameters
      NOTE: No specific number of times parameter is required to compute the elapsed time of a process using GetProcessTimes() SO answer is 0.
    - 1
    - 2
    - 3
28. Producer consumer problem is a classical problem in _____.
    - Multi exclusion
    - Mutual process
    - Mutual exclusion TOPIC 142
    - Object exclusion
29. Which of the following is not a thread model
    - Peer to peer TOPIC 119
    - Boss worker
    - Client-server
    - Pipeline
30. The use of global variable flag in the solution of the critical section problem suffers or fails due to the _____of threads.
    - Deadlock condition
    - Sleeping condition
    - Hold and Wait Condition NOT SURE CONFIRM it ACCORDINGLY
    - Context switch
31. In Thread Local Storage (TLS) arrangement row represents___ While column represent_____
    - Thread Number, TSL index TOPIC 123
    - TLS index, Thread Number
    - Thread number, process Number
    - Process Number, TSL index

32. When all the threads of a process are made exit or terminate, then the process_____
    - Will still survive
    - Will terminate TOPIC 113
    - Will terminate for specified period of time
    - Will temporarily suspended
33. In a GetProcessTime() API, the elapsed time of a process can be computed by subtracting the _____
    - ExitTime from CreationTime parameter
    - CreationTime from ExitTime Parameter TOPIC 100
    - UserTime from ExitTime parameter
    - ExitTime from UserTime Paramater
34. How can you specify a manual-reset event?
    - By Setting bManualReset to FALSE
    - By Setting bManualReset to TRUE TOPIC 150
    - By Setting bInitialState to FALSE
    - By Setting OpenEvent to TRUE
35. ReadFile() and WriteFile() functions perform much____ then memory mapped file processing.
    - Faster
    - Convenient
    - Slower TOPIC 71
    - Nimble
36. File processing is _____ than ReadFile() and WriteFile().
    - Faster TOPIC 71
    - Convenient
    - Slower
    - Nimble
37. Which function is used to obtained the psuedohandle?
    - GetCurrentProcess() TOPIC 93
    - OpenProcess()
    - CreateProcess()
    - GetCurrentProcessId()
38. Which function is used to obtained the actual handle of a process?
    - GetCurrentProcess()
    - OpenProcess() TOPIC 93
    - CreateProcess()
    - GetCurrentProcessId()
39. We provide_____ parameter of the WaitForSingleObject() API.
    - 0
    - 1
    - 2 TOPIC 116
    - 3
40. Windows treats threads as _____.

- ❖ Objects TOPIC 116
- ❖ Threads
- ❖ Processes
- ❖ Classes

41. For Sequential files, File mapping may reduce the processing time by ____ folds as compared to conventional File operations.
    - ➢ Three TOPIC 76
    - ➢ Six
    - ➢ Nine
    - ➢ Ten
42. If the Functions inside the DLL are uploaded, then older program attached to this DLL will not work properly. This statement refers to____ of DLL versioning.
    - ➢ Strength
    - ➢ Problem TOPIC 86
    - ➢ Advantage
    - ➢ Caution
43. Which statement is incorrect when processors are made synchronized?
    - ➢ It can decrease the overhead of system
    - ➢ It can increase the overhead of system CONCEPTUAL
    - ➢ Some process can be blocked in synchronization
    - ➢ Mutual Exclusion is achieved
44. When any key is pressed from the keyboard, the scan code of that key will be sent at port_____
    - ➢ Port 40
    - ➢ Port 20
    - ➢ Port 80
    - ➢ Port 60 CONCEPTUAL
45. What is the purpose of set functions for the mask?
    - ➢ To restrict access to the system
    - ➢ To control the scheduling of threads
    - ➢ To set affinity mask for process It's my OWN knowledge
    - ➢ To set the affinity mask for another process or for itself
46. The generic syntax for declaring bases pointers in Windows is_____
    - ➢ Type_based (base) declarator TOPIC 178
    - ➢ Declarator tyoe_based (base)
    - ➢ Type (base) declarator_based
    - ➢ Type declarator_based(base)
47. Which of the following is return type of CreateThreadPoolWork() API?
    - ➢ LPVOID
    - ➢ PTP_WORK TOPIC 169
    - ➢ PTP_CALLBACKL
    - ➢ PTP_WORK_CALLBACK
48. What is the function of the masks in the System?

- To restrict access to the system
- To control the scheduling of threads
- To set the affinity mask for a process
- To set the affinity mask for another process

49. Which parameter is required to be pass to the EXITThread() API?
    - ThreadID
    - Thread Object name
    - Both Thread ID its Object name
    - Thread EXITCODE TOPIC 113
50. Using ExitProcess() function a process can be made exit or terminate. Which of the following parameters is required to pass the function to make this execution successful
    - process's real handle
    - Process's pseudo handle
    - Process's ID
    - Process ExitCode TOPIC 95
51. Dynamic memory is allocated from the____
    - Paging file TOPIC 71
    - Stack
    - Static memory
    - Cache
52. While using MapViewOfFile(), which of the following are the three commonly used flags?
    - FILE_WRITE, FILE_READ, AND FILE_ALL_ACCESS
    - FILE_MAP_WRITE, FILE_MAP_READ,AND FILE_MAP_ALL_ACCESS TOPIC 74
    - MAP_WRITE,MAP_READ, AND MAP_AL_ACCESS
    - WRITE, READ, AND ALL_ACCESS
53. Each____ is a unit within the____
    - Thread, process TOPIC 87
    - Process, Thread
    - Kernel, Process
    - Process, kernel
54. Window OS does not have any struct that keeps track record of the____ process.
    - Parent-child TOPIC 88
    - Child
    - Parent
    - Independent
55. To executable image name of a process in a CreateProcess() API is specified by____ parameter(s).
    - IpApplicationName only
    - IpCommandLine Only
    - Either IpApplicationName or IpCommandLine TOPIC 90
    - Neither IpApplicationName or IpCommandLine

56. lpApplicationName handle's value_____ be NULL.

- ❖ Should not TOPIC 90
- ❖ May not
- ❖ May
- ❖ should

57. A DLL can____the function

- ➢ Only export
- ➢ Only Import
- ➢ Export and Import CONCEPT
- ➢ Neither export and import

58. Optimal Spin-count ensure____

- ➢ Less system calls TOPIC 162
- ➢ More System Call
- ➢ More overheads
- ➢ Critical Sections Overhead

59. By default, the____ makes scheduling decisions and allocates processors to each thread.

- ➢ System programmer
- ➢ Kernel TOPIC 179
- ➢ BIOS
- ➢ Loader

60. _____API opens an existing thread's Handle from a thread ID.

- ➢ GrtThreadId()
- ➢ GetCurrentThreadId()
- ➢ GetCurrentThread()
- ➢ OpenThread() TOPIC 114

61. ___API creates thread handle from a threadID.

- ➢ GrtThreadId()
- ➢ GetCurrentThreadId()
- ➢ GetCurrentThread()
- ➢ OpenThread() TOPIC 114

62. Each process must have at least _____thread

- ➢ 0
- ➢ 1 CONCEPTUAL
- ➢ 2
- ➢ 3

63. Which of the following is return type of CreateMutex() API?

- ➢ VOID
- ➢ LPVOID
- ➢ HANDLE TOPIC 144
- ➢ BOOL

64. Which of the following is return type of ReleaseMutex() API?

- ➢ VOID
- ➢ LPVOID

- BOOL TOPIC 144
- HANDLE

65. What is an interlocked function in C++?
    - A function that performs non-atomic operations
    - A function that performs atomic operation CONCEPTUAL
    - A function that performs complex operator
    - A function that performs basic operations
66. What is Deadlock?
    - A situation when two or more processes are executing in parallel. CONCEPT
    - A situation when two or more processes are blocked temporally, waiting for each other.
    - A situation when two or more processes are abandoned.
    - A situation when two or more processes are blocked Identically, waiting for each other.
67. What is the purpose of Spin count in SRW lock?
    - To support recursion
    - To determine the size of the lock
    - To reduce the overhead of context switches CONCEPT
    - To allow configuration or adjustment
68. What is range of clock rates for most system today?
    - 0-1GHz
    - 1-2GHz
    - 2-3GHz TOPIC 175
    - 3-4GHZ
69. How are the set Functions used?
    - With Thread and process handles
    - With System and Thread handles CONCEPTUAL
    - With Process and Memory Handled
    - With input and output handle
70. How we can determine the number of processors from the system affinity mask?
    - By Counting the number of bits set CONCEPTUAL
    - By calling the windows version
    - By using a simple program
    - By cutting the number word set
71. If invalid file handle is passed as parameter to CloseFiile() API, then it will be return___
    - True value
    - False Value TOPIC 15
    - A new Handle
    - Close a handle
72. If the file is not opened in concurrent mode, then ReadFile() API starts reading from the____
    - Start of file
    - Current position TOPIC 33

- End of File
- Backup of File

73. The exception handler is actually a code portion associated with _____ block.
    - -finally
    - -try
    - -except TOPIC 48
    - -catch
74. It is_____ to refer to a mapping file multiple times while using parameter as data structure.
    - Useful TOPIC 72
    - Meaning
    - Necessary
    - Optional
75. Which option is incorrect when the TraverseDirectory() API is required to be used?
    - It allow non-recursive traversal
    - Recursive and non-recursive traversal option is irrelevant
    - It allows recursive traversal
    - It allows both non- recursive and recursive traversal
76. The Filter Function _____the type of an exception.
    - Identifies TOPIC 58
    - Evaluates
    - Excludes
    - Restricts
77. If binary Functions inside DLL are accessed at load time, the called as____
    - Explicit linking
    - Implicit linking TOPIC 80
    - Both implicit & Explicit
    - Thread Linking
78. The producer also computes a simple_____ of the message
    - Checksum TOPIC 142
    - cheksub
    - checkdive
    - checkmul
79. Windows provide an object called a _____. Using this object, we can enforce mutual exclusion
    - Multex
    - Multask
    - Mutex TOPIC 144
    - Mutext
80. procedure consumer problem is a classical problem in_____
    - Mutual exclusion TOPIC 142
    - Mutual process
    - Multi exclusion

- Object exclusion

81. Mutexes have advantages beyond CRITICAL_SECTION. Mutex can be named and have____
    - Sections
    - Timers
    - Handles TOPIC 144
    - Self-loop
82. Which of the following synchronization objects is not a named object?
    - Critical section TOPIC 138

    NOTE: ALL names are given and Nearly to close answer is CRITICAL SECTION because it can be only used in syncronize

    - Mutex
    - Semaphore
    - Event
83. To minimize the concurrency related issue_____
    - Mutex is unlocked on entry and locked on exit TOPIC 154
    - Size of the critical region should be Minimized
    - Premature exits from the critical region are necessary
    - Critical Region must have more than one entry
84. According to the MSDN the optimal spin count for heap related function is__
    - 3000
    - 4000 TOPIC 162
    - 5000
    - 6000
85. A(n)____ is a collection of worker threads that execute asynchronization callback on behalf of the application.
    - Address Space
    - Thread pool TOPIC 172
    - Callback Function
    - Environmental Variables
86. _____ Indicates the processors configured on the system
    - Process Affinity mask
    - Processor Affinity Mask System **Affinity Mask:** **TOPIC 180**
    - Thread Affinity Mask
    - Kernel Affinity Mask

    NOTE:

    The actual answer is System Affinity Mask:
87. _____indicates the processors that can be used by the process threads
    - Process Affinity mask TOPIC 180
    - Processor Affinity Mask
    - Thread Affinity Mask
    - Kernel Affinity Mask

88. _____ can be used assign a thread to a specific processor
    - Process Affinity mask TOPIC 180
    - Processor Affinity Mask
    - Thread Affinity Mask
    - Kernel Affinity Mask
89. Each thread has a thread affinity mask which is a subset of process affinity mask.
    - Process Affinity mask
    - Processor Affinity Mask
    - Thread Affinity Mask TOPIC 180
    - Kernel Affinity Mask
90. Thread Affinity Mask is subset of ____
    - Process Affinity mask TOPIC 180
    - Processor Affinity Mask
    - Thread Affinity Mask
    - Kernel Affinity Mask
91. A _____ API is used to create a single threaded child process.
    - CreateFile()
    - CreateProcess()
    - CreateThread() TOPIC 112
    - CreateProcessThread()
92. Total number of parameters required for **CreateProcess()** API is_____.
    - 9
    - 10 TOPIC 88
    - 11
    - 12
93. From which structure, we obtain the newly created child process's handle and ID?
    - LPSTARTUPINFO structure
    - LPPROCESS_INFORMATION structure TOPIC 89
    - lpEnvironment Variable
    - LPSECURITY_ATTRIBUTES
94. _____API is used to map a view of a file-mapping object into the address space of a calling process.
    - CreateViewOfFile()
    - MapViewOfFile() TOPIC 74
    - CreateFileMapping()
    - OpenFileMapping()
95. Which one is not a disadvantage of the static linking during compilations?
    - A big executable file
    - Takes more space on disk
    - Take less physical memory TOPIC 180
    - Consume more bandwidth
96. In Implicit linking, the library functions are linked to the source code at _____.
    - Written time
    - Compile time
    - Run time
    - Load time TOPIC 80
97. In Explicit linking, the library functions are linked to the source code at _____.
    - Written time

- Compile time
- Load time
- Run time TOPIC 80

98. In which type of linking, the DLLs are not used?
- Static linking TOPIC 80
- Explicit Linking only
- Implicit linking only
- Both Implicit and Explicit linking

99. The **CreateProcess()** API returns _____ handle(s) after successful execution.
- Zero
- One
- Two TOPIC 88
- Three

100. The return type of **ExitProcess()** API is ____.
- DWORD
- LPTSTR
- HANDLE
- VOID TOPIC 95

101. The number of parameters required for **TerminateProcess()** API is _____.
- 1
- 2 TOPIC 95
- 3
- 4

102. After loading a DLL explicitly, we can obtain the address of any entry point in DLL using the function_____.
- GetModuleHandle
- GetProcAddress() TOPIC 83
- LoadLibrary()
- LoadLibraryEx()

103. The number of parameters required for **CreateJobObject()** API is _____.
- 0
- 1
- 2 TOPIC 108
- 3

104. The return type of **CreateThread()** API is _____.
- DWORD
- LPDWOR
- HANDLE TOPIC 112
- LPTSTR

105. The default stack size for a thread is _____.
- 1 MB TOPIC 126
- 1 KB
- 1 Byte
- 1 bit

106. The number of parameters required for **GetExitCodeThread()** API is _____.
- 0
- 1
- 2 TOPIC 113
- 3

107. TLS stands for _____.

- Thread Local Storage TOPIC 111
- Thread Last State
- Thread Local Size
- Timestamp Local Storage

108. A unit of execution that is scheduled by the application rather than by the OS Kernel, is called as _____.
- Fiber TOPIC 180
- Thread
- Process
- Mutex

109. A fiber can obtain its starting address by using the API _____.
- CreateFiber()
- GetFiberData()
- GetCurrrentFiber()
- ConvertThreadToFiber()

110. The return type of **CreateFiber()** API is _____.
- VOID
- LPVOID CONCEPT
- HANDLE
- BOOL

111. Which one of the following is not a Process or thread synchronization object?
- Fiber CONCEPTUAL
- Critical Section
- Mutex
- Semaphore

112. Interlocked functions apply to _____ integers.
- 32 bits signed
- 32 bits unsigned
- 64 bits signed
- 64 bits unsigned

113. Which one of the following is not a requirement for correct thread code?
- Global Storage
- Volatile Storage
- Memory Barrier
- Critical Code Region

114. The return type of **CreateSemaphore()** API is _____.
- DWORD
- HANDLE
- BOOL TOPIC 147
- VOID

115. The code region where only one thread can execute at a time is known as _____.
- program
- critical section CONCEPT
- non – critical section
- Exception Handler

116. Which one of the following APIs block a thread if another thread is in the section?
- InitializeCriticalSection
- EnterCriticalSection
- DeleteCriticalSection
- LeaveCriticalSection

117. Which one of the following numbers is good spin count for heap management according to MSDN?
- 1000
- 2000
- 3000
- 4000 TOPIC 162

118. Which one of the following concepts limit the number of active, contending threads?
- Mutual Exclusion
- Thread Contention
- Semaphore Throttle CONCEPT
- Parallelism

119. Which one of the following APIs returns a pointer to a PTP_WORK structure?
- TrySubmitThreadpoolCallback
- WaitForThreadpoolWorkCallbacks
- CreateThreadpoolWork TOPIC 169
- SubmitThreadpoolWork

120. Thread synchronization is a process of handling situation when _____ or more threads need access to a shared resource.
- One
- Two
- Three
- Four

121. Which one of the following options best describe the mutex?
- is a binary mutex
- does not have handles
- must be accessed from only one process
- can be accessed from multiple processes

122. What will happen if a non-recursive mutex is locked more than once?
- Starvation
- Deadlock CONCEPT
- Critical Section
- Semaphore

123. What are the two kinds of semaphores?
- mutex & counting
- binary & counting CONCEPT
- counting & decimal
- decimal & binary

124. _____ function is supported only on the Itanium processor.
- InterlockedExchangeAdd
- InterlockedExchange
- InterlockedExchangeAcquire
- lockedCompareExchange

125. Instead of starting a new thread for every task to execute concurrently, the task can be passed to a _____.
- process
- thread pool CONCEPTUAL
- thread queue
- CPU

126. Each process has its own process affinity mask, which is a _____ vector.
- Bit TOPIC 180
- byte
- Nibble
- Word

127. Which one of the followings is not a feature of named pipes?
- message-oriented
- unidirectional TOPIC 184
- independent instances of pipes
- Networked clients can be accessing pipes by name

128. Which one of the following is not a characteristic of Anonymous pipe?
- Character based
- Half Duplex
- Full Duplex TOPIC 181
- Communicate with Linux & UNIX both

129. Which one of the followings has two handles: Read handle and Write handle?
- Named Pipes
- Anonymous Pipes TOPIC 181
- Unnamed Pipes
- Interposes Pipes

130. The return type of **CreatePipe()** API is _____.
- DWORD
- HANDLE
- BOOL TOPIC 182
- VOID

131. Total number of parameters required for **CreateNamedPipe()** API is _____.
- 8 TOPIC 186
- 9
- 10
- 11

132. Number of parameters taken by OpenFileMapping()?
- 0
- 2
- 3 TOPIC 73
- 1

133. What the return Type of InterlookedExchange()?
- LONG 100% CONFIRM
- BOOL
- DWORD
- LOPVOID

PREE-POST ASSISMENT TOPIC-WISE

134. The GetCommandLine() function returns the command line as a single_____
- Float variable
- Integer Array
- Integer Variable
- Character string Topic 101

135. Simple job management shell will allow _____ commands to run.
- Two
- Five
- Three Topic 104 (jobbg,jobs,kill)
- Four

136. In job management shell the shell uses ____ specific file keeping track of process ID and other related information
- Shell
- Process
- System
- User Topic 104

137. Process identify can be obtained from the _____ structure.
- CreateProcess()
- CREATE_PROCESS()
- PROCESS_INFORMATION Topic 93
- GetProcessInfo()

138. If the system is _____ time is multiplexed among multiple processes in an interleaved manner.
- Uniprocessor system Topic 99
- Linear processor system

- Multicore processor system
- Multiprocessor system

139. If a system is _____ then windows scheduler can run process threads on separate processors.
- Uni processor
- Linear processor
- Single core processor
- Multiprocessor Topic 99

140. FILETIME parameter of GetThreadTime() function is a ____ bit value.
- 32
- 8
- 16
- 64 Topic100

141. Closing the child process handle _____ the process.
- Does not create
- Create
- Destroy
- Does not destroy Topic 93

142. Closing the child process only closes the access of the _____.
- All parent process
- Parent process Topic 93
- All children processes of parent process
- Child process

143. In windows, there are _____ ways to get command line parameter for a process.
- Two Topic 96
- Four
- Five
- Three

144. Ip ApplicationName handle's value _____ be NULL.
- Should
- Should not According to me
- May
- May not

145. Control of a running fiber can be given to another fiber by using --------- function.
- GetCurrentFiber()
- SwitchToFiber() Topic 129
- CreateFiber()
- ConvertThreadToFiberEx()

146. A thread can enable fiber operation by calling _____function.
- GetCurrentFiber()

- SwitchToFiber()
- CreateFiber()
- ConvertThreadToFiberEx() Topic 129

147. Inherited handles are _____ copies that a parent and child might be accessing.
- Connected
- Similar
- Distinct Topic 118
- Related

148. A fiber can obtain its identity by calling _____function.
- GetFiberData()
- GetCurrentFiber() Topic 129
- GetFiberId()
- GetFiberIdentify()

149. The _____ function is used to obtain the process handle using the process ID.
- openProcesses()
- GetCurrentProcess()
- OpenProcess() Topic 93
- OpenprocessID()

150. Parent process usually creates a _____ handle if parent and child process require ____ access rights.
- Duplication, same
- Duplication, different Topic 94
- New, same
- New, different

151. There are ------ APIs to manage fiber.
- 10
- 9
- 7 Topic 129
- 8

152. One process can finish another process using the function _____
- FinishProcess()
- CreateProcess()
- ExistProcess()
- TerminateProcess() Topic 95

153. The simplest form of synchronization can be achieved through___ construct.
- Run
- Halt
- Lock
- Wait Topic 96

154. The single object for which the process waits is activated by _____.

- IpHandle
- bWaitAll
- hHandle Topic 96
- nCount

155. The array of handles for which the process wait is activated by _____
- IpHandle Topic 96
- hHandle
- nCount
- bWaitAll

156. _____is the number of objects in an array. Should not exceed MAXIMUM WAITOBJECTS
- IpHandle
- hHandle
- nCount Topic 96
- bWaitAll

157. _____is the timeout period for wait. 0 for no wait and INFINITE for indefinite wait.
- IpHandle
- hHandle
- nCount
- dwMilliseconds Topic 96

158. _____handle describes if it's necessary to wait for all the objects to get free
- IpHandle
- hHandle
- nCount
- bWaitAll Topic 96

159. How many The possible return values are___
- 2
- 3
- 4
- 5 Topic 95

160. The timeout period for wait() function is measured in _____.
- Millisecond Topic 96
- Kilo hertz
- Mega hertz
- Microsecond

161. Its not a good idea to use _____ within the program as it will not give it a chance to release resources.
- createProcess()
- TerminateProcess()

- ExitProcess() Topic 95
- FinishProcess()

162. A new fiber can be created by using_____
- ConvertThreadToFiberEx()
- ConvertThreadToFiber()
- CreateFiber() Topic 129
- GetFiber()

163. In Environment Block, IpName is a ____ name.
- Stack
- Array
- Process
- Variable Topic 97

164. In Environment Block, GetEnvironmentVariable() function returns _____ in case of failure.
- 0 Topic 97
- NULL
- -1
- UNDEFINED

165. In Environment Block, PATH is an example of an environment _____
- Stack
- Array
- Process
- String Topic 97

166. An Environment Block is associated with _____ process(es) in the system.
- Some
- No
- Each and every Topic 97
- Exactly one

167. In Environment Block (EB), each string is _____
- Five character long
- Null terminated Topic 97
- Undefined
- Empty

168. The wait() function is limited to _____ handles.
- 32
- 8
- 64 Topic 98
- 16

169. A _____ value will cause a thread to move from the running state to the ready state.

- Negative
- INFINITE
- 1
- 0 Topic 127

170. Fiber which is shared to different threads_____
- Should not access private data
- Should not access global data
- Should access thread specific data
- Should not access thread specific data Topi 128

171. Fibers are scheduled by the _____
- Hardware
- Operating system
- Application Topic 128
- BIOS

172. IPC stands for
- Inter process communication My Point of View
- Inter privacy communication
- Information process communication
- Information and privacy communication

173. The sleep function allows a thread to move the ____ to the ____ state.
- Running, Suspended
- Running, terminated
- Running, ready
- Running,wait Topi 127

174. The function SwitchToThread() provides a way for a thread to yield its processor to another _____ thread if there is one that is _____ to run.
- Ready, Running
- Running, Running
- Ready, Ready Topic 127
- Running, Ready

175. The time period in sleep function is specified in _____.
- Seconds
- Milliseconds My point of View
- Nanoseconds
- Microseconds

176. Default stack size for a thread is _____.
- 1MB Topic 126
- 1 Byte
- 1 Bit
- 1KB

177. A scheduler will move a _____ threads to the _____ state if the threads time slice expires without the thread waiting.
- Waiting, Waiting
- Running, Ready Topic 125
- Running, Waiting
- Waiting, Ready

178. When a thread is suspended it goes into _____
- Terminated state
- Waiting state
- Suspended state
- Ready state Topic 125

179. A thread that is ready but do not have required resource will go into _____
- Running state
- Suspended state
- Waiting state My point of View
- Execution state

180. The scheduler will run the _____ priority ____ thread when a processor becomes available.
- Highest, ready Topic 125
- Lowest, ready
- Highest, running
- Lowest, running

181. If there are no THREAD_PRIORTY_TIME_CRITICAL thread, the process will run _____ first.
- THREAD_PRIORTY_LOWEST
- THREAD_PRIORTY_NORMAL My point of View
- THREAD_ORIORITY_BELOW_NORMAL
- THREAD_PRIORTY_IDLE

182. Which of the following priority is the highest class in the thread?
- THREAD_PRIORTY_ABOVE HIGHEST
- THREAD_PRIORTY_TIME_CRITICAL My point of View
- REALTIME_ORIORITY_CLASS
- THREAD_PRIORTY_HIGHEST

183. The range of relative thread priorities is between_____.
- 0 to 1
- 0 to 4
- -2 to +2 (+-2) Topic 123
- -1 to +1

184. Which of the following thread priorities will run first?
- Thread with THREAD_PRIORTY_ABOVE_NORMA My point of View

- Thread with THREAD_PRIORTY_BELOW_NORMAL
- Thread with THREAD_PRIORTY_ _IDEAL
- Thread with THREAD_PRIORTY_NORMAL

185. In windows, most common processes have _____ priority class
- HIGH_PRIORTY_CLASS
- IDLE_PRIORTY_CLASS
- NORMAL_PRIORTY_CLASS My Point View
- REALTIME_PRIORTY_CLASS

186. The value returned using TIsSetValue() function is in the form of _____
- BOOL My Point of View
- INT
- LPVOID
- DWORD

187. In Thread Local Storage (TLS) arrangement, row represents _____ while column represent _____
- Thread TLS Index Topic 123
- Process Number, TLS Index not sure
- TLS index, Thread
- Thread, process Number

188. Which of the following function frees the specified index numbers?
- TIsClear()
- TIsFree() Topic 123
- TIsReSet()
- TIsDelete()

189. API is used to allocate the index and it returns the TLS index in theform of the double word
- TIsClear()
- TIsAlloc() Topic 123
- TIsReSet()
- TIsDelete()

190. _____Provided valid indexes are used, The programmer can access TLS spaceusing these simple GET/SET APIs
- TlsGetValue ()
- Both A and C Topic 123
- TlsSetValue ()
- TIsDelete()

191. In the form of failure, TIsAlloc() function returns_____.
- DWORD
- 0
- -1 Topic 123

- BOOL

192. Which of the following is a correct statement?
- Every worker work with multiple processors on multiple processor
- Every worker work as a separate thread on a separate processor Topic 122
- Every worker work as a separate thread on a single processor
- Every worker work with multiple processors on single processor

193. Which of the following is an optimal situation?
- After certain limit processor speed can also be enhanced
- After certain limit processor speed cannot be enhanced Topic 122
- Multiprocessing is not responsible for multiple flows of execution
- Output of parallel program should not be same when it is serialized

194. Which of following statement is incorrect?
- Program performance can be scaled without any certain limit.
- Program performance can be enhanced by using multithreading.
- Program performance can be enhanced with Parallelism.
- Program performance scales automatically, up to some limit.

195. Four threads are created i.e. thread 0, thread 1, thread 2, and thread 3 and are running to sort a large file, select the most appropriate statement?
- Wait for thread 1 to complete and merge it with thread 2.
- Wait for thread 0 to complete and merge it with thread 2.
- Wait for thread 1 to complete and merge it with thread 0. My point of view
- Wait for thread 2 to complete and merge it with thread 0.

196. Why a file is always mapped before accessing it?
- To access it just like accessing some data from main memory My point of view
- To make it secure
- To reuse it any time
- To access it in the form of LinkedList

197. Once all the threads are created, then can be run using _____ function.
- ResumeThread()
- ReadyThread()
- RunThread() Topic 121
- StartThread()

198. The basic difference between boss-worker thread model and client-server model is _____
- In boss worker all the thread are run at the boss's end, but in client-server model, each client run a different thread
- In the client-server all the thread are run at the server end, but in the boss worker model, each worker runs different thread Conceptual

199. Which of the following is not a thread model?
- Peer-to-peer Topic 119

- Pipeline
- Boss-worker
- Client-server

200. In client server model, rather than_____ work is done _____
- Concurrently, sequentially
- Problematic, Efficiency
- Sequentially, Concurrently Topic 119
- Linear, Straight

201. In which of the following models, work moves from one thread to the next thread?
- Peer-to-peer Model
- Pipeline Model Topic 119
- Boss-worker Model
- Client-server Model

202. The _____ flag is set to be _____ in the CreateProcess() function, which will determine whether child process will inherit copies of parent open handles.
- bInheritFlag, FALSE
- bInheritHandle, TRUE My point of View
- bInheritFlag, TRUE
- bInheritHandles, FALSE

203. Inherited handles are _____ copies that a parent and child might be accessing
- Similar
- Connected
- Distinct Wikipedia
- Related

204. The get.JobNumber() function looks into the file for a vacant place. If no place is available, It appends a new record at the_____
- Next
- Start
- Middle
- End My point of View

205. In getting a job number, the job Management Shell uses a number of job management functions to manage jobs information within the _____ life.
- Process
- User Topic 103
- Shell
- System

206. Fiber management occurs at the _____ level.
- Hardware
- User space Topic 128

- Kernel
- BIOS

207. In Asynchronous input/ Output _____.

- Input from the keyboard will be taken after the song is ended.
- Song cannot be played until the other operations are not completed.
- Song can be played with other operations being executed same time.

Topic 119 also Conceptual (correct Accordingly)

- Two operations cannot be run at the same time.

208. The first two parameters in argc will be _____ and _____ respectively.

- Process name, pattern which is to be searched Topic 118
- Inputted file names, pattern to be searched
- Pattern to be searched, process name
- Inputted file names, process name

209. Total number of files inputted, can be obtained with _____-

- Argc-1
- Argc-2 Topic 118 (Confusion kindly Correct Accordingly)
- Argc-3
- Argc

210. C Library Threading functions are _____ than windows library functions but not _____

- Simpler, Hard
- Hard, Simpler
- Diverse, Simpler
- Simpler, Diverse Topic 118

211. LIBCMT is a _____

- BIOS Library for thread
- Java Library for threads
- C Library for threads Topic 118
- Windows Library for threads

212. _____function is used to extract ____ from a string.

- gettok(), Token
- getTok, String
- strtok(), String
- strtok(), Token Topic 118

213. The return type of _beingthreadex() function _____

- is HANDLE, but we need to type caste it for further processing.
- Is not HANDLE, but we cannot type caste it for further processing.
- is not HANDLE, we need to type caste it for further processing.

Topic 118 Conceptual

- is HANDLE, but we don't need to type caste for further processing.

214. _____ library function are thread safe than _____ library functions.
- C, java
- Java, C
- C, Windows Topic 118
- Windows, C

215. Windows treats threads as _____.
- Objects Topic 116
- Threads
- Processes
- Classes

216. When a thread exits, the thread _____ is deallocated and the handle referring to the thread in invalidated.
- Stack Topic 113
- List
- Queue
- Array

217. ResumeThread() function will _____.
- Decrease the value of suspend count Topic 112
- Reset the value of suspend count
- Increase the value of suspend count
- Not affect the value of suspend count

218. By default, the value of Suspend count, when creating a thread, is_____.
- 2
- 1
- 0 Topic 112
- -1

219. Which of the following version of windows was not compatible with GetProcessIdOfThread() function?
- Windows 2003
- Windows 7
- Windows NT
- Windows XP Topic 115

220. Which of the following function can be used to map a process with a thread?
- GetProcessIdOfThread() My Point of View
- DWORD ResumeThread (HANDLE hThread)
- DWORD SuspendThread (HANDLE hThread)
- GetThreadOPendingFlag()

221. A thread will execute if and only if its Suspend Count value is _____
- Non zero

- Above 1
- 0 Topic 115
- 1

222. Threads can also be treated as parent and child although the _____ is unaware of that.

- Kernel
- Thread
- Program
- Operating system Topic 112

223. Threads share resources within a _____

- Code
- Program
- Thread
- Process Topic 110

224. Threads uses the space assigned to a _____

- Thread
- Process Topic 111
- Code
- Program

225. In a multi-threading system, multiple threads may exist within a single _____

- Thread
- Process Topic 109
- Code
- Program

226. _____ is an independent unit of execution within a process.

- Thread Topic 109
- Process
- Code
- Program

227. A job object is used to _____ process execution time and obtain user time statistics.

- Limit Topic 108
- Write
- Read
- Open

228. The getJobNumber() function looks into the file for a vacant place. If no place is available, it appends a new recover of the file.

- Start
- Next
- Middle

- End — My Point of View

229. In job Management Shell, the shell uses _____ specific file keeping track of process ID and other related information.
   - User — Topic 104
   - Process
   - Shell
   - System

230. In finding a Process ID, the FindProcessId() obtains the process ID of the given job number. It simply looks on job number and _____ the record at the specific location.
   - Sums
   - Executes
   - Reads — Topic 106
   - Writes

231. Simple job management shell will allow _____ commands to run.
   - Two
   - Three — Topic 104 (Jobbg, jobs, kill)
   - Four
   - Five

232. In listing background jobs, the job management function uses display jobs () function. This function _____ a file
   - Run
   - Start
   - Process
   - Open — Topic 105

233. In Job Objects, firstly a job object is created using CreateJobObject(). It _____ a name and security attributes.
   - Writes
   - Closes
   - Uses — Topic 107
   - Throws

234. Control of a running fiber can be given to another fiber by using _____ function.
   - GetCurrentFiber()
   - SwitchToFiber() — Topic 129
   - CreateFiber()
   - ConvertThreadToFiberEx()

235. A thread can enable fiber operation by calling_____ function.
   - GetCurrentFiber()
   - SwitchToFiber()
   - CreateFiber()
   - ConvertThreadToFiberEx() OR ConvertThreadToFiber() Topic 129

236. Fiber can obtain its identity by calling _____ function.
- GetFiberData()
- GetCurrentFiber() Topic 129
- GetFiberId()
- GetFiberIdentity()

237. There are _____ APIs to manage fibers.
- 10
- 9
- 7 Topic 129
- 8

238. A new fiber can be created by using_____
- ConvertThreadToFiberEx()
- ConvertThreadToFiber()
- CreateFiber() Topic 129
- GetFiber()

239. The wait() function is limited to _____ handles
- 32
- 16
- 8
- 64 Topic 98

240. Windows OS does not have any structure that keeps track record of the _____ processes.
- Child
- Grand-child
- Parent
- Parent-Child My point of View

241. In comparison of DLL, executable library files are linked at _____ time.
- Link
- Run
- Compiler My Point of View and GOOGLE
- Load

242. Interlocked functions are _____to use.
- Simpler, Faster but hard
- Simpler, Faster and easy Topic 135
- Not simpler but faster and easy
- Simpler easy but slow

243. Thread should not change the _____ environment.
- Process Topic 137
- Hard disk
- Integer

- Ram

244. Which of the following function can be used to map the process with thread?

- DWORD Resume Thread (HANDLE hThread)
- DWORD Suspend Thread (HANDLE hThread)
- GetProcessIdOfThread() My Point of View
- GetThreadIOPendingFlag()

245. Mutexes, Semaphores, Events, and Critical Section are the four synchronization objects provided by _____.

- Software
- RAM
- Windows Topic 138
- processor

246. Like every other resource, threads are also treated as_____

- Code
- Object Topic 112
- Program
- Thread

247. The process Execution times uses the API GetCommandLine() to get the command line as a single _____.

- Bool
- Float
- String Topic 102
- None

248. lpApplicationName _____ be NULL.

- May
- Should
- Should not My point of View
- May not

249. Volatile stage is a _____ level facility.

- Compiler or windows Topic 133
- Programmer
- BIOS
- Hardware

250. CRITICAL_SECTION Objects do not have_____.

- Loops
- Handler Topic 139
- Variables
- Process

251. CRITICAL_SECTION Objects do not have___and are not shared among the____.

- Handler
- Both A and C Topic 139
- process
- Process

252. _____ variables should not be accessible globally.

- String
- Locally required Topic 139
- Long
- Integer

253. Thread IDs and handles can be obtained using functions quite similar to the ones used with_____

- Process Topic 114
- Program
- Thread
- Object

254. A thread can be signaled from which of the following?

- ResumeThread()
- CloseThread()
- CreateThread()
- ExitThread() OR TerminateThread() Topic 117

255. If there are 129 objects, how many times we will have to call the wait () function?

- 4
- 6
- 3 My Point of View
- 2

256. ResumeThread() function will ___

- Increase
- Decrease Conceptual
- None of given
- Both

257. If you want to create 4 thread i.e, thread 0, thread 1, thread 2, and thread 3, they all must be created at _____ state.

- Suspended Topic 120
- Blocked
- Running
- Ready

258. It must be ensured that _____ thread(s) is/ are not modifying ____ data at same time.

- Many, Many
- 1, Many

- Many,1 Topic 131 Conceptual
- 1,1

259. The result of concurrent processing and normal processing _____
- Yield same output
- Is always different
- Is always same
- May differ in some cases My point of View

260. In memory architecture and barriers, before writing the value s back to memory, the processor usually keeps them in _____
- Hard disk
- Memory card
- Processor
- Cache Topic 134

261. _____ are used to ensure that memory is accessed in the desired order.
- Memory barriers or memory fences Topic 134
- Memory paths or memory houses
- Multi processors
- Memory blocks or memory chains

262. Turning off _____ may adversely affect the performance but sometimes may _____ the program.
- Usability, faster
- Optimization, slow Topic 133
- Performance, slow
- Security, faster

263. Turning off Optimization may adversely affect the performance but sometimes may slow the program. ANSI C provides a qualifier _____ for this purpose.
- Volatile Topic 131
- Non-volatile
- Both A and B
- None of Given

264. The volatile qualifier does not guarantee that the modifications will be visible to _____ in a desired order.
- Processors Topic 134
- User
- Ram
- Hard disk

265. The _____ functions provide memory barriers.
- Interlocked Topic 139
- User define
- Interchange

- Intermediate

266. Interlocked increment () takes a 32-bit signed variable as an arguments, which should be placed in memory at the _____ boundary.
- 4-byte Topic 135
- 8-byte
- 16-byte
- 2-byte

267. If a variable with a volatile scope only needs to be incremented, decremented, or exchanged, interlocked functions are the _____ choice.
- Last
- Best Topic 135
- 2nd last
- Worst

268. _____are most suited if variable with volatile scope only need to be incremented,decremented and exchanges
- Interlocked functions Topic 135
- Local
- Global
- None

269. The most basic of Interlocked Functions
- Interlockedincrement()
- InterlockedDecrement()
- Both A and B Topic 135
- None of Given

270. How many Basic Function of Interlocked()
- 1
- 2 Topic 135
- 3
- 4

271. If any needed variables are not initialized, then create _____ threads until variables are initializes.
- Suspended Topic 137
- Double
- Slow
- Fast

272. In thread safe code, _____ conditions are avoided.
- Race Topic 137
- Loop
- Local
- Global

273. When more than one thread can run the same code without introducing synchronization issues, it is said to be _____
- Thread safe Topic 137
- Local
- Threaded
- Initialized properly

274. InterlockedIncremen64() and interlockedDecrement64() can be utilized on 64-bit systems if the addend is paced at the _____ boundary.
- 8-byte Topic 135
- 4-byte
- 2-byte
- 16-byte

275. _____ have a performance disadvantage since they create a memory barrier.
- Interchanged function
- Interlocked function Topic 135
- Intermediate function
- Interlocked memories

276. If a _____ is used to store thread-specific data, other threads will also use it.
- Small variable
- Large variable
- Global variable topic 136
- Local variable

277. Make a variable_____ if we know it includes thread-specific information.
- Static
- Local Topic 136
- Global
- Random

278. According to the criterion for good thread code, _____ storage should not be used for the purpose of local storage.
- Hard disk
- Global Topic 136
- Cache
- Ram

279. If we know the information in a variable will be used by other threads, we can make it _____.
- Global Topic 136
- Local
- Main
- Auto

280. When using thread synchronization objects, there are always inherent risks associated with them such as _____
- Lower power
- Multi clocks
- Load
- Deadlocks Topic 137

281. A thread waits for other threads to ____ using the waitForString0bject() or waitForMultipleObjects() methods.
- Wait
- Terminate Topic 138
- Start
- Update

282. The part of the code that can only be executed by one thread at a time is referred to as the _____
- Process section
- Loop section
- Top section
- Critical section Topic 139

283. Only _____ can be in the CRITICAL_SECTION variable at a time
- Odd threads
- Many threads
- One thread Topic 139
- Even threads

284. Which function is used to initialized the CRITICAL_SECTION.
- InitializeCriticalSection() Topic 139
- DeleteCriticalSection()
- Both A and B
- None of Given

285. Which function is used to Delee the CRITICAL_SECTION.
- InitializeCriticalSection()
- DeleteCriticalSection() Topic 139
- Both A and B
- None of Given

286. Multiple threads may call EnterCriticalSection() but only _____ thread is allowed to enter the critical section while the rest areblocked.
- One Topic 139
- Two
- Three
- Foure

287. A call to LeaveCriticalSection() must match a/ an____

- EnterSection()
- Variables
- EnterCriticalSection() Topic 139
- CriticalSectionPart()

288. CRITICAL_SECTION object do not have _____
- Process
- Variables
- Handles Topic 139
- Loops

289. EnterCriticalSection() can be called by ____ but only one of them is allowed to enter the critical section, while the others are blocked.
- Two threads
- One thread
- Half thread
- Many threads Topic 139

290. Using the critical section construct is ____ and intuitive.
- Easy Topic 140
- Difficult
- Short
- Long

291. It would be ____ to use different objects in the same thread or across numerous threads that share the same data.
- Easy
- Incorrect Topic 141
- Correct
- Difficult

292. For mutual exclusion to work, _____ variables must be protected by a single object across all threads.
- Integer
- Important
- Shared Topic 141
- specific

293. Within the critical section, all the variables must be guarded by _____
- Critical object
- All objects
- A single object Topic 141
- Double object

294. What is the limit for object to be waited for waitForMultiple Objects () function in window?
- 63

- 65
- 62
- 64 Topic 141

295. Producer consumer problem is a classical problem in_____
- Critical Section
- Conical collection
- Mutual Exclusion Topic 142
- None of given

296. The producer periodically _____ a message.
- Creates Topic 142
- Delete
- Update
- None

297. The producer also computes a simple_____ of the message
- Checksum Topic 142
- Contents
- Update
- None

298. _____is a short form of Mutual Exclusion
- Mechanism
- Checksum
- Mutex Topic 143
- All of the given

299. Which of the following Windows functions used to manage mutexes.
- CreateMutex()
- ReleaseMutex()
- OpenMutex()
- All of the given Topic 144

300. Home Many windows Function are used to manage mutex.
- 1
- 2
- 3 Topic 144
- 4

301. lpMutexName is the name of the mutex.
- True Topic 144
- False

302. OpenMutex() is used to _____ and _____ named mutex
- Existing,open
- Open,Existing Topic 144
- Open,Delete

- Exsiting,Delete

303. However, abrupt termination of a thread indicates a serious programming flaw. Mutex waits can_____
- Time In
- Time Out Topic 146
- Has
- Does Not

304. If NULL is returned it indicates dose not failure.
- True
- False Topic 144

305. Semaphore maintains a count
- True Topic 147
- False

306. A semaphore is in a signaled state when the count is greater than.
- 1
- -1
- 0 Topic 147
- None of given

307. Semaphore is_____ when the count is 0.
- Signaled
- Un-signaled Topic 147
- Release
- None of the given

308. Which of the following are used in semaphore Count?
- CreateSemaphore()
- CreateSemaphoreEx()
- OpenSemaphore()
- All of the Given Topic 147

309. cReleaseCount gives the count after the release and must be greater than____.
- 1
- -1
- 0 Topic 147
- None of given

310. Events are classified as either
- manual-reset
- auto-reset
- Both A and B Topic 150

311. A _____ event can signal several waiting threads simultaneously and can be reset.
- manual-reset Topic 150

- auto-reset

312. An _____event signals a single waiting thread, and the event is reset automatically.
- manual-reset
- auto-reset Topic 150

313. If bInitialState is TRUE, the event is set to a_____state
- Signaled Topic 150
- Un-Signaled

314. If the event is manual-reset, it remains signaled until a thread explicitly calls ResetEvent().
- True Topic 150
- False

315. How many ways to use Events?
- 1
- 2
- 3
- 4 Topic 151

316. Windows operating system is a multithreaded _____ that provides support for real time applications and multiprocessors.
- Hardware
- User space
- Kernel Topic 153
- BIOS

317. Program design and performance can be simplified and improved by
- Odd threads
- Many threads
- Threads Topic 153
- Even threads

318. _____Synchronization is a way to coordinate processes that use shared data
- Shell
- Process Topic 153
- System
- User

319. Which of the following Synchronization objects.
- Critical Section
- Semaphore
- Mutex
- All of the Given Topic 153

320. The_____functions provide a simple mechanism for synchronizing access to a variable that is shared by multiple threads.

- Interlocked Topic 155
- Local
- Global
- None

321. Memory allocation is then performed using using _____rather than using_____
- HeapAlloc() and HeapFree(),malloc() and free() Topic 56
- malloc() and free() and HeapAlloc() and HeapFree(),

322. Each thread that performs memory management can create a Handle to its own heap using _____
- HeapCreate() Topic 156
- HeapAlloc()
- HeapFree()
- None of the given

323. Threads allocate memory and free memory using _____functions respectively
- malloc () and free () Topic 156
- free () and malloc ()

324. How may aspects compare the performance on the Basis.
- 1
- 2
- 3 Topic 157
- 4

325. Which of the Following aspects that are used to compare the Performance.
- Real-Time
- User-Time
- System-time
- All of the given Topic 157

326. MX(Mutexes) version costs more than _____ times than IN
- 2 times
- 2 to 30 times Topic 158
- 1 to 30 times
- 3 times

327. CS (Critical Section) version costs _____ times more than IN
- 2 to 30 times
- 2 times Topic 158
- 1 to 30 times
- 3 times

328. For older version of windows CS was not scalable.
- True Topic 159
- False

329. Critical Section works in_____
- Hardware
- User space Topic 161
- Kernel
- BIOS

330. How many methods of Lightweight Reader Writer Locks.
- 1
- 2 Topic 163
- 3
- 4

331. Which of the following methods of Lightweight Reader Writer Locks
- Exclusive Mode
- Shared Mode
- Both A and B Topic 163
- None of given

332. How many APIs which are used to access SRWs in Exclusive mode.
- 1
- 2 Topic 164
- 3
- 4

333. When a thread is created almost _____space is reserved for that thread and with increasing threads memory area is piled up.
- 1 MB Topic 166
- 1 KB
- 1 Bit
- 1 Byte

334. If you_____ the threads the you can lose the benefits of parallelism and synchronization
- Minimize Topic 166
- Maximize

335. Which of the following optimization technique?
- Use of semaphore throttles
- Asynchronous I/O
- Using I/O completion ports
- All of the Given Topic 166

336. Every process has a dedicated_____ pool.
- Odd threads
- Many threads

- Threads Topic 172
- Even threads

337. The thread pool is used to _____the number of application threads and provide management of the worker threads
- Increased
- Reduce Topic 172
- Constant
- Equal

338. How many types of call back Function.
- 1
- 2 Topic 173
- 3
- 4

339. How many sorts of parallelism?
- 1
- 2 Topic 177
- 3
- 4

340. Which of the following parallelism define every loop iteration can execute concurrently.
- Loop parallelism Topic 177
- Fork-join parallelism

341. _____the control flow divides (like the shape of the fork) into multiple flows that join later
- Loop parallelism
- Fork-join parallelism Topic 177

342. Each process has its own process affinity mask and a system affinity mask.
- True Topic 180
- False

343. How many Bits vector of MASK?
- 1
- 2
- 3 Topic 180
- 4

344. Which of the follwing Bits vector of MASK?
- System Affinity Mask
- Process Affinity Mask
- Thread Affinity Mask
- All of the Given Topic 180

345. IPC stand for_____

- Internet process control
- Inter-process Communication Topic 180
- Inter-parameter control
- None of the Given

346. How many Types of Pipe?
- 1
- 2 Topic 181 (Anonymous Pipes, Named Pipes)
- 3
- 4

347. File-like object called Pipe can be used for IPC.
- True Topic 181
- False

348. Which of the following are correct statement of Anonymous Pipes?
- Simple anonymous pipes are character based and half duplex. Topic 181
- They allow network wide communication.

349. Which of the following are correct statement of Named Pipes?
- There can be multiple open handles for a pipe
- They allow network wide communication.
- Much more powerful than anonymous pipes
- All of the Given Topic 181

350. Anonymous pipes allow _____(half-duplex) communication
- one-way Topic 182
- two-way

351. The default set of cbPipe is____
- 0 Topic 182
- 1
- 2
- 3

352. Named Pipe is____
- Directional
- Bi-Directional Topic 184
- Tri-Directional
- None of the Given

353. The pipe name would be like this.
- \\.\pipe\[path]pipename Topic 187
- \\servername\pipe\pipename

354. Which of the following connection sequences for the Client.
- The client connects with a server
- Communicates with the server using CreateFile()

- Performs Read and Write operations and ultimately disconnects
- All of the given Topic 190

355. Which of the following connection sequences for the Server.
- Communicates with the client.
- As a result ReadFile() returns FALSE
- The server-side connection is disconnected
- All of the given Topic 190

356. Which of the following Returns information whether the pipe is in blocking or non-blocking mode, message oriented or byte oriented, number of pipe instances, and so on.
- SetNamedPipeHandleState()
- GetNamedPipeHandleState() Topic 188
- GetNamedPipeInfo()
- None of the given

357. Which of the following Allows the program to set the same state attributes. Mode and other values are passed as reference so that NULL can also be passed indicating no change is desired.
- SetNamedPipeHandleState() Topic 188
- GetNamedPipeHandleState()
- GetNamedPipeInfo()
- None of the given

358. Which of the following Determines whether the handle is for client or a server, buffer sizes, and so on.
- SetNamedPipeHandleState()
- GetNamedPipeHandleState()
- GetNamedPipeInfo() Topic 188
- None of the given

359. Call_____to disconnect from the handle
- DisconnectNamedPipe() Topic 189
- WaitNamedPipe()
- ConnectNamedPipe()
- All of the given

360. _____ is used to synchronize connections to the server
- DisconnectNamedPipe()
- WaitNamedPipe() Topic 189
- ConnectNamedPipe()
- All of the given

361. _____are the security attributes as discussed previously
- lpSecurityAttributes Topic 186
- nOutBufferSize

- dwOpenMode
- nMaxInstance

362. _____and _____give the size in bytes of input and output buffer
- nOutBufferSize and nInBufferSize Topic 186
- nInBufferSize and nOutBufferSize

363. which of the following determines maximum number of pipe instances?
- lpSecurityAttributes
- nOutBufferSize
- dwOpenMode
- nMaxInstance Topic 186

364. which of the following indicates whether writing is message oriented or byte oriented?
- dwPipeMode Topic 186
- nOutBufferSize
- dwOpenMode
- nMaxInstance

365. _____ is the default timeout period.
- nDefaultTimeOut Topic 186
- nOutBufferSize
- dwOpenMode
- nMaxInstance

366. The period (.) stands for local machine.
- True Topic 186
- False

367. In mutex which type of data structure that stores the resource should also be used to store the mutexes because mutexes correspond to the resources.
- Different Topic 154
- Same
- Equal
- None of the Given

368. Only the_____ Scheduler decides which thread has the priority according to its scheduling policy.
- Hardware
- Application
- OS Topic 154
- BIOS

369. The shell uses a …… file keeping track of process ID and other related information?
- **User-specific Topic 104**

370. The array of handles for which the process waits.
- **lpHandle**

371. In Listing Background Jobs the job management function used for the purpose is DisplayJobs(). The function _____ the file.
- Process
- Runs
- Opens Topic 105
- Starts

372. In using Job objects, job objects are used to _____ process execution time and obtain user time statistics.
- Open
- Limit Topic 108
- Read
- White

373. FILETIME is a(an) _____-bit value. GetThreadTime()can be used similarly and required a hread handle.
- 32
- 64 Topic 100
- 16
- 8

374. In Finding a process Id, the FindProcessId() obtains the process Id of given job number. It simply looks up into the File based on the job number and _____ the record at the specific location.
- Executes
- Reads Topic 106
- Writes
- Sums

375. Thread Issues Threads Share Resources Within a_____. One Thread May inadvertently another Threads' Data.
- Process Topic 10
- Threads
- Program
- Code

376. An Environment Block is associated with_____process.
- Some
- One
- Each Topic 97
- No

377. When a fixed size data structure is allocated from a single heap, it reduces_____
- ❖ Fragmentation
- ❖ Errors
- ❖ Memory density
- ❖ Throughput

378. The heapReAlloc() API has _____ parameter(s).
- ❖ 4
- ❖ 1
- ❖ 3
- ❖ 2

379. The heapAlloc() API has _____ parameter(s).
- ❖ 3
- ❖ 4
- ❖ 2
- ❖ 1

380. When a heap (logical structure) is created the memory is _____ allocated at the program.
- ❖ Partially
- ❖ Completely
- ❖ Not directly
- ❖ Directly

381. _____ are the APIs for heap memory allocation.
- ❖ Heapcreate ()and HeapRealloc()
- ❖ Allocheap () and HeapRealloc()
- ❖ HeapAlloc() and HeapRealloc()
- ❖ HeapAlloc() and HeapRealloc()

382. For a non growable heap, the value of dwbytes in heap memory alloction is _____
- ❖ 0*7FEE8
- ❖ 0*7FDD8
- ❖ 0*AAAA8
- ❖ 0*7FFF8

383. _____ is the first step to allocate heap in a program.
- ❖ HeapDestroy()
- ❖ HeapFree()
- ❖ Release and handle
- ❖ Get heap handle

384. The function heapSize() returns the size of a block, or _____ in case failure.
- NULL
- 1
- -1
- 0

385. _____ is used to deallocate the entire heap.
- HeapDestroy()
- HeapFree()
- HeapTruncate()
- HeapDelete()

386. Sorting is performed in the _____
- RootHeap
- RecHeap
- ProcHeap
- NodeHeap

387. _____ stores the root address.
- RootHeap
- RecHeap
- ProcHeap
- NodHeap

388. The NodeHeap maintains a _____
- Data
- Data structure
- Record
- Root

389. There are _____ parameters taken by the HeapCreate() API.
- 3
- 4
- 2
- 1

390. Which of the following is the correct windows API for accessing heap?
- INT GetProcessHeap(VOID)
- VOID GetProcessHeap(HANDLE)
- HANDLE GetProcessHeap(VOID)
- INT*GetProcessHeap(VOID)

391. When a fixed size data structure is allocated from a single heap, it reduces _____
- Memory density

- Errors
- Throughput
- Fragmentation

392. The parameters “flOption” in the HeapCreate() API is a combination of _____ flafs.

- 1
- 2
- 4
- 3

393. While using CreateFileMapping(), _____ allow the mapping object to be secured.

- INVALID_VALUES
- PSECURITY_ATTRIBUTES
- LPSECURITY_ATTRIBUTES
- INVALID_HANDLE_VALUES

394. While using CreateFileMapping(),setting lpMapName to _____ disables the map sharing.

- -1
- NULL
- 0
- 1

395. _____ is the API for file mapping objects.

- Create_File_Mapping()
- CreateFileMapping()
- FileCreateMapping()
- MakeFileMapping()

396. Which of the following are the number of parameters taken by CreateFileMapping()?

- 7
- 6
- 5
- 4

397. The _____-- flag is set to be_____ in the CreateProcess() function, which will determine whether child process will inherit copies of parent open handles.

- blanheritFlag, TRUE
- blnheritHandles, FALSE
- blanheritFlag,FALSE

- bInheritHandles, TRUE

398. IPC stands for_____.

- Information and privacy communication
- Inter privacy communication
- Information process communication
- Inter Process Communication

399. Inherited handles are _____ copies that a parent and child might be accessing.

- Connected
- Similar
- related
- Distinct

400. Process IDs are always _____.

- Frequent
- Repeated
- Constant
- Unique

401. The process obtains environment and other information from_____ call.

- CreateThread()
- GetEnvironmentinfo()
- Getinfo()
- CreateProcess()

402. lpApplicationName handle's value_____ be NULL.

- May not
- May
- should
- Should not

403. In windows there are _____ ways to get command line parameters for a process.

- Five
- Four
- Two
- Three

404. Windows OS does not have structure that keeps track record of the _____ processes.

- Child
- Grand –child
- Parent

- ❖ Parent_Child

405. The most fundamental process management function in windows is CreateProcess() that has_____--- parameters.

- ❖ 12
- ❖ 6
- ❖ 4
- ❖ 10

406. The process can share memory and files but the process itself lie an individual _____ memory space .

- ❖ Non_volatile
- ❖ Physical
- ❖ permanent
- ❖ Virtual

407. Thread Local Storage (TLS) is an array of collection of pointers enabling a thread to _____ storage to create its unique data environment.

- ❖ De-allocate
- ❖ Clear
- ❖ Re-allocate
- ❖ Allocate

408. Each thread has its own_____.

- ❖ TLS
- ❖ Environment Block
- ❖ Stack
- ❖ TLS and Stack

409. The process of DLL detachment in explicit linking is invoke by_____ function call.

- ❖ Free()
- ❖ freeLib()
- ❖ Flibra
- ❖ FreeLibrary()

410. Information regarding DLLs is placed in the _____data structure.

- ❖ dwBuilderNumber
- ❖ dwPlatform
- ❖ MAJORVERSION
- ❖ DLLVERSION

411. LoadLibrary() and LoadLibraryEx() should never be called from_____ as it will create more DLL entry Points.

- ❖ ThreadLibrarycalls()

- DllMinFunc()
- DisableThreadLibraryCalls()
- DllMain()

412. LoadLibraryEx() can suppress the execution of entry point, in _____-- linking of DLL.

- Implicit
- Static
- Dynamic
- Explicit

413. "Application that require newer updated functionality may sometime link with older DLL version". This statement refers to _____ of DLL versioning

- Strength
- Advantages
- Caution
- Problem

414. If entry point of DLL is not specified, then it is an example of _____-- linking.\

- Explicit
- Dynamic
- Hard
- Implicit

415. In case of_____ linking the DLL attaches at the time of process start and detaches when process ends

- Explicit
- Dynamic
- Hard
- Implicit

416. Explicit linking requires the program to explicitly specify the DLL to be _____.

- Freed
- Loaded
- Loaded and freed
- Ans: Loaded or freed

417. In a pointer function declaration for DLL explicit linking, HMODULE is NULL in case of_____.

- Execution
- Waiting
- success

- Failure

418. Once the DLL is loaded, the programmer needs to obtain _____ into the DLL for an entry point.

- Dynamic address
- Physical address
- Bus address
- Procedure Address

419. We write and _____ function in DLL and invoke them explicitly

- Compile
- Encrypt
- decrypt
- Encapsulate

420. In DLLs the executable library files are linked at_____ time

- . Ans: Compile

421. Each DLL program will have its own copy of _____ variables.

- Ans: Globle

422. In_____ operating system DLLs are used to invoke all kernel services.

- Ans: Windows

423. Dynamic memory is allocated from the

- Cache
- Paging file
- Stack
- Static memory

424. Which of the following is recommended to use while dealing with memory mapped file to look for EXCEPTION_IN_PAGE_ERROR exception?

- ESH exception handling
- SHE exception handling
- HE exception handling
- HES exception handling

425. To create a file mapping object, we have to declare _____ maximum parameters>

- 4
- 2
- 6
- 8

426. It is not possible for a system to map a file greater than _____ Into virtual memory space, while using Win32 OS.

- 2GB

- ❖ 3MB
- ❖ 3GB
- ❖ 2MB

427. It is much_____- to sort large data available in memory rather than in files.

- ❖ Harder
- ❖ Costly
- ❖ Unyielding
- ❖ Easier

428. qsort() is a _____ function.

- ❖ Standard library
- ❖ EXE
- ❖ Windows DLL
- ❖ User defined

429. When we create a file mapped object for sorting 1000 numbers in a file recorder will be saved in a/an_____-.

- ❖ Heap
- ❖ Stack
- ❖ Queue
- ❖ Array

430. Which of the following are the number of parameters taken by MapViewFile()?

- ❖ 2
- ❖ 4
- ❖ 3
- ❖ 5

431. While using MapViewOfFile(), which of the following are the three commonly used flags?

- ❖ FILE_WRITE, FILE_READ, AND FILE_ALL_ACCESS
- ❖ FILE_MAP_WRITE, FILE_MAP_READ,AND FILE_MAP_ALL_ACCESS
- ❖ MAP_WRITE,MAP_READ, AND MAP_AL_ACCESS
- ❖ WRITE, READ, AND ALL_ACCESS

432. _____ and _____ specify the starting address of the file from where the mapping starts.

- ❖ High, low
- ❖ dwFileHigh,dwFileLow
- ❖ dwFileOffsetHigh, dwFileOffsetLow
- ❖ dbFileOffsetHigh, dbFileOffsetLow

433. unmapViewOfFile() takes _____argument(s)

- 2
- 0
- 4
- 3

434. _____ Is the API for file mapping objects.

- MakeFileMapping()
- CreateFileMapping()
- FilecreateMapping()
- Create_file_Mapping()

435. While using CreateFileMapping(),_____ refers to the paging file.

- LPSECURITY_ATTRIBUTES
- PSECURITY_ATTRIBUTES
- INVALID_HANDLE_VALUES
- INVALID_VALUES

436. While using CreatFileMapping(),_____ allows the mapping object to be secured.

- LPSECURITY_ATTRIBUTES
- PSECURITY_ATTRIBUTES
- INVALID_HANDLE_VALUES
- INVALID_VALUES

437. While using CreateFileMapping(), setting IpMapName to_____ disables the map sharing.

- 0
- 1
- -1
- NULL

438. DLL stand for

- Direct layout library
- Dynamic link library
- Dynamic layout library
- Direct link library

439. The approach to gather all the source code and library functions after encapsulation into a single executable file, is called as _____

- Process linking
- Static linking
- Dynamic linking
- Thread linking

440. Each DLL program will have its own copy of ____ variables.
- ❖ Global
- ❖ Local
- ❖ Dynamic
- ❖ Static

441. In _____ operating system DLLs are used to invoke all kernel services.
- ❖ Windows
- ❖ Unix
- ❖ Linux
- ❖ Solaris

442. In DDLs the executable library files are linked at ____ time.
- ❖ Link
- ❖ Run
- ❖ Compile
- ❖ Load

443. The entry point in DLL defined structure (DWORD) ____ values.
- ❖ 8
- ❖ 4
- ❖ 2
- ❖ 16

444. ReadFile() and writeFile() functions perform much _____ than memory mapped file processing
- ❖ Slower
- ❖ Faster
- ❖ Convenient
- ❖ Nimble

445. Which of the following controls the paging file?
- ❖ The pager
- ❖ Direct memory access
- ❖ Memory mapped I/o
- ❖ Virtual memory management system

446. While using memory mapped I/O there is/are _____ to manage buffers for repetitive operation on the file operations.
- ❖ Needed
- ❖ Not needed
- ❖ Useful
- ❖ Mandatory

447. In order to make a program more efficient, _____ heap(s) may be required.
- ❖ partial
- ❖ only one
- ❖ several
- ❖ Minimum number of

448. There are _____ parameters taken by the HeapCreate() API.
- ❖ 3
- ❖ 2
- ❖ 1
- ❖ 4

449. The parameter “flOptions” in the HeapCreate() API is a combination of _____ flags.
- ❖ 2
- ❖ 4
- ❖ 3
- ❖ 1

450. A process can have _____ heap(s).
- ❖ Only two
- ❖ At the most one
- ❖ only one
- ❖ Many

451. _____ API is used to create a new heap.
- ❖ createHeap()
- ❖ HeapCreate()
- ❖ BuildHeap()
- ❖ NewHeap()

452. If threads have separate memory space, then it will reduce_____
- ❖ Memory contention
- ❖ Access speed
- ❖ Direct memory access
- ❖ Memory density

453. _____ is an appropriate API to dispose-off a heap handle.
- ❖ shudderHandle()
- ❖ DestroyHandle()
- ❖ DeleteHeap()
- ❖ HeapDestroy()

Virtual University	CS609-System Programming Solved MCQ(S) From FinalTerm Papers BY Arslan	Jan 14, 2018
V-U	For Updated Files Visit Our Site : Www.VirtualUstaad.blogspot.com	Updated.

Final-Term Papers Solved MCQS with Reference

1. DOS command _____ gives the status of the memory and also points out which memory area occupied by which process.

- **mem/d** **PG # 13**
- mem/e
- mem/m
- None of the Given

2.To store each character in keyboard buffer ____ bytes are required.

- **2** **PG # 54**
- 4
- 6
- 8

بری صحبت سے تنہائی بہتر ہے اور تنہائی سے نیک صحبت بہتر ہے

3.The interval timer is used to divide an input frequency.

- **True** **PG #68**
- False

4.The printer interface uses the _____.

- IRQ 6
- **IRQ 7** **PG # 95**
- IRQ 1
- None of given options

5.You can send _____ bits in parallel communication between two PC's at a time.

- 2
- 4
- 6
- **8**

6.In UART the line control register signify the _____.

- Word size and length of stop bits
- parity check and parity type
- Control bit
- **All of given** **PG # 114**

7.The bit _____ of Line control register in UART, if set indicates that the DLL and DLM will act as the divisor register.

- 1
- 3
- 5
- **7** **PG # 114**

8.Int 14H _____ can be used to set the line parameter of the UART or COM port.

- **Service # 0** **PG #119**
- Service # 1
- Service # 2
- Service # 3

9._____ is used to read time from RTC.

- **1A/02H** **PG # 137**
- 1A/03H
- 1A/04H
- 1A/05H

10.The _____ signals are used by various devices to request a DMA operation.

- **DREQ** **PG #186**
- REQ
- DRE
- None of the given

ایماندار کو غصہ دیر سے آتا ہے اور جلدی دور ہو جاتا ہے

11.DMA works in _____ modes.

- 2
- **3** **PG #188**
- 5
- 7

DMA Modes Block Transfer, Single Transfer, Demand Transfer

12.DMA Request register can be used to simulate a DMA request through software in case of _____.

- I/O to memory transfer
- None of the given
- **Memory to memory transfer** **PG #192**
- Memory to I/O transfer

13._____ are the circular division of the disk.

- **Tracks** **PG #201**
- Sectors
- Blocks
- None of given

Tracks are the circular division of the disk and the sectors are the longitudinal division of the disk

14.Blocks are _____.

- **Sectors per track** **PG #202**
- Track per sector
- Cylinder per track
- None of the given

زندگی میں کامیابی کا یہی راز ہے کہ پریشانیوں سے پریشان مت بنو

15.Last ____ bytes of data part in partitioning table are the partition table signature.

- **2** **PG #218**
- 4
- 6
- 8

16.Information about further partitioned drives is available at _____ physical block of extended partition.

- **First** **PG #222**
- Second
- Third
- Fourth

17._____ will be used to get drive parameters.

- Int 13H/42H
- **Int 13H/48H** **PG #238**
- Int 13H/66H
- Int 13H/66H

18.Inside a boot block operating system name occupies _____ bytes.

- 3
- 5
- 8 **Not sure**
- 11

19. A programmer wants to access the DPB(Drive Parameter Block) for C drive using undocumented service 21H/32H, what he should place in DL register _____ ?

- **0** **PG #249**
- 1
- 2
- 3

DL – 0 for current Drive

20. Blocks for FAT can be accessed using _____.

- BPB
- DPB
- **FCB**
- Both BPB and DPB.

21. In main memory smallest addressable unit is ____.

- **Byte**
- Nibble
- Word
- None of the given

22. Using the root directory entry and the _____, we can access the contents of File.

- Reserved Blocks
- Number of FAT copies
- **File Allocation Table(FAT)** **PG # 241**
- None of the given

23.If a file is having more than one cluster then it will be managed by _____.

- FAT
- BPB
- DPB
- **None of the given**

24.Practically in FAT 32, total number of entries are _____.

- 2^{26}
- 2^{28}
- 2^{30}
- **2^{32}**

25._____ keeps the back up of its boot block.

- FAT 12
- FAT 16
- **FAT 32**
- None of the given

26.NTFS can store file name in _____ form.

- **UNICODE** **Click Here For More Detail**
- Both ASCII and UNICODE
- None of the given
- ASCII

27.FAT based file system can store file name in _____ form.

- ASCII
- UNICODE
- **Both ASCII and UNICODE**
- None of the given

28.The LSN of first logical sector of NTFS partition is _____.

- **0** **PG # 240**
- 1
- 2
- 3

29.To minimize _____ defragmenter is used.

- **Seek time** **PG # 316**
- Access time
- Rotational delay
- Both seek and access time

30.In memory map of first 1 MB of RAM, the higher _____ is called system memory.

- 64 KB
- **384 KB** **PG # 317**
- 640 KB
- None of the given

Memory map of the first 1MB of RAM. The first 640KB is called conventional RAM and the higher 384KB is called system memory.

جھوٹ انسان اور ایمان دونوں کا دشمن ہے

31.Extended memory can be accessed in _____ mode.

- Real
- **Protected**
- Both real and protected
- None of the given

32.Extended memory can be accessed only after installing the driver _____ .

- EMM386.EXE
- **HIMEM.SYS** **PG # 318**
- IO.SYS
- None of the given

33.DOS uses _____ memory allocation system.

- **Contiguous** **PG # 325**
- Non Contiguous
- Both Contiguous and Non Contiguous
- None of the given

34.In protected mode segment register will be called _____.

- **Selector** **PG # 326**
- Descriptor
- IDT
- Both Selector and Descriptor

35.There are _____ Control registers in 80386 and above systems.

- 2
- **4** **PG # 331**
- 8
- 16

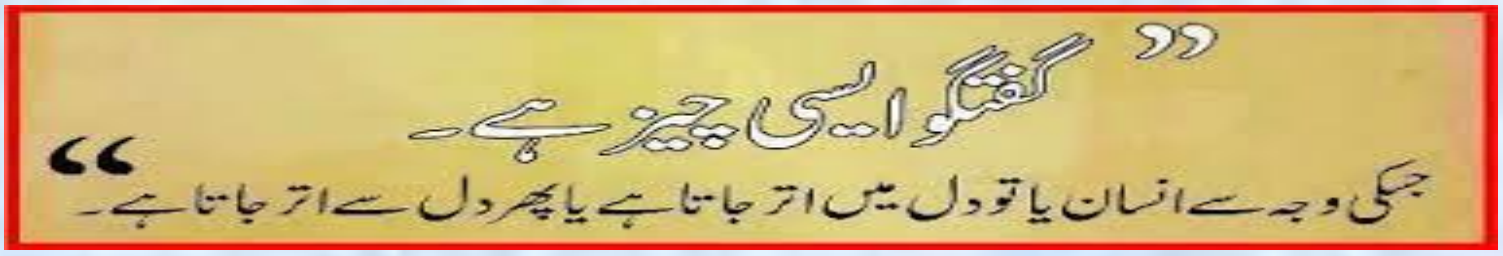
36.If the CHS (cylinder, head, and sector) address of a disk is known it can be translated in to the _____ address and vice versa.

- Logical
- **LBA** **PG # 213**
- Physical
- None of the Given

37._____ is a data structure maintained by DOS in the boot block for each drive.

- DPB
- **BPB** **PG # 242**
- FCB
- None of the Given

BIOS parameter block



38.System Stores the DPBs (Drive Parameter Block) for different drives in the form of _____.

- Chain / link list
- Array
- **Table** **Not sure**
- None of Given

39.The extension of a compiled TSR (Terminate and Study Resident) program should be _____

- .EXE
- **.COM**
- .CPP
- .C

40. A single interrupt controller can arbitrate among ____ different devices.

- 4
- 6
- **8** **PG # 47**
- 10

41.The microprocessor package has many signals for data. Below are some in Correct priority order (Higher to Lower).

- Reset,Hold,NMI,INTR
- **NMI, INTR,Hold,Reset** **PG # 46**
- INTR,NMI,Reset,Hold
- None of the Given

42.A standard PC can have _____PPI.

- 1
- **4** **PG # 84**
- 8
- 16

A standard PC can have 4 PPI named LPT1, LPT2, LPT3 and LPT4

43.BIOS DO NOT support _____.

- LPT1
- LPT2
- LPT3
- **LPT4**

44._____ is used to set date.

- 1A/02H
- 1A/03H
- 1A/04H
- **1A/05H** **PG # 139**

45.If real time clock interrupts microprocessor then interrupt ____ will occur.

- 13h
- 17h
- **70h**
- 71h

46._____ is LED control byte.

- 0xF3
- **0xED** **PG # 181**
- 0xE5
- 0xFF

47.The DMA requests to acquire buses through the _____ signal.

- **HOLD** **PG # 186**
- ACR
- ACK
- All of the given

48._____ will specify if the next DMA transfer will happen as a single transfer, block transfer or demand transfer

- DMA Request register
- DMA Mask register
- **DMA Mode register**
- DMA Command register

49._____ will specify if the channel is to cascade two DMA controllers.

- DMA Request register
- DMA Mask register
- **DMA Mode register**
- DMA Command register

50._____ are the longitudinal division of the disk.

- Tracks
- **Sectors** **PG # 201**
- Blocks
- None of given

Tracks are the circular division of the disk and the sectors are the longitudinal division of the disk

51.Highest physical capacity of the disk according to the IDE interface is_____.

- 504 MB
- 80 GB
- **127 GB** **PG # 212**
- 300 GB

52.In partition table, first _____ bytes contain code which loads the boot block of active partition.

- 128
- 286
- **446** **PG # 218**
- 500

53.On a single disk there can be _____ different Operating Systems.

- 2
- **4** **PG # 219**
- 6
- 8

عقل مند کہتا ہے میں کچھ نہیں جانتا جبکہ بے وقوف کہتا ہے کہ میں سب کچھ جانتا ہوں

54._____ is physical or absolute address.

- **LBA** **PG # 240**
- LSN
- CHS
- None of the given

55.The _____ block is the first block on disk.

- LSN =0
- **LBA=0** **PG # 240**
- Both LBA=0 and LSN=0
- None of the given

56.A directory is collection of _____.

- Bios Parameter Blocks
- Drive Parameter Blocks
- **File Control Blocks**
- None of the given

57.First cluster in user data area is numbered _____ in a FAT based systems.

- 0
- 1
- **2** **PG # 258**
- 3

58.Blocks for FAT can be accessed using _____.

- BPB
- DPB
- **FCB**
- Both BPB and DPB.

59.In FAT12 to calculate the address or offset from index, we need to multiply it with ___.

- 1/2
- **3/2** **PG 267**
- 5/2
- 7/2

60.Using the root directory entry and the _____, we can access the contents of File.

- Reserved Blocks
- Number of FAT copies
- **File Allocation Table(FAT)** **PG #269**
- None of the given

61.In FAT16 if FAT entry is between FFF0H to FFF6H then _____.

- Cluster is available
- **It is a Reserved cluster**
- It is next file cluster
- It is a last file cluster

62._____ needed to store a UNICODE character.

- Nibble
- **2 Bytes**
- 4 Bytes
- Byte

63._____ used for FCB in FAT 12 and FAT 16.

- Nibble
- Byte
- **2 Bytes**
- 4 Bytes

64._____ file system is used in CD's.

- None of the given
- **Contiguous**
- Chained
- Indexed

65.FAT based file system can store file name in _____ form.

- ASCII
- UNICODE
- **Both ASCII and UNICODE**
- None of the given

66.In case of _____ contents, the indexes of clusters of file are stored in MFT.

- Small files
- Large files
- **Both small and large files**
- None of the given

67.In memory map of first 1 MB of RAM, the higher _____ is called system memory.

- 64 KB
- **384 KB**
- 640 KB
- None of the given

Memory map of the first 1MB of RAM. The first 640KB is called conventional RAM and the higher 384KB is called system memory

68.PC's operated on 20 bit address bus can operate in _____ mode.

- **Real** **PG # 319**
- Protected
- Both real and protected
- None of the given

69. Int 21H/52H returns the address of DOS internal data structures in ES: BX, the far address of the first MCB in memory lies _____ behind the address returned.

- 2-bytes
- **4-bytes**
- 6-bytes
- 8-bytes

70. In memory management, descriptor entry is of _____ bytes.

- **64**
- 32
- 8
- 16

71. There are _____ Control registers in 80386 and above systems.

- 2
- **4** **PG # 331**
- 8
- 16

72. Control Registers are used for conveying certain control information for _____ Addressing and Co-Processors.

- Real Mode
- **Protected Mode** **PG # 331**
- Virtual Mode
- None of the given

73.The least significant bit of Control register _____ is PE-bit which can be set to enable Protected Mode Addressing.

- **CR0**
- CR1
- CR2
- CR3

74.CD.. command gives the _____ of directory.

- None of the given
- One previous level
- One next level
- **Current path** **PG #282**

75.Cluster size is variable and can be measured in power of _____.

- **2**
- 16
- 4
- 8

کسی انسان کی خوبی کو پہچانو اور اِسے بیان کرو، لیکن اگر کسی کی
خامی مل جائے تو یہاں تمہاری خوبی کا امتحان ہے۔
فرمان حضرت علیؓ

76.Who notifies the end of interrupt (EOI) to the programmable interrupt controller?

- Programmer
- **Microprocessor**
- Operating system
- None of the given

77.Choose the correct statement to notify the interrupt controller about end of interrupt (EOI).

- Inportb(0x20,0x20)
- **Outportb(0x20,0x20)**
- Outportb(0x30,0x22)
- None of the given

78.Information about Pending interrupts are held by _____.

- Interrupt Service Register
- Interrupt Mask Register
- None of the Given
- **Interrupt Request Register**

79. BPB stands for _____.

- **BIOS parameter block**
- BIOS processing block
- Base processing block
- BIOS partition block

80.The input frequency used by the interval timer is the _____signal generated by the clock generator.

- None of Given
- **PCLK**
- ACL
- ICP

81.BIOS DO NOT support _____.

- LPT1
- LPT2
- LPT3
- **LPT4**

82.DTE is _____.

- **Data terminal equipment** **PG #109**
- Data transmitting equipment
- Dual terminal equipment
- None of the given.

83.Int 14H _____ can be used to send a byte.

- Service # 0
- **Service # 1**
- Service # 2
- Service # 3

84.A single DMA can transfer _____ operands to and from memory in a single a bus cycle.

- **8 bit**
- 16 bit
- 32 bit
- 64 bbit

85.DMA does not work in _____ mode.

- **Double Mode**
- Block Transfer
- Demand Transfer
- None of the given

86.An addressable unit on disk can be addressed by three parameters _____.

- **head #, sector # and track #**
- HD #, AH# and track #
- Hd0 #, sector # and track #
- None of the given

87.Each addressable unit has a unique combination of sec#, head# and track# as its _____ address.

- Both Physical and Logical
- None of the given
- **Physical**
- Logical

جو لوگوں کے سامنے فخر کرتا ہے وہ لوگوں کی نظروں سے گر جاتا ہے

88.In partition table, first _____ bytes contain code which loads the boot block of active partition.

- 128
- 286
- **446**
- 500

89.On a single disk there can be _____ different Operating Systems.

- 2
- **4**
- 6
- 8

90.The last partition table within the chain contains _____ entry/entries about the logical drive.

- **Single**
- Double
- Three
- Four

91.Partition Table can be read using the extended _____ Services.

- 13H
- **14H**
- 15H
- 17H

92.File control block (FCB) is _____ byte long.

- 8
- 16
- **32**
- 64

93._____ is collection of contagious blocks.

- **Cluster**
- Sector
- File
- Directory

94.A FAT 12 table can have maximum _____ values.

- 1024
- 2048
- **4096**
- None of given

2^12 entries maximum = 4096

95.The first cluster number of file can be found in _____.

- BPB
- DPB
- **FCB** **PG # 265**
- None of the given

96.In FAT12 if FAT entry is 000H then _____.

- It is a last file cluster
- Cluster is available
- **It is a Reserved cluster**
- It is next file cluster

97.Total number of clusters of FAT12 are _____.

- FF0 H
- FFF H
- FEF H
- **FEE H** **PG # 266**

98.When we mark a file as deleted by placing 0xE5 then the chain of clusters in FAT is also replaced by _____.

- E5
- **0**
- 1
- N

99.To store a cluster in FAT 32 _____ is/are needed.

- Byte
- 2 Bytes
- **4 Bytes**
- Nibble

100._____ keeps the back up of its boot block.

- FAT 12
- FAT 16
- **FAT 32**
- None of the given

101.Up to _____ bytes can be used to store a file name in NTFS.

- 30
- 126
- **254**
- 510

102.First logical sector of NTFS partition is _____.

- DPB
- MFT
- **Boot sector**
- None of the given

103.The LSN of first logical sector of NTFS partition is _____.

- **0**
- 1
- 2
- 3

104.In case of _____ the contents of file are stored in MFT.

- Small files
- Large files
- **Both small and large files**
- None of the given

105.Interrupt Descriptor Table can have up to _____ entries.

- 64
- 128
- **256**
- 512

106.There can be _____ different descriptors privilege levels.

- 2
- **4**
- 8
- 10

107.The least significant bit of Control register _____ is PE-bit which can be set to enable Protected Mode Addressing.

- **CR0** **PG # 331**
- CR1
- CR2
- CR3

108.Entry point of execution in EXE File can be _____.

- From first instruction
- From last instruction
- **Anywhere in the program** **PG # 335**
- From anywhere in the middle

109.In NTFS, Block # ____ is the safest block to store the back up of boot block.

- 8
- **10** **Not sure**
- 20
- 6

110.Size of single entry in Partition Table is _____.

- **512 bytes**
- 128 bytes
- 64 bytes
- 16 bytes

111.MBR partition table can have _____ entries at maximum.

- 1
- 2
- 3
- **4** **Not sure**

112._____ is a data structure maintained by DOS in the boot block for each drive.

- DPB
- **BPB** **PG # 242**
- FCB
- None of the Given

113.Hidden blocks between the first physical block on each partition are used by _____ for storing of data

- User
- Operating System
- BIOS
- **None of the Given**

114.Which of the following data structure is NOT used in Operating System's File management?

- **Memory Control Block** **PG # 06**
- File Allocation Table
- Drive Parameter Block
- File Control Block

115.Interrupt procedures are _____ procedures.

- **Reentrant** **PG # 38**
- Non-Reentrant
- Recursive
- Virtual

116.The extension of a compiled TSR (Terminate and Study Resident) program should be _____

- .EXE
- **COM**
- .CPP
- .C

to make a .C program a TSR program, you are required to generate a .COM file by using the following command in DOS prompt.

bcc –EEXAMPLE.COM –IC:\BORLANDC\INCLUDE –LC:\BORLANDC\LIB EXAMPLE.C

Here, we are generating EXAMPLE.COM for of EXAMPLE.C program.

117.Choose the correct statement to notify the interrupt controller about end of interrupt (EOI).

- Inportb(0x20,0x20)
- **Outportb(0x20,0x20)**
- Outportb(0x30,0x22)
- None of the given

118.Information about Pending interrupts are held by _____.

- Interrupt Service Register
- Interrupt Mask Register
- **Interrupt Request Register**
- None of the Given

عقل مند اپنے عیب خود دیکھتا ہے اور بیوقوفوں کے عیب دنیا دیکھتی ہے

119. _____ Procedure can have parameters.

- Hardware Interrupt
- Software Interrupt
- **Both Hardware and software Interrupt** **PG # 10**
- None of the Given

120.The Address of partition block on hard disk is _____.

- head # =0, track # = 0 and sector # = 0
- head # =0, track # = 0 and sector # = 1
- head # =0, track # = 1 and sector # = 1
- **head # =1, track # = 0 and sector # = 1** **PG # 42**

121.NMI Stand for _____

- **Non Maskable Interrupt** **PG #46**
- Non Multitude Interrupt
- Non Maskable Instruction
- None of Given

122.The interval timer is used to divide an input frequency.

- **True** **PG # 68**
- False

بدصورت چہرہ بدصورت دماغ سے بہتر ہے

123.The PPI acts as an interface between the CPU and a parallel _____ .

- **I/O device** **PG # 86**
- CPU
- BUS
- None of Given

124.BIOS DO NOT support _____.

- LPT1
- LPT2
- LPT3
- **LPT4**

125._____ is used to identify the cause of interrupt.

- Interrupt Enable register
- **Interrupt ID register**
- Interrupt Status register
- None of the given

126.The bit _____ of Line control register in UART, if cleared will indicate that DLL is the data register.

- 1
- 3
- 5
- **7**

127.Int 14H _____ can be used to send a byte.

- Service # 0
- **Service # 1** **PG # 121**
- Service # 2
- Service # 3

Service of 14h include service #1 which is used to send a byte and service #2 which is used to receive a byte

128.To access bettery powered RAM, only _____ ports are important from programming point of view.

- **70 and 71H** **PG # 141**
- 71 and 72H
- 70 and 72H
- 72 and 73H

129.The DMA requests to acquire buses through the _____ signal.

- **HOLD** **PG # 186**
- ACR
- ACK
- All of the given

130._____ used to program various common parameters of transfer for all the channels.

- None of the above
- DMA Status Register
- **DMA Command Register** **PG # 191**
- DMA Request Register

131.Bit # _____ of mode register in DMA determine the direction of a transfer.

- **2**
- 3
- 4
- 5

132._____ will specify if the next DMA transfer will happen as a single transfer, block transfer or demand transfer.

- DMA Request register
- DMA Mask register
- **DMA Mode register**
- DMA Command register

133.Each addressable unit has a unique combination of sec#, head# and track# as its _____ address.

- Both Physical and Logical
- None of the given
- **Physical** **PG # 202**
- Logical

134.Highest capacity of disk can be accessed using BIOS functions is _____.

- 128 MB
- 256 MB
- **504 MB** **PG # 211**
- 127 GB

عقل مند آدمی اس وقت تک نہیں بولتا جب تک خاموشی نہیں ہو جاتی

135.DOS has built in limit of _____ blocks per cluster.

- 32
- 64
- **128** **PG # 242**
- 256

136.BIOS Parameter block is situated in _____ Block.

- **Boot** **PG # 242**
- Data
- Extended Data
- None of Given

137.The _____ block is the first block on disk.

- LSN =0
- **LBA=0** **PG # 240**
- Both LBA=0 and LSN=0
- None of the given

138.Inside a boot block jump code part occupies _____ byte.

- 5
- 8
- 11
- **3** **PG # 244**

بہترین تجربہ وہ ہے جس سے نصیحت حاصل ہو

139.In boot block BIOS parameter block starts from _____.

- 03H
- 05H
- 08H
- **0BH**

140.In main memory smallest addressable unit is ____.

- **Byte**
- Nibble
- Word
- None of the given

141.Total number of clusters of FAT12 are _____.

- FF0 H
- FFF H
- FEF H
- **FEE H** **PG # 266**

142.In FAT12 to calculate the address or offset from index, we need to multiply it with ___.

- 1/2
- **3/2** **PG# 267**
- 7/2
- 5/2

143.Total number of cluster of FAT16 is _____.

- FFF0 H
- FFFF H
- FFEF H
- **FFEE H** **Page 272**

144.Total ____ fragments can be supported for storing long file names.

- 26
- 28
- **32**
- 48

145._____ used for FCB in FAT 12 and FAT 16.

- Nibble
- Byte
- **2 Bytes**
- 4 Bytes

146.To store a cluster in FAT 32 _____ is/are needed.

- Byte
- 2 Bytes
- **4 Bytes**
- Nibble

147.Practically in FAT 32, total number of entries are _____.

- 2^{26}
- 2^{28}
- 2^{30}
- **2^{32}**

148.First logical sector of NTFS partition is _____.

- DPB
- MFT
- **Boot sector**
- None of the given

149.Extended memory can be accessed in _____ mode.

- Real
- **Protected**
- Both real and protected
- None of the given

150.MCB is a _____ bytes data structure.

- 8
- **16** **PG # 321**
- 32
- 64

151.Real mode does not support _____ memory allocation system.

- **Contiguous**
- Non Contiguous
- Both Contiguous and Non Contiguous
- None of the given

152.There can be _____ different descriptors privilege levels.

- 2
- **4**
- 8
- 10

153.In memory management, descriptor entry is of _____ bytes.

- 8
- 16
- 32
- **64**

154.For virus to propagate itself, it has to intercept interrupt _____.

- 9H
- 11H
- **13H**
- 17H

خوبصورتی علم و ادب سے ہوتی ہے لباس و حسن سے نہیں

155.If virus wants to be in memory independently, it should have its own _____.

- MCB
- PSP
- EB
- **Both MCB and PSP** **PG #334**

156.Size of single entry in Partition Table is _____.

- **512 bytes** **PG#218**
- 128 bytes
- 64 bytes
- 16 bytes

157._____ is a data structure maintained by DOS in the boot block for each drive.

- DPB
- **BPB** **PG#242**
- FCB
- None of the Given

158.Cluster size is variable and can be measured in power of _____.

- **2** **PG#242**
- 16
- 4
- 8

159.Bit # _____ of Eflag is used for alignment check

- 12
- 14
- 15
- **18** **PG # 164**

160.Each addressable unit has a unique combination of sec#, head #, track # as its _____ address.

- **Physical** **PG #202**
- Logical
- Both
- None

161.First cluster in user data is numbered in a FAT based system.

- 0
- 1
- **2** **PG # 258**
- 3

162.BIOS services understand --------.

- **LBA** **PG # 212**
- LSN
- Cluster #
- None

163.The first cluster number of a file can be found in---------

- BPB
- DPB
- **FCB** **PG # 265**
- None

164.The size of FS Info block is

- 64byte
- 128 byte
- 256 byte
- **512 byte** **PG # 300**

165.In NTFS first -------- entries are reserved.

- 4
- 6
- **16** **PG # 303**
- 32

166.In memory map of first 1 MB of ram ,the first -------- is called conventional RAM.

- 64kb
- 384kb
- **640kb** **PG # 317**
- None

167.In memory map of first 1 MB of ram ,the higher -------- is called system memory.

- 64kb
- **384kb** **PG # 317**
- 640kb
- None

168.The -------- of boot block constitutes of BPB.

- Code part
- **Data part** **PG # 242**
- Both
- None

169.Extended BIOS function make use of --------- address

- **LBA** **PG # 212**
- CHS
- LSN
- None

170.LBA address can be used in place of the CHS address.

- **True** **PG # 235**
- False

171.In FAT12, the maximum range of clusters is ……...

- 0 ~ FEFH
- 1~ FEFH
- **2 ~ FEFH** **PG # 266**
- 3 ~ FEFH

172.NTFS volume can be accessed directly in DOS.

- True
- **False** **PG # 310**

173.Each partition information chunk is 16 bytes long and the last two bytes at the end of the partition table data part is the partition table signature whose value should be _____ indicating that the code part contains valid executable code.

- 00AA
- 0055
- 050A
- **AA55** **PG # 219**

174.Service 21H/52H service returns the address of DOS internal data structures in ES: BX _____behind the address returned lies the far address of the first MCB in memory.

- 2-bytes
- **4-bytes** **PG # 322**
- 6-bytes
- 8-bytes

175.80386 can have _____ control registers.

- 2
- 5
- 3
- **4** **PG # 331**

176.The partition table uses the extended _____ service.

- **13H** **PG # 234**
- 14H
- 15H
- 16H

177.The entry point of execution in EXE File can be

- Start of the first instruction
- Start of the last instruction
- **Anywhere in the Program** **PG # 335**
- Can be in the middle of the program

178.Using the____ entry and the FAT we can access the contents of file.

- Reserved blocks
- **Root Directory** **PG # 269**
- Number of FAT copies
- None of the given

179.Control information in files is maintained using

- BPB
- DPB
- **FCB** **PG # 256**
- FPB

180.What will happen if NTFS volume is accessed in DOS?

- Convert it to FAT volume
- Nothing will happen
- **Error of invalid media** **PG # 310**
- None of the given

181.LSN of FS Info block is available at

- BPB
- **FAT**
- Root Directory
- None of the given

خاموشی غصے کا بہترین علاج ہے

182.DOS device drivers do not understand the_____ data structures.

- FAT12
- FAT16
- FAT32
- **NTFS** **PG # 310**

183._____ is a collection of contagious blocks.

- **Cluster** **PG # 242**
- Sector
- Byte
- None of Given

184._____ used to determine the amount of conventional memory interfaced with the processor in kilobytes.

- INT 10 H
- INT 11 H
- **INT 12 H** **PG # 162**
- INT 13 H

185.Bit number _____ of coprocessor control word is the Interrupt Enable Flag.

- **7** **PG #168**
- 8
- 9
- 10

186.To distinguish 486 with Pentium CPUID Test is used.

- **True** **PG # 166**
- False

187.Practically _____ entries are there in FAT 32.

- 2^26
- 2^28
- 2^30
- **2^32** **PG # 265**

188.BPB stands for _____.

- **BIOS parameter block** **PG # 243**
- BIOS processing block
- Base processing block
- BIOS partition block

189.The keyboard input character scan code is received at ___ port.

- **60H** **PG # 179**
- 61H
- 62H
- 63H

190._____ is LED control byte.

- 0xFD
- **0xED** **PG #181**
- 0xFF
- 0xEE

جھوٹ رزق کو کھا جاتا ہے

191._____means typematic rate will be sent in next byte.

- **0xF3** **PG # 180**
- 0xF4
- 0xF5
- 0xF6

192.Keyboard uses port ____ as status port.

- **64H** **PG # 177**
- 66H
- 67H
- 69H

193.The keyboard can perform _____ serial I/O.

- asynchronous
- **synchronous**
- Multiple
- Single

194.Bit number 2 of port 64H Status register used for output buffer full.

- True
- **False**

195.Bit number _____ can declares the parity error of port 64H Status register.

- 4
- 5
- 6
- **7**

196.Bit number _____ of port 64H Status register used for input buffer full.

- **0**
- 1
- 2
- 3

197.Disadvantage of FAT32 is _____.

- Large disk size can be managed in FAT32
- Cluster size is reduced
- Internal fragmentation is reduced
- **Very large table** **PG # 299**

198. Maximum possible entries in FAT12 are _____.

- 1024
- 2048
- **4096** **PG # 264**
- 65536

199. What will be the value of the word located at 1Fh in DPB when number of free clusters on drive is not known?

- 0000H
- 1111H
- **FFFFH** **PG # 250**
- None of the given.

افضل انسان وہ ہے جو اپنی اصلاح کی کوشش کرتا ہے

200.Jump code part contains ____ bytes in boot block.

- **3** **PG # 302**
- 5
- 8
- 11

201. Operating system name contains ____ bytes in boot block.

- 3
- 5
- **8** **PG # 257**
- 11

202.File can be _____ viewed as organization of data.

- Physically
- **Logically** **PG # 256**
- Both logically and physically
- None of the give

203. _____ is used to read a block against its LSN.

- **absread()** **PG # 247**
- abswrite()
- lsnread()
- None of the given

204. File can be_____ viewed as collection of clusters or blocks.

- **Physically** **PG # 256**
- Logically
- Both physically and logically
- None

205. When we talk about FAT based file system, in user data area first cluster number is _____.

- 0
- 1
- **2** **PG # 258**
- None of the given

206. Cluster number can also be referred as block number.

- True
- **False** **PG # 258**

207.To access the block within cluster using BIOS services the cluster number should be converted into _____.

- CHS
- LBA
- **LSN** **PG # 258**
- None of the given

208.What will be the value of DL register when we are accessing C drive using undocumented service 21H/32H?

- 0
- 1
- 2
- **3** **PG # 249**

209.The directory structure of DOS is like _____.

- Array
- **Tree** **PG # 256**
- Linked list
- None of the given

210.Control information about files is maintained using _____.

- BPB
- DPB
- **FCB** **PG # 256**
- FPB

211.When LSN is equal to zero (0), it means _____.

- First block of the disk
- **First block of the logical drive** **PG # 240**
- First block of hidden blocks
- None of the given

212.In FAT32, lower _____ bits are used.

- 26
- **28** **PG # 292**
- 30
- 32

213._____ is relative address with respect to the start of Logical Drive.

- LBA
- **LSN** **PG #240**
- CHS
- None of the given

214.The practical limit of blocks per cluster is _____.

- 32 blocks per cluster
- **64 blocks per cluster** **PG #242**
- 128 blocks per cluster
- 256 blocks per cluster

215.In dos we have limit of _____.

- **128 blocks per cluster** **PG #242**
- 256 blocks per cluster
- 32 blocks per cluster
- 64 blocks per cluster

216.Highest capacity physical capacity of the disk according to the IDE interface is _____.

- **127 GB** **PG # 212**
- 100 GB
- 80 GB
- 300 GB

217.Partition Table can be read using the extended _____ Services.

- **13 H** **PG # 234**
- 14 H
- 15 H
- None of given

218.In Protected Mode, the segment registers are used as _____

- Descriptor
- **Selector** **PG # 326**
- All of the given choices
- None of the given choices

219.To access drive parameter block we use undocumented service _____

- 09H/32H
- 11H/32H
- 17H/32H
- **21H/32H** **PG # 249**

220. _____ is an absolute address relative to the start of physical drive.

- **LBA** **PG # 240**
- LSN
- CHS
- None of the above

221.Boot block consists of _____ bytes.

- 64
- 128
- 256
- **512** **PG # 242**

222.The DMA requests to acquire buses through the _____ signal.

- **HOLD** **PG #186**
- ACR
- ACK
- None of Given

223.The keyboard device writes a code 0xFA on the port 60H to indicate that the _____.

- Input buffer is full
- **Byte has been received properly** **PG # 179**
- Output buffer is full
- None of the given

224.A single DMA can transfer _____ operands to and from memory in a single a bus cycle.

- **8-bits** **PG # 186**
- 16-bits
- 32-bits
- 12-bits

225.In FAT12, to calculate the address or offset from index, we need to multiply it with ___.

- 1/2
- **3/2** **PG #267**
- 5/7
- 7/2

226._____ Register can be used to show that the channel is single transfer, block transfer or demand transfer mode.

- DMA Command register
- DMA Request Register
- DMA Mode Register
- **DMA controller Register** **PG #187-188**

227.When we mark a file as deleted by placing 0xE5 then the chain of clusters in FAT is also replaced by _____.

- E5
- 1
- **0** **PG # 79**
- N

228.Cluster size is reduced in _____.

- FAT12
- FAT16
- **FAT32** Click here for detail
- None of the given

229.In FAT32 _____ root directory entries are there.

- 128
- 256
- 512
- **None of the given** Click here for detail

230.If a file is having more than one cluster then it will be managed by _____.

- FAT
- BPB
- DPB
- **None of the above**

بری صحبت سے تنہائی بہتر ہے اور تنہائی سے نیک صحبت بہتر ہے

231.Internal fragmentation is reduced in _____.

- FAT12
- **FAT16** **Click here for detail**
- FAT32
- None of the given

232.For supporting long file names, _____ fragments can be supported.

- 12
- 20
- 26
- **32**

233.To store a cluster in FAT 32 _____ is/are needed.

- Nibble
- Byte
- 2 Bytes
- **4 Bytes** **Click here for detail**

234.If a file size is 12K and the size of the cluster is 4K then _____ clusters are used for the file.

- 2
- **3**
- 4
- 5

ایماندار کو غصہ دیر سے آتا ہے اور جلدی دور ہو جاتا ہے

235.We can access the contents of File by using the root directory entry and _____.

- Reserved Blocks
- Number of FAT copies
- **File Allocation Table (FAT)** **PG # 269**
- None of the given

236.FAT based file system can store file name in _____ form.

- ASCII
- UNICODE
- **Both ASCII and UNICODE**
- None of the given

237.Drive parameter block is derived from _____.

- FCB
- FAT
- **BPB** **PG # 249**
- CPB

238.We can access Blocks for FAT using _____.

- BPB
- DPB
- **FCB**
- Both BPB and DPB

جھوٹ انسان اور ایمان دونوں کا دشمن ہے

239.If we know the cluster number, we can access the blocks within the cluster using BIOS services directly.

- **True** **PG # 258**
- False

240._____ is an internal data structure of DOS and resides in main memory.

- BPB
- DPB
- CPB
- **None of the given**. **Click here for detail**

241.The size of DPB data structure is _____ bytes.

- 16
- 32
- 64
- **128** **click here for detail**

242.The size of FCB data structure is _____ bytes.

- **16** **Click here for detail**
- 32
- 64
- 128

243.Advantages of FAT32 is/are _____.

- Large disk size can be managed in FAT32
- **Cluster size is reduced** Click here for detail
- Internal fragmentation is reduced
- All of the given

ایماندار کو غصہ دیر سے آتا ہے اور جلدی دور ہو جاتا ہے

244._____ file system keeps the backup of its boot block.

- FAT12
- FAT16
- **FAT32** **Click here for detail**
- None of the given

245.To store a UNICODE character ____ is/are needed.

- Nibble
- Byte
- **2 Bytes** **Click here for detail**
- 4 Bytes

246._____ is the first block on disk.

- LSN =0
- **LBA=0** **PG # 240**
- LBA=1
- Both LBA=0 and LSN=0

247.If FAT entry is between FFF0H to FFF6H in FAT16 then _____.

- Cluster is available
- **It is a Reserved cluster** **PG # 272**
- It is next file cluster
- It is a last file cluster

248.File system used in CD's is_____ file system

- **Contiguous** **Click here for detail**
- Chained
- Indexed
- None

249.A file has 2 clusters and the size of cluster is 4K. What will be the size of file?

- 2K
- **8K**
- 16K
- 32K

250.In NTFS, Backup of boot block is stored at block # _____.

- 2
- 6
- **8**
- 10

251.The interval timer can operate in _____ modes.

- Five
- Seven
- Four
- **Six** **PG # 72**

252.File control block (FCB) is _____ byte long.

- **32** **Click here for detail**
- 64
- 16
- 128

253.DOS command _____ which gives the status of the memory and also points out which memory area occupied by which process.

- **mem/d** **PG # 13**
- mem/e
- mem/m
- None of the given

254.Each entry in the IVT is _____ in size.

- **4-bytes** **PG # 12**
- 6-bytes
- 8-bytes
- 2-bytes

255.The tail value of the keyboard buffer should be examined to get to the _____ of the buffer.

- **Start** **PG # 56**
- End
- Middle
- None of given

256.Usually interrupt procedures are reentrant procedures especially those interrupt procedure compiled using C language compiler are reentrant.

- **True** **PG # 38**
- False

جو شخص ناکامیوں سے ڈر کر بھاگتا ہے کامیابی اُس سے ڈر کر بھاگتی ہے

257._____ is Disk interrupt.

- 10H
- 11H
- **13H PG # 42**
- 14H

258.In parallel communication, the maximum numbers of bits we can send between two computers are _____.

- 2-bits
- 4-bits
- 6-bits
- **8-bits**

259.14h include _____ which is used to send a byte.

- Service #0
- **Service #1 PG # 121**
- Service #2
- Service #3

260.The status register _____ is the main control register.

- **B PG # 146**
- A
- C
- D

261._____ is used to identify the cause of interrupt.

- **Interrupt ID Register PG # 116**
- PC Register
- AC Register
- None of All These

262.In NTFS, up to _____ characters are used to store files names,

- 30
- 48
- **255** **PG # 283**
- 510

263.A cluster is a collection of contiguous _____.

- **Blocks** **PG # 242**
- Sectors
- Bytes
- None of Given

264.In BPB, root directory is saved in _____.

- **Cluster#0**
- Cluster#1
- Cluster#2
- Cluster#3

265.In NTFS, total sizes of MFT entries are _____.

- **16-bytes** **PG # 303**
- 20-bytes
- 26-bytes
- 32-bytes

266.In NTFS, _____ store the contents of file.

- **Both small & large file Record**
- Small record
- Large Record
- None of given

267.In NTFS, contents and indexed of file is stored in _____.

- Small record
- Large Record
- Both small & large file Record
- **None of given**

268.Total No. of bytes that can be stored in Keyboard Buffer is____.

- 16
- **32** **PG #54**
- 64
- 128

269.BIOS support _____UARTS as COM ports.

- 6
- **4** **PG #113**
- 3
- 2

270.DCE stands for _____.

- **Data communication equipment** **PG # 109**
- Distributed Computing Environment
- Data Communications Equipment
- Data Carrier Equipment

271.In counter register bit no. 3 changes its value between 0 and 1 with in ____clock cycles

- 1
- 2
- 4
- **16** **PG #69**

272.In _____each byte is needed to be encapsulated in start and end.

- Synchronous communication
- **Asynchronous communication** **PG # 106**
- Both
- None of given

273.The _____ service # is not used in any interrupt.

- 01
- 02
- 03
- **FF**

274.If we want to send printing on the printer then we have to perform following steps.

- Initialize printer
- Read Status
- Check Error
- **All of the given**

275.If printer is _____ then printer sends back the ACK signal to the printer interface

- **idle** **PG # 97**
- busy
- Out of paper
- None of the given

276.DSR stands for _____.

- **Data set ready** **PG # 111**
- Data service ready
- Data stock ready
- None of the given

277.At IRQ 7 Interrupt # ___ is used.

- 0x0A
- 0x0B
- 0x0C
- **0x0F PG # 95**

278.The memory addresses of COM ports remain same for all computers

- True
- **False**

279.In keyboard status byte bit no. 2 and 3 are used for ctrl and alt keys respectively. which of the following condition is used to check that Ctrl + Alt keys are pressed. Where: unsigned char far * scr = (unsigned char far *)(0x00400017);

- **if (((*scr)&12)==12)**
- if (((*scr)&8)==8)
- if (((*scr)&4)==4)
- if (((*scr)&2)==2)

280.In case of synchronous communication a timing signal is required to identify the start and end of a bit.

- **True PG # 105**
- False

281.The baud rate is set in accordance with the divisor value loaded within the UART internal registers base +0 and base +1.

- **True PG # 114**
- False

عقل مند آدمی اس وقت تک نہیں بولتا جب تک خاموشی نہیں ہو جاتی

282.Software based flow control make use of -------- control characters

- Xon
- XOFF
- **Both** **PG # 135**
- None

283._____ is used to read time from RTC

- **1A\02H** **PG # 137**
- 1A\03H
- 1A\04H
- 1A\05H

284.Int_____ service 0 can be used to set the line parameter of the UART or COM port.

- **14H** **PG # 119**
- 15H
- 13H
- None of the given option

285.When LBA is equal to zero (0), it means _____.

- **First block of the disk** **PG # 240**
- First block of the logical drive
- First block of the hidden block
- None of the given

286.In IRQ2 and IRQ3 which one has the highest priority?

- Can't be determined
- Both have same priority
- IRQ3
- **IRQ2** **PG # 47**

287.Following is not a method of I/O

- Programmed I/O
- Interrupt driven I/O
- **Hardware Based I/O** **PG # 4**
- None of given

288.It is possible to perform I/O operations from three different methods.

- **True** **PG # 7**
- False

289.The Function of I/O controller is to provide _____.

- I/O control signals
- Buffering
- Error Correction and Detection
- **All of given** **PG # 5**

290.Which of the following are types of ISR _____.

- BIOS (Basic I/O service) ISR
- DOS ISR
- ISR provided by third party device drivers
- **All of the given** **PG # 13**

291.Interrupt service number is usually placed in _____ register.

- CH
- CL
- **AH** **PG # 26**
- AL

خوبصورتی علم و ادب سے ہوتی ہے لباس و حسن سے نہیں

292.NMI Stand for

- **Non Maskable Interrupt** **PG # 46**
- Non Multitude Interrupt
- Non Maskable Instruction
- None of Given

293.A single interrupt controller can arbitrate among ____ different devices.

- 4
- 6
- **8** **PG # 47**
- 10

294. The microprocessor package has many signals for data. Below are some incorrect priority order (Higher to Lower).

- **Reset,Hold,NMI,INTR** **PG # 46**
- NMI, INTR,Hold,Reset
- INTR,NMI,Reset,Hold
- None of the Given

295.The following command "outportb (0x61,inportb(0x61) & 0xFC);" will

- Turn on the speaker
- **Turn off the speaker** **PG # 75**
- Toggle the speaker
- None of the given

296.The following command "outportb (0x61,inportb(0x61) | 3);" will _____ .

- **Turn on the speaker** **PG # 74**
- Turn off the speaker
- Toggle the speaker
- None of the above

297.The PPI acts as an interface between the CPU and a parallel _____ .

- **I/O device** **PG # 83**
- CPU
- BUS
- None of Given

298.BIOS DO NOT support _____.

- LPT1
- LPT2
- LPT3
- **LPT4** **PG # 91**

299._____ bit is cleared to indicate the low nibble is being sent.

- D1
- D2
- D3
- **D4** **PG # 104**

300.The bit _____ of Line control register in UART, if cleared will indicate that DLL is the data register.

- 1
- 3
- 5
- **7** **PG # 114**

301._____ file system is used in NTFS based systems.

- Contiguous Chained
- **Indexed**
- None of the given

302.Communication between keyboard and keyboard controller is _____.

- Asynchronous serial
- **Synchronous serial** **PG # 77**
- Parallel communication
- None of the given

303.In NTFS, boot sector is stored at

- First and 6th sector
- First and Last sector
- Only at Last sector
- **Only at First sector**

304.Standard PC operates in two modes in terms of memory which are

- Real mode and Extended Mode
- Base mode and Memory Mode
- None of the given
- **Real mode and protected mode** **PG # 6**

305.IVT is a table containing _____ byte entries each of which is a far address of an interrupt service routine.

- 2
- **4** **PG # 20**
- 8
- 16

306.Each paragraph in keep function is ____ bytes in size.

- 4
- 8
- **16** **PG # 24**
- 32

307.A software interrupt does not require EOI (End of interrupt.

- **True** **PG # 49**
- False

308.To store each character in keyboard buffer ____ bytes are required.

- **2** **PG # 54**
- 4
- 6
- 8

309.Interrupt _____ is empty; we can use its vector as a flag.

- 9H
- 13H
- 15H
- **65H** **PG # 65**

310.Command register is an _____ bit register

- 4
- **8** **PG # 71**
- 16
- 32

311.How many bytes can be used to store a file name in NTFS?

- 128
- **255**
- 510
- 1024

ہر چیز کی ایک پہچان ہوتی ہے اور عقلمند کی پہچان غوروفکر کرنا ہے اور غوروفکر کی پہچان خاموشی ہے

312.____ is the first logical sector of NTFS partition.

- DPB
- MFT
- **Boot sector**
- None

313.In boot block BIOS parameter block starts from 03H

- 05H
- 08H
- **0BH** **PG # 302**

314.IN NTFS, FAT and root directory is replaced by

- FCB
- **MFT** **PG # 301**
- Hidden blocks
- Boot sector

315.Block # 2 is the safest block to store the backup of boot block.

- True
- **False**

316.The keyboards interface as discussed earlier uses the IRQ0 and the port 64H as data port.

- True
- **False**

317.FAT12 will have 12-bit wide entries and can have 2^12=4096 entries maximum

- **True**
- False

318.In order to produce the sound from PC internal Speaker we have to load the___bit divisor value at the ___port.

- 8, 0x21
- **16, 0x42**
- 32, 0x22
- 64, 0x32

319.DMA stands for_____

- **Direct Memory Access** **PG # 4**
- Distinct Memory Access
- Direct Module Access
- Direct Memory Allocation

320.UART stands for_____

- **Universal Asynchronous Receiver Transmitter** **PG # 107**
- Universal Adjustment and Realigning Tool
- Unconventional Assisted Recovery Team
- None of these

321.Interrupt Vector Table (IVT) in short is a _____ bytes sized table.

- **1024** **PG # 10**
- 2048
- 3072
- 4096

322.Hardware Interrupts are _____.

- Preemptive
- **Non-Preemptive** **PG # 48**
- Both Preemptive and Non-Preemptive
- None of Given

323.Timer interrupt is a _____.

- **Hardware Interrupt** **PG # 28**
- Software Interrupt
- Both of these
- None of These

324.The keyboard makes use of interrupt number _____ for its input operations.

- **9** **PG # 34**
- 10
- 11
- 12

325.Register can be used to divide frequency is _____

- **Counter Register** **PG # 69**
- Accumulator Register
- None of these

326.Which port is known as Data Port_____

- **60H** **PG # 177**
- 61H
- 64H
- 69H

327.LPTs can be swapped.

- **True** **PG # 92**
- False

328.PPI is used to perform parallel communication

- **True** **PG # 81**
- False

329._____is used to control the printer via the BIOS

- Int 16H
- **Int 17H** **PG # 84**
- Int 18H
- Int 19H

330.There are two main types of interrupts namely _____.

- PC based and Window based
- Hardware based and Kernal based
- **Hardware interrupts and Software interrupts** **PG # 10**
- None of the given

331.To set the interrupt vector means is to change the double word sized interrupt vector within the IVT.

- **True** **PG # 22**
- False

332.The service number is usually placed in the _____ register.

- AL
- CL
- **AH** **PG # 26**
- AX

333.The keyboard makes use of interrupt number _____ for its input operations.

- **9** **PG # 34**
- 10
- 11
- 12

ایماندار کو غصہ دیر سے آتا ہے اور جلدی دور ہو جاتا ہے

334.The service _____ is called the keyboard hook service.

- 15H/2FH
- **15H/4FH** **PG # 44**
- 15H/FFH

335.The BIOS interrupt _____ can be used to configure RTC.

- **1AH** **PG # 136**
- 2AH
- 3AH
- 4AH

336.PPI stands for

- Parallel Programmable interface
- **Peripheral Programmable interface** **PG # 76**
- Port Programmable interface
- None of the given

337.Int _____ is used to control the printer via the BIOS.

- **17H** **PG # 84**
- 18H
- 20H
- 21H

338.Counter register can be used to divide clock signal.

- **True** **PG # 69**
- False

339.The bit number _____ of the coprocessor control word is the interrupt enable flag.

- **7** **PG # 168**
- 8
- 9
- 6

340.There are _____ kinds of serial communication.

- **2** **PG # 105**
- 3
- 4
- 5

341._____ store the base address for LPT1.

- 40:00H
- 40:02H
- **40:08H** **PG # 92**
- 40:1AH

342.The amount of memory above conventional memory (extended memory can be determined using the service _____.

- **15H/88H** **PG # 162**
- 16H/88H
- 17H/88H
- 21H/88H

343.The output on the monitor is controlled by a controller called _____within the PC.

- **Video controller** **PG # 30**
- Bus controller
- Ram controller
- None of the given

344.Interrupt 9 usually reads the _____ from keyboard.

- ASCII code
- **Scan code** **PG # 34**
- Both ASCII and Scan code
- None of the above

345.NMI Stand for

- **Non Maskable Interrupt** **PG # 46**
- Non Multitude Interrupt
- Non Maskable Instruction
- None of Given

346.A single interrupt controller can arbitrate among ____ different devices.

- 4
- 6
- **8** **PG # 47**
- 10

347.The microprocessor package has many signals for data. Below are some in Correct priority order (Higher to Lower).

- **Reset,Hold,NMI,INTR** **PG # 46**
- NMI, INTR,Hold,Reset
- INTR,NMI,Reset,Hold
- None of the Given

348.The _____function initialize the COM port whose number is passed as parameter using BIOS services.

- Initializecom()
- **Initialize()** **PG # 125**
- Recievechar()
- None of these option

349.There are two type of communication synchronous and Anti Synchronous

- True
- **False** **PG # 105**

350.REGS is a Union

- **True**
- False

351.Keyboard Status Byte is located at the address

- 0040:0000H
- 0040:0013H
- 0040:0015H
- **0040:0017H** **PG # 29**

352.If we use keep (0, 1000) in a TSR program, the memory allocated to it is

- 64000 bytes
- 32000 bytes
- **16000 bytes**
- 80000 bytes

353.Maximum number of interrupts in a standard PC is

- 64
- 128
- **256** **PG #1 0**
- 512

354.The -------- function receive a byte and COM port number is passed as parameter using BIOS service

- Receivebyte ();
- Receive ();
- **Receivechar ();** **PG # 125**
- None of the given option

355._____ whenever receive indicates the start of communication …….. whenever receive indicates the end of communication

- **XON\XOFF** **PG #135**
- XOFF\XON
- XON\YOFF
- YON\XOFF

356._____ is used to set time from RTC

- 1A\02H
- **1A\03H** **PG #138**
- 1A\04H
- 1A\05H

357.Set the Interrupt vector means to change the double word sized interrupt vector within IVT.

- **True** **PG #22**
- False

358.If keyboard buffer is empty the head and tail points at the same location.

- **True** **PG #55**
- False

359.Standard PC can have _____ PPI.

- 1
- **4** **PG #84**
- 8
- 16

360.By cascading two DMAs ____ bits can be transferred.

- 4
- 8
- **16** **PG #186**
- 32

361.PPI interconnection _____ bits is cleared to indicate low nibble is being sent.

- D1
- D2
- D3
- **D4** **PG # 101**

362.Display device (Monitor) performs _____ **I**/O.

- **memory mapped** **PG #30**
- Isolated
- Both of above
- None of these

363.Timer interrupt occurs _____ times every second by means of hardware.

- **18.2** **PG # 28**
- 16.2
- 15.2
- 14.2

364.An I/O device cannot be directly connected to the busses so controller is placed between CPU and I/O.

- **True** **PG # 83**
- False

365.Tail of keyboard should get to get the start of buffer.

- **True** **PG # 55**
- False

366.____ No. of bytes are used to store the character in the keyboard buffer.

- 1
- **2** **PG # 54**
- 4
- 8

367.We have set the bit No. 7 of IMR(Interrupt Mask Register) to unmask the Interrupt so that interrupt _____ can occur at ____ line.

- **0xf ,IRQ 7**
- 0xa, IRQ 6
- 0x8, IRQ 5
- 0x6, IRQ 2

368.If we want to produce the grave voice from speaker phone then we have to load the ____ divisor values at Port ____.

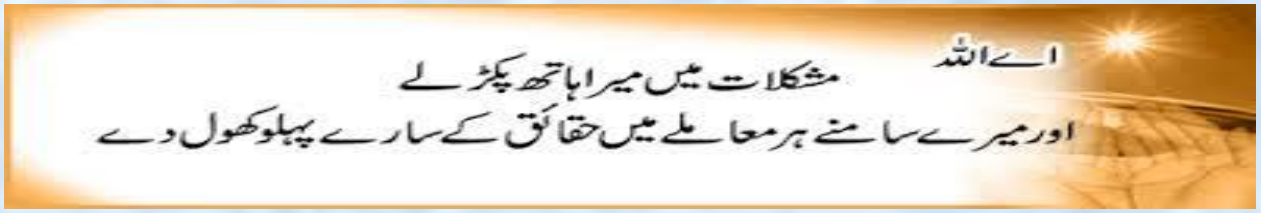
- high, 0x42
- low, 0x22
- high, 0x22
- **low, 0x42**

369.On the execution of IRET instruction, number of bytes popped from stack is

- 4 bytes
- 6 bytes
- **8 bytes** **Click here for detail**
- 10 bytes

370.If CPUID instruction is not present then the processor can be a

- **486 processor** **PG # 166**
- 386 processor
- 286 processor
- All of the above



371.Extended memory is available if the processor is of the type _____

- **AT** **PG # 171**
- XT
- All of the given choices
- None of them

372.The built in mechanism within the UART for error detection is_____

- hamming code
- **parity** **PG # 107**
- CRC16 (cyclic redundancy check 16 bit)
- CRC32 (cyclic redundancy check 32 bit)

373.If three Programmable interrupt controllers are cascaded then how many interrupt driven hardware IO devices can be attached _____

- 12
- 18
- 23
- **24** **PG #48**

374.Int 14H_____ can be used to send a byte

- Service#0
- **Service#1** **PG #121**
- Service#2
- None of the given option.

375.Int 14H _____ can be used to set the line parameter of the UART or COM port.

- **Service # 0** **PG # 119**
- Service # 1
- Service # 2
- None of the given options

376.Int 14H _____ can be used to receive a byte.

- Service # 0
- Service # 1
- **Service # 2** **PG # 121**
- None of the given options

377.The _____ function simply enables the self test facility within the modem control register

- STOn()
- SelfTest()
- **SelfTestOn()** **PG # 127**
- None of these

378._____ is a device incorporated into the PC to update time even if the computer is off.

- Clock counter
- ROM
- Clock
- **Real time clock** **PG # 136**

379.Interrupt _____ is used to get or set the time.

- 0AH
- **1AH** **PG #136**
- 2AH
- 3AH

380._____ is used to set time.

- 1A/02H
- **1A/03H** **PG # 138**
- 1A/04H
- 1A/05H

381.---------- is used to read date from RTC

- 1A\02H
- 1A\03H
- **1A\04H** **PG # 138**
- 1A\05H

382.____whenever received indicates the start of communication and _____ whenever received indicates a temporary pause in the communication.

- **XON & XOFF** **PG # 135**
- XOFF & XON
- XON & YOFF
- YON & XOFF

383.The_____ function uses the COM port number to receive a byte from the COM port using BIOS services.

- recievebyte()
- initialize ()
- receive()
- **recievechar()** **PG # 125**

384.In self test mode the output of the UART is routed to its input.

- **True** **PG # 117**
- False

385.Only _____ ports are important from programming point of view.

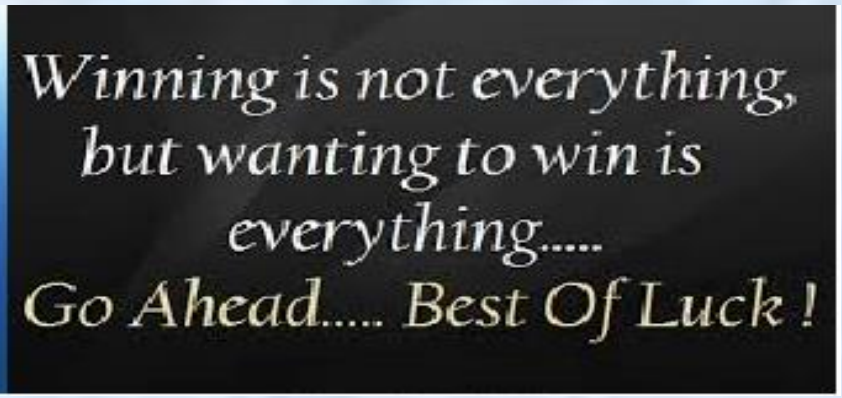
- **70 and 71H** **PG # 141**
- 71 and 72H
- 70 and 72H
- 72 and 73H

386.The BIOS interrupt 0x1AH can be used to configure real time clock

- **True** **PG # 136**
- False

Note: Give me a feedback and your Suggestion also If you find any mistake in mcqz plz inform me Viva Contact us Page on our Site. And tell me your answer with references.

For More Solved Papers By Arslan Visit Our Website:
Www.VirtualUstaad.blogspot.com



FINALTERM EXAMINATION
Spring 2012
CS609- System Programming

Question No: 1 (Marks: 1) - Please choose one
Bit # _____ of Eflag is used for alignment check

►12
►14
►15
►**18 (page 164)**

Question No: 2 (Marks: 1) - Please choose one
Each addressable unit has a unique combination of sec#, head #, track # as its --------- address.

►**Physical (page 202)**
►Logical
►Both
►None

Question No: 3 (Marks: 1) - Please choose one
First cluster in user data is numbered in a FAT based system.

►0
►1
►**2 (page 258)**
►3

دنیا میں سب سے مشکل کام اپنی اصلاح اور سب سے آسان کام دوسروں پر نکتہ چینی کرنا ہے

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question No: 4 (Marks: 1) - Please choose one
BIOS services understand --------.

►**LBA (page 212)**
►LSN
►Cluster #
►None

Question No: 5 (Marks: 1) - Please choose one
The first cluster number of a file can be found in---------

►BPB
►DPB
►**FCB(page 265)**
►None

Question No: 6 (Marks: 1) - Please choose one
The size of FS Info block is

►64byte
►128 byte
►256 byte
►**512 byte(page 300)**

Question No: 7 (Marks: 1) - Please choose one
In NTFS first -------- entries are reserved.

►4
►6
►**16 (page 303)**
►32

Question No: 8 (Marks: 1) - Please choose one
In memory map of first 1 MB of ram ,the first -------- is called conventional RAM.

►64kb
►384kb
►**640kb (page 317)**
►None

بری صحبت سے تنہائی بہتر ہے اور تنہائی سے نیک صحبت بہتر ہے

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question No: 9 (Marks: 1) - Please choose one
In memory map of first 1 MB of ram ,the higher -------- is called system memory.

►64kb
►**384kb (page 317)**
►640kb
►None

Question No: 10 (Marks: 1) - Please choose one
The -------- of boot block constitutes of BPB.

►Code part
►**Data part (page 242)**
►Both
►None

Question No: 11 (Marks: 1) - Please choose one
Extended BIOS function make use of --------- address

►**LBA (Page 212)**
►CHS
►LSN
►None

Question No: 12 (Marks: 1) - Please choose one
LBA address can be used in place of the CHS address.
►**True (Page 235)**
►**False**

Question No: 13 (Marks: 1) - Please choose one
In FAT12, the maximum range of clusters is ……..

►0 ~ FEFH
►1~ FEFH
►**2 ~ FEFH (Page 266)**
►3 ~ FEFH

Question : 14 (Marks: 1) - Please choose one
NTFS volume can be accessed directly in DOS.

►True
►**False (Page 310)**

Question No: 15 (Marks: 1) - Please choose one
Each partition information chunk is 16 bytes long and the last two bytes at the end of the partition table data part is the partition table signature whose value should be _____ indicating that the code part contains valid executable code.

►00AA
►0055
►050A
►AA55 (Page 219)

Question No: 16 (Marks: 1) - Please choose one
Service 21H/52H service returns the address of DOS internal data structures in ES: BX _____behind the address returned lies the far address of the first MCB in memory.

►2-bytes
►4-bytes (Page 322)
►6-bytes
►8-bytes

Question No: 17 (Marks: 1) - Please choose one
80386 can have _____ control registers.

►2
►5
►3
►4 (Page 331)

Question No: 18 (Marks: 1) - Please choose one
The partition table uses the extended _____ service.
►13H (p234)
►14H
►15H
►16H

Question No: 19 (Marks: 1) - Please choose one
The entry point of execution in EXE File can be

►Start of the first instruction
►Start of the last instruction
►Anywhere in the Program (Page 335)
►Can be in the middle of the program

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 20 (Marks: 1) - Please choose one
Using the____ entry and the FAT we can access the contents of file.

►Reserved blocks
►**Root Directory (Page 269)**
►Number of FAT copies
►None of the given

Question : 21 (Marks: 1) - Please choose one
Control information in files is maintained using
►BPB
►DPB
►**FCB (Page 256)**
►FPB

Question : 22 (Marks: 1) - Please choose one
What will happen if NTFS volume is accessed in DOS?

►Convert it to FAT volume
►Nothing will happen
►**Error of invalid media (Page 310)**
►None of the given

Question : 23 (Marks: 1) - Please choose one
LSN of FS Info block is available at

►BPB
►**FAT**
►Root Directory
►None of the given

Question : 24 (Marks: 1) - Please choose one
DOS device drivers do not understand the_____ data structures.

►FAT12
►FAT16
►FAT32
►**NTFS (Page 310)**

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 25 (Marks: 1) - Please choose one
_____ is a collection of contagious blocks.

►**Cluster (Page 242)**
►Sector
►Byte
►None of Given

Question No: 26 (Marks: 1) - Please choose one
_____ used to determine the amount of conventional memory interfaced with the processor in kilobytes.

►INT 10 H
►INT 11 H
►**INT 12 H (Page 162)**
►INT 13 H

Question No: 27 (Marks: 1) - Please choose one
Bit number _____ of coprocessor control word is the Interrupt Enable Flag.
►**7 (Page 168)**
►8
►9
►10

Question No: 28 (Marks: 1) - Please choose one
To distinguish 486 with Pentium CPUID Test is used.
►**True (Page 166)**
►False

Question : 29 (Marks: 1) - Please choose one
Practically _____ entries are there in FAT 32.
►2^26
►2^28
►2^30
►**2^32 (Page 265)**

Question No: 30 (Marks: 1) - Please choose one
BPB stands for _____.

►**BIOS parameter block (Page 243)**
►BIOS processing block
►Base processing block
►BIOS partition block

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question No: 31 (Marks: 1) - Please choose one
The keyboard input character scan code is received at ___ port.

►**60H (Page 179)**
►61H
►62H
►63H

Question No: 32 (Marks: 1) - Please choose one
_____ is LED control byte.

►0xFD
►**0xED (Page 181)**
►0xFF
►0xEE

Question : 33 (Marks: 1) - Please choose one
_____means typematic rate will be sent in next byte.

►**0xF3 (Page 180)**
►0xF4
►0xF5
►0xF6

Question : 34 (Marks: 1) - Please choose one
Keyboard uses port ____ as status port.

►**64H (Page 177)**
►66H
►67H
►69H

Question : 35 (Marks: 1) - Please choose one
The keyboard can perform _____ serial I/O.

►asynchronous
►**synchronous**
►Multiple
►Single

ایماندار کو غصہ دیر سے آتا ہے اور جلدی دور ہو جاتا ہے

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 36 (Marks: 1) - Please choose one
Bit number 2 of port 64H Status register used for output buffer full.

►True
►**False**

Question : 37 (Marks: 1) - Please choose one
Bit number _____ can declares the parity error of port 64H Status register.

►4
►5
►6
►**7**

Question : 38 (Marks: 1) - Please choose one
Bit number _____ of port 64H Status register used for input buffer full.

►**0**
►1
►2
►3

زندگی میں کامیابی کا یہی راز ہے کہ پریشانیوں سے پریشان مت بنو

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

FINALTERM EXAMINATION
Spring 2010
CS609- System Programming

Question No: 1 (Marks: 1) - Please choose one
Maximum possible entries in FAT12 are _____.

►1024
►2048
►4096 (Page 264)
►65536

Question No: 2 (Marks: 1) - Please choose one
Disadvantage of FAT32 is _____.

►Large disk size can be managed in FAT32
►Cluster size is reduced
►Internal fragmentation is reduced
►Very large table (Page 299)

Question No: 3 (Marks: 1) - Please choose one
What will be the value of the word located at 1Fh in DPB when number of free clusters on drive is not known?

►0000H
►1111H
►FFFFH (Page 250)
►None of the given.

Question No: 4 (Marks: 1) - Please choose one
Jump code part contains ____ bytes in boot block.

►3 (Page 302)
►5
►8
►11

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 5 (Marks: 1) - Please choose one
Operating system name contains ____ bytes in boot block.

►3
►5
►8 (Page 257)
►11

Question No: 6 (Marks: 1) - Please choose one
File can be _____ viewed as organization of data.

►Physically
►Logically (Page 256)
►Both logically and physically
►None of the give

Question No: 7 (Marks: 1) - Please choose one
_____ is used to read a block against its LSN.

►absread() (Page 247)
►abswrite()
►lsnread()
►None of the given

Question : 8 (Marks: 1) - Please choose one
File can be_____ viewed as collection of clusters or blocks.

►Physically (Page 256)
►Logically
►Both physically and logically
►None

Question No: 9 (Marks: 1) - Please choose one
When we talk about FAT based file system, in user data area first cluster number is _____.

►0
►1
►2 (Page 258)
►None of the given

جھوٹ انسان اور ایمان دونوں کا دشمن ہے

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question No: 10 (Marks: 1) - Please choose one
Cluster number can also be referred as block number.
►True
►**False (Page 258)**

Question : 11 (Marks: 1) - Please choose one
To access the block within cluster using BIOS services the cluster number should be converted into _____.

►CHS
►LBA
►**LSN (Page 258)**
►None of the given

Question : 12 (Marks: 1) - Please choose one
What will be the value of DL register when we are accessing C drive using undocumented service 21H/32H?

►0
►1
►2
►**3 (Page 249)**

Question No: 13 (Marks: 1) - Please choose one
The directory structure of DOS is like _____.

►Array
►**Tree (Page 256)**
►Linked list
►None of the given

Question No: 14 (Marks: 1) - Please choose one
Control information about files is maintained using _____.
►BPB
►DPB
►**FCB (Page 256)**
►FPB

Question No: 15 (Marks: 1) - Please choose one
When LSN is equal to zero (0), it means _____.

► First block of the disk
► **First block of the logical drive (Page 240)**
►First block of hidden blocks
► None of the given

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question No: 16 (Marks: 1) - Please choose one
In FAT32, lower _____ bits are used.
► 26
► 28 (Page 292)
►30
►32

Question No: 17 (Marks: 1) - Please choose one
_____ is relative address with respect to the start of Logical Drive.
►LBA
►LSN (Page 240)
►CHS
►None of the given

Question No: 18 (Marks: 1) - Please choose one
The practical limit of blocks per cluster is _____.

►32 blocks per cluster
►64 blocks per cluster (Page 242)
►128 blocks per cluster
►256 blocks per cluster

Question No: 19 (Marks: 1) - Please choose one
In dos we have limit of _____ .
►128 blocks per cluster (Page 242)
►256 blocks per cluster
►32 blocks per cluster
►64 blocks per cluster

Question No: 20 (Marks: 1) - Please choose one
Highest capacity physical capacity of the disk according to the IDE interface is _____.
►127 GB (Page 212)
►100 GB
►80 GB
►300 GB

Question No: 21 (Marks: 1) - Please choose one
Partition Table can be read using the extended _____ Services.

►13 H (Page 234)
►14 H
►15 H
►None of given

Question No: 22 (Marks: 1) - Please choose one
In Protected Mode, the segment registers are used as _____

►Descriptor
►**Selector (Page 326)**
►All of the given choices
►None of the given choices

Question No: 23 (Marks: 1) - Please choose one
To access drive parameter block we use undocumented service _____

►09H/32H
►11H/32H
►17H/32H
►**21H/32H (Page 249)**

Question No: 24 (Marks: 1) - Please choose one
_____ is an absolute address relative to the start of physical drive.

►**LBA (Page 240)**
►LSN
►CHS
►None of the above

Question No: 25 (Marks: 1) - Please choose one
Boot block consists of _____ bytes.

►64
►128
►256
►**512 (Page 242)**

Question No: 26 (Marks: 1) - Please choose one
The DMA requests to acquire buses through the _____ signal.

►**HOLD (Page 186)**
►ACR
►ACK
►None of Given

عقل مند کہتا ہے میں کچھ نہیں جانتا جبکہ بے وقوف کہتا ہے کہ میں سب کچھ جانتا ہوں

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question No: 27 (Marks: 1) - Please choose one
The keyboard device writes a code 0xFA on the port 60H to indicate that the _____.

►Input buffer is full
►**Byte has been received properly (Page 179)**
►Output buffer is full
►None of the given

Question No: 28 (Marks: 1) - Please choose one
A single DMA can transfer _____ operands to and from memory in a single a bus cycle.

►**8-bits (Page 186)**
►16-bits
►32-bits
►12-bits

Question No: 29 (Marks: 1) - Please choose one
In FAT12, to calculate the address or offset from index, we need to multiply it with ___.

►1/2
►**3/2 (Page 267)**
►5/7
►7/2

Question No: 30 (Marks: 1) - Please choose one
_____ Register can be used to show that the channel is single transfer, block transfer or demand transfer mode.

►DMA Command register
►DMA Request Register
►DMA Mode Register
►**DMA controller Register (Page 187-188)**

خود کو تمھیں سے بڑھ کر کوئی اچھا مشورہ نہیں دے سکتا

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

CS609 – Solved Quizzes (Quiz No.3 & 4)

Question : 1 of 10 (Marks: 1) - Please choose one
When we mark a file as deleted by placing 0xE5 then the chain of clusters in FAT is also replaced by _____.

►E5
►1
►0 (Page 79)
►N

Question : 2 of 10 (Marks: 1) - Please choose one
Cluster size is reduced in _____.

►FAT12
►FAT16
►FAT32 Click here for detail
►None of the given

Question : 3 of 10 (Marks: 1) - Please choose one
In FAT32 _____ root directory entries are there.

►128
►256
►512
►None of the given Click here for detail

Question : 4 of 10 (Marks: 1) - Please choose one
If a file is having more than one cluster then it will be managed by _____.

►FAT
►BPB
►DPB
►None of the above

Question : 5 of 10 (Marks: 1) - Please choose one
Internal fragmentation is reduced in _____.

►FAT12
►FAT16 Click here for detail
►FAT32
►None of the given

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 6 of 10 (Marks: 1) - Please choose one
For supporting long file names, _____ fragments can be supported.

►12
►20
►26
►32

Question : 7 of 10 (Marks: 1) - Please choose one
To store a cluster in FAT 32 _____ is/are needed.

►Nibble
►Byte
►2 Bytes
►4 Bytes **Click here for detail**

Question : 8 of 10 (Marks: 1) - Please choose one
If a file size is 12K and the size of the cluster is 4K then _____ clusters are used for the file.

►2
►3
►4
►5

Question : 9 of 10 (Marks: 1) - Please choose one
We can access the contents of File by using the root directory entry and _____.

►Reserved Blocks
►Number of FAT copies
►File Allocation Table (FAT) (Page 269)
►None of the given

Question : 10 of 10 (Marks: 1) - Please choose one
FAT based file system can store file name in _____ form.

►ASCII
►UNICODE
►Both ASCII and UNICODE
►None of the given

جو شخص ناکامیوں سے ڈر کر بھاگتا ہے کامیابی اُس سے ڈر کر بھاگتی ہے

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 1 of 10 (Marks: 1) - Please choose one
Drive parameter block is derived from _____.

►FCB
►FAT
►**BPB (Page 249)**
►CPB

Question : 2 of 10 (Marks: 1) - Please choose one
We can access Blocks for FAT using _____.
►BPB
►DPB
►**FCB**
►Both BPB and DPB

Question : 3 of 10 (Marks: 1) - Please choose one
If we know the cluster number, we can access the blocks within the cluster using BIOS services directly.

►**True (Page 258)**
►False

Question : 4 of 10 (Marks: 1) - Please choose one
_____ is an internal data structure of DOS and resides in main memory.

►BPB
►DPB
►CPB
►None of the given. Click here for detail

Question : 5 of 10 (Marks: 1) - Please choose one
The size of DPB data structure is _____ bytes.
►16
►32
►64
►**128** click here for detail

Question : 6 of 10 (Marks: 1) - Please choose one
The size of FCB data structure is _____ bytes.
►**16** Click here for detail
►32
►64
►128

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 7of 10 (Marks: 1) - Please choose one
Advantages of FAT32 is/are _____.

►Large disk size can be managed in FAT32
►**Cluster size is reduced** Click here for detail
►Internal fragmentation is reduced
►All of the given

Question : 8 of 10 (Marks: 1) - Please choose one
_____ file system keeps the backup of its boot block.

►FAT12
►FAT16
►**FAT32 Click here for detail**
►None of the given

Question : 9 of 10 (Marks: 1) - Please choose one
To store a UNICODE character ____ is/are needed.

►Nibble
►Byte
►**2 Bytes Click here for detail**
►4 Bytes

Question : 10 of 10 (Marks: 1) - Please choose one
_____ is the first block on disk.

►LSN =0
►**LBA=0 (Page 240)**
►LBA=1
►Both LBA=0 and LSN=0

Question : 1 of 10 (Marks: 1) - Please choose one
If FAT entry is between FFF0H to FFF6H in FAT16 then _____.

►Cluster is available
►**It is a Reserved cluster (Page 272)**
►It is next file cluster
►It is a last file cluster

جو لوگوں کے سامنے فخر کرتا ہے وہ لوگوں کی نظروں سے گر جاتا ہے

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 2 of 10 (Marks: 1) - Please choose one
File system used in CD's is_____ file system

►**Contiguous** **Click here for detail**
►Chained
►Indexed
►None

Question : 3 of 10 (Marks: 1) - Please choose one
A file has 2 clusters and the size of cluster is 4K. What will be the size of file?

►2K
►**8K**
►16K
►32K

Question : 4 of 10 (Marks: 1) - Please choose one
In NTFS, Backup of boot block is stored at block # _____.

►2
►6
►**8**
►10

Question : 5 of 10 (Marks: 1) - Please choose one
The interval timer can operate in _____ modes.

►Five
►Seven
►Four
►**Six (Page 72)**

Question : 6 of 10 (Marks: 1) - Please choose one
File control block (FCB) is _____ byte long.

►**32** Click here for detail
►64
►16
►128

عقل مند اپنے عیب خود دیکھتا ہے اور بیوقوفوں کے عیب دنیا دیکھتی ہے

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 7 of 10 (Marks: 1) - Please choose one
On the execution of IRET instruction, number of bytes popped from stack is

► 4 bytes
► 6 bytes
► **8 bytes** **Click here for detail**
► 10 bytes

Question : 8 of 10 (Marks: 1) - Please choose one
If CPUID instruction is not present then the processor can be a

► **486 processor (Page 166)**
► 386 processor
► 286 processor
► All of the above

Question : 9 of 10 (Marks: 1) - Please choose one
Extended memory is available if the processor is of the type _____

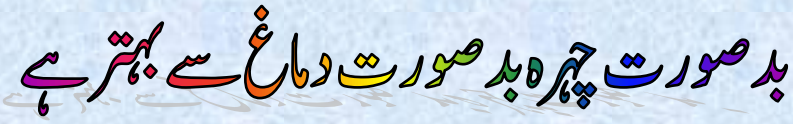
►**AT (Page 171)**
►XT
►All of the given choices
►None of them

Question : 10 of 10 (Marks: 1) - Please choose one
The built in mechanism within the UART for error detection is_____

►hamming code
►**parity (Page 107)**
►CRC16 (cyclic redundancy check 16 bit)
►CRC32 (cyclic redundancy check 32 bit)

Question : 1 of 10 (Marks: 1) - Please choose one
If three Programmable interrupt controllers are cascaded then how many interrupt driven hardware IO devices can be attached _____

►12
►18
►23
►**24 (Page 48)**



Question : 2 of 10 (Marks: 1) - Please choose one
Int 14H_____ can be used to send a byte

►Service#0
►**Service#1 (Page 121)**
►Service#2
►None of the given option.

Question : 3 of 10 (Marks: 1) - Please choose one
Int 14H _____ can be used to set the line parameter of the UART or COM port.

►**Service # 0 (Page 119)**
► Service # 1
►Service # 2
►None of the given options

Question : 4 of 10 (Marks: 1) - Please choose one
Int 14H _____ can be used to receive a byte.

►Service # 0
►Service # 1
►**Service # 2 (Page 121)**
►None of the given options

Question : 5 of 10 (Marks: 1) - Please choose one
The _____ function simply enables the self test facility within the modem control register

►STOn()
►SelfTest()
►**SelfTestOn() (Page 127)**
►None of these

Question : 6 of 10 (Marks: 1) - Please choose one
_____ is a device incorporated into the PC to update time even if the computer is off.

►Clock counter
►ROM
►Clock
►**Real time clock (Page 136)**

عقل مند آدمی اس وقت تک نہیں بولتا جب تک خاموشی نہیں ہو جاتی

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 7 of 10 (Marks: 1) - Please choose one
Interrupt _____ is used to get or set the time.

►0AH
►1AH (Page 136)
►2AH
►3AH

Question : 8 of 10 (Marks: 1) - Please choose one
_____ is used to set time.
►1A/02H
►1A/03H (Page 138)
►1A/04H
►1A/05H

Question : 9 of 10 (Marks: 1) - Please choose one
---------- is used to read date from RTC
►1A\02H
►1A\03H
►1A\04H (Page 138)
►1A\05H

Question : 10 of 10 (Marks: 1) - Please choose one
____whenever received indicates the start of communication and _____ whenever received indicates a temporary pause in the communication.

►XON & XOFF (Page 135)
►XOFF & XON
►XON & YOFF
►YON & XOFF

Question : 1 of 10 (Marks: 1) - Please choose one
The_____ function uses the COM port number to receive a byte from the COM port using BIOS services.
►recievebyte()
►initialize ()
►receive()
►recievechar() (Page 125)

Question : 2 of 10 (Marks: 1) - Please choose one
In self test mode the output of the UART is routed to its input.

►True (Page 117)
►False

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 3 of 10 (Marks: 1) - Please choose one
Only _____ ports are important from programming point of view.

► **70 and 71H (Page 141)**
►71 and 72H
►70 and 72H
►72 and 73H

Question : 4 of 10 (Marks: 1) - Please choose one
The BIOS interrupt 0x1AH can be used to configure real time clock

►**True (Page 136)**
►False

Question : 5 of 10 (Marks: 1) - Please choose one
DOS command _____ which gives the status of the memory and also points out which memory area occupied by which process.

►**mem/d (Page 13)**
►mem/e
►mem/m
►None of the given

Question : 6 of 10 (Marks: 1) - Please choose one
Each entry in the IVT is _____ in size.

►**4-bytes (Page 12)**
►6-bytes
►8-bytes
►2-bytes

Question : 7 of 10 (Marks: 1) - Please choose one
The tail value of the keyboard buffer should be examined to get to the _____ of the buffer.

►**Start (Page 56)**
►End
►Middle
►None of given)

Question : 8 of 10 (Marks: 1) - Please choose one
Usually interrupt procedures are reentrant procedures especially those interrupt procedure compiled using C language compiler are reentrant.

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

►**True (Page 38)**
►False

Question : 9 of 10 (Marks: 1) - Please choose one
_____ is Disk interrupt.

►10H
►11H
►**13H (Page 42)**
►14H

Question : 10 of 10 (Marks: 1) - Please choose one
In parallel communication, the maximum numbers of bits we can send between two computers are _____.

►2-bits
►4-bits
►6-bits
►**8-bits**

Question : 1 of 10 (Marks: 1) - Please choose one
14h include _____ which is used to send a byte.

►Service #0
►**Service #1 (Page 121)**
►Service #2
►Service #3

Question : 2 of 10 (Marks: 1) - Please choose one
The status register _____ is the main control register.

►**B (Page 146)**
►A
►C
►D

Question : 3 of 10 (Marks: 1) - Please choose one
_____ is used to identify the cause of interrupt.

►**Interrupt ID Register (Page 116)**
►PC Register
►AC Register
►None of All These

Question : 4 of 10 (Marks: 1) - Please choose one
In NTFS, up to _____ characters are used to store files names,

►30
►48
►255 (Page 283)
►510

Question : 5 of 10 (Marks: 1) - Please choose one
A cluster is a collection of contiguous _____.

►Blocks (Page 242)
►Sectors
►Bytes
►None of Given

Question : 6 of 10 (Marks: 1) - Please choose one
In BPB, root directory is saved in _____.

►Cluster#0
►Cluster#1
►Cluster#2
►Cluster#3

Question : 7 of 10 (Marks: 1) - Please choose one
In NTFS, total sizes of MFT entries are _____.

►16-bytes (Page 303)
►20-bytes
►26-bytes
►32-bytes

Question : 8 of 10 (Marks: 1) - Please choose one
In NTFS, _____ store the contents of file.

►Both small & large file Record
►Small record
►Large Record
►None of given

انسان دکھ نہیں دیتے بلکہ انسانوں سے وابستہ امیدیں دکھ دیتی ہیں

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 9 of 10 (Marks: 1) - Please choose one
In NTFS, contents and indexed of file is stored in _____.

►Small record
►Large Record
►Both small & large file Record
►None of given

Question : 10 of 10 (Marks: 1) - Please choose one
Total No. of bytes that can be stored in Keyboard Buffer is____.

►16
►32 (Page 54)
►64
►128

Question : 1 of 10 (Marks: 1) - Please choose one
BIOS support _____UARTS as COM ports.
►6
►4 (Page 113)
►3
►2

Question : 2 of 10 (Marks: 1) - Please choose one
DCE stands for _____.

►Data communication equipment (Page 109)
►Distributed Computing Environment
►Data Communications Equipment
►Data Carrier Equipment

Question : 3 of 10 (Marks: 1) - Please choose one
In counter register bit no. 3 changes its value between 0 and 1 with in ____clock cycles

►1
►2
►4
► 16 (Page 69)

بہترین تجربہ وہ ہے جس سے نصیحت حاصل ہو

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 4 of 10 (Marks: 1) - Please choose one
In _____each byte is needed to be encapsulated in start and end.

►Synchronous communication
►**Asynchronous communication (Page 106)**
►Both
►None of given

Question : 5 of 10 (Marks: 1) - Please choose one
The _____ service # is not used in any interrupt.

►01
►02
►03
►**FF**

Question : 6 of 10 (Marks: 1) - Please choose one
If we want to send printing on the printer then we have to perform following steps.

►Initialize printer
►Read Status
►Check Error
►**All of the given**

Question : 7 of 10 (Marks: 1) - Please choose one
DTE is _____.

►**Data terminal equipment (Page 109)**
►Data transmitting equipment
►Dual terminal equipment
►None of the given.

Question : 8 of 10 (Marks: 1) - Please choose one
If printer is _____ then printer sends back the ACK signal to the printer interface

►**idle (Page 97)**
►busy
►Out of paper
►None of the given

خوبصورتی علم وادب سے ہوتی ہے لباس و حسن سے نہیں

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 9 of 10 (Marks: 1) - Please choose one
DSR stands for _____ .

►**Data set ready (Page 111)**
►Data service ready
►Data stock ready
►None of the given

Question : 10 of 10 (Marks: 1) - Please choose one
At IRQ 7 Interrupt # ___ is used.

► 0x0A
►0x0B
► 0x0C
►**0x0F (Page 95)**

Question : 1 of 10 (Marks: 1) - Please choose one
The memory addresses of COM ports remain same for all computers
►True
►**False**

Question : 2 of 10 (Marks: 1) - Please choose one
In keyboard status byte bit no. 2 and 3 are used for ctrl and alt keys respectively. which of the following condition is used to check that Ctrl + Alt keys are pressed. Where: unsigned char far * scr = (unsigned char far *)(0x00400017);

►**if (((*scr)&12)==12)**
►if (((*scr)&8)==8)
►if (((*scr)&4)==4)
►if (((*scr)&2)==2)

Question : 3 of 10 (Marks: 1) - Please choose one
In case of synchronous communication a timing signal is required to identify the start and end of a bit.

►**True (Page 105)**
►False

Question : 4 of 10 (Marks: 1) - Please choose one
The baud rate is set in accordance with the divisor value loaded within the UART internal registers base +0 and base +1.

►**TRUE (Page 114)**
►FALSE

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 5 of 10 (Marks: 1) - Please choose one
Software based flow control make use of -------- control characters

►Xon
►XOFF
►**Both (Page 135)**
►None

Question : 6 of 10 (Marks: 1) - Please choose one
---------- is used to read time from RTC

►**1A\02H (Page 137)**
►1A\03H
►1A\04H
►1A\05H

Question : 7 of 10 (Marks: 1) - Please choose one
Int_____ service 0 can be used to set the line parameter of the UART or COM port.

►**14H (Page 119)**
►15H
►13H
►None of the given option

Question : 8 of 10 (Marks: 1) - Please choose one
When LBA is equal to zero (0), it means _____.

►**First block of the disk (Page 240)**
►First block of the logical drive
►First block of the hidden block
►None of the given

Question : 9 of 10 (Marks: 1) - Please choose one
In IRQ2 and IRQ3 which one has the highest priority?

►Can't be determined
►Both have same priority
►IRQ3
►**IRQ2 (Page 47)**

تم اچھا کرو زمانہ تم کو برا سمجھے یہ اس سے بہتر ہے کہ تم برا کرو اور زمانہ تم کو اچھا سمجھے

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 10 of 10 (Marks: 1) - Please choose one
Following is not a method of I/O

►Programmed I/O
► Interrupt driven I/O
►Hardware Based I/O (Page 4)
►None of given

Question : 1 of 10 (Marks: 1) - Please choose one
It is possible to perform I/O operations from three different methods.

►True (Page 7)
►False

Question : 2 of 10 (Marks: 1) - Please choose one
The Function of I/O controller is to provide _____.

►I/O control signals
►Buffering
►Error Correction and Detection
►All of given (Page 5)

Question : 3 of 10 (Marks: 1) - Please choose one
Which of the following are types of ISR _____.

►BIOS (Basic I/O service) ISR
►DOS ISR
►ISR provided by third party device drivers
►All of the given (Page 13)

Question : 4 of 10 (Marks: 1) - Please choose one
Interrupt service number is usually placed in _____ register.

►CH
►CL
►AH (Page 26)
►AL

انسان کے لئے بری صحبت سے بڑھ کر بری کوئی چیز نہیں

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 5 of 10 (Marks: 1) - Please choose one
NMI Stand for

►**Non Maskable Interrupt (Page 46)**
►Non Multitude Interrupt
►Non Maskable Instruction
►None of Given

Question : 6 of 10 (Marks: 1) - Please choose one
A single interrupt controller can arbitrate among ____ different devices.
►4
► 6
►**8 (Page 47)**
►10

Question : 7 of 10 (Marks: 1) - Please choose one
The microprocessor package has many signals for data. Below are some incorrect priority order (Higher to Lower).

►**Reset,Hold,NMI,INTR (Page 46)**
►NMI, INTR,Hold,Reset
►INTR,NMI,Reset,Hold
►None of the Given

Question : 8 of 10 (Marks: 1) - Please choose one
The following command "outportb (0x61,inportb(0x61) & 0xFC);" will

►Turn on the speaker
►**Turn off the speaker (Page 75)**
►Toggle the speaker
►None of the given

Question : 9 of 10 (Marks: 1) - Please choose one
The following command "outportb (0x61,inportb(0x61) | 3);" will _____ .
►**Turn on the speaker (Page 74)**
►Turn off the speaker
►Toggle the speaker
►None of the above

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 10 of 10 (Marks: 1) - Please choose one
The PPI acts as an interface between the CPU and a parallel _____ .

►**I/O device (Page 83)**
►CPU
►BUS
►None of Given

Question : 1 of 10 (Marks: 1) - Please choose one
BIOS DO NOT support _____.

►LPT1
►LPT2
►LPT3
►**LPT4 (Page 91)**

Question : 2 of 10 (Marks: 1) - Please choose one
_____ bit is cleared to indicate the low nibble is being sent.

►D1
►D2
►D3
►**D4 (Page 104)**

Question : 3 of 10 (Marks: 1) - Please choose one
The bit _____ of Line control register in UART, if cleared will indicate that DLL is the data register.

►1
►3
►5
►**7 (Page 114)**

Question : 4 of 10 (Marks: 1) - Please choose one
_____ file system is used in NTFS based systems.

Contiguous Chained
Indexed
None of the given

خاموشی غصے کا بہترین علاج ہے

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 5 of 10 (Marks: 1) - Please choose one
Communication between keyboard and keyboard controller is _____.

►Asynchronous serial
►**Synchronous serial (P 77)**
►Parallel communication
►None of the given

Question : 6 of 10 (Marks: 1) - Please choose one
In NTFS, boot sector is stored at

►First and 6th sector
►First and Last sector
►Only at Last sector
►**Only at First sector**

Question : 7 of 10 (Marks: 1) - Please choose one
Standard PC operates in two modes in terms of memory which are

►Real mode and Extended Mode
►Base mode and Memory Mode
►None of the given
►**Real mode and protected mode (Page 6)**

Question : 8 of 10 (Marks: 1) - Please choose one
IVT is a table containing _____ byte entries each of which is a far address of an interrupt service routine.

►2
►**4 (Page 20)**
►8
►16

Question : 9 of 10 (Marks: 1) - Please choose one
Each paragraph in keep function is ____ bytes in size.

►4
►8
►**16 (Page 24)**
►32

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 10 of 10 (Marks: 1) - Please choose one
A software interrupt does not require EOI (End of interrupt).

►**True (Page 49)**
►False

Question : 1 of 10 (Marks: 1) - Please choose one
To store each character in keyboard buffer ____ bytes are required.
►**2 (Page 54)**
►4
►6
►8

Question : 2 of 10 (Marks: 1) - Please choose one
Interrupt _____ is empty; we can use its vector as a flag.

►9H
►13H
►15H
►**65H (Page 65)**

Question : 3 of 10 (Marks: 1) - Please choose one
Command register is an _____ bit register
►4
►**8 (Page 71)**
►16
►32

Question : 4 of 10 (Marks: 1) - Please choose one
How many bytes can be used to store a file name in NTFS?
►128
►**255**
►510
►1024

Question : 5 of 10 (Marks: 1) - Please choose one
____ is the first logical sector of NTFS partition.

►DPB
►MFT
►**Boot sector**
►None

Question : 6 of 10 (Marks: 1) - Please choose one
In boot block BIOS parameter block starts from 03H

►05H
►08H
►0BH (Page 302)

Question : 7 of 10 (Marks: 1) - Please choose one
IN NTFS, FAT and root directory is replaced by

►FCB
►MFT (Page 301)
►Hidden blocks
►Boot sector

Question : 8 of 10 (Marks: 1) - Please choose one
Block # 2 is the safest block to store the backup of boot block.

►True
►False

Question : 2 of 10 (Marks: 1) - Please choose one
The keyboards interface as discussed earlier uses the IRQ0 and the port 64H as data port.

►True
► False

Question : 3 of 10 (Marks: 1) - Please choose one
FAT12 will have 12-bit wide entries and can have 2^12=4096 entries maximum

►True
► False

Question : 4 of 10 (Marks: 1) - Please choose one
In order to produce the sound from PC internal Speaker we have to load the___bit divisor value at the ___port.

►8, 0x21
►16, 0x42
►32, 0x22
►64, 0x32

Some More MCQs and Quizzes

Question : 1 of 10 (Marks: 1) - Please choose one
DMA stands for_____

►**Direct Memory Access (Page 4)**
►Distinct Memory Access
►Direct Module Access
►Direct Memory Allocation

Question : 2 of 10 (Marks: 1) - Please choose one
UART stands for_____

►**Universal Asynchronous Receiver Transmitter (Page 107)**
►Universal Adjustment and Realigning Tool
►Unconventional Assisted Recovery Team
►None of these

Question : 3 of 10 (Marks: 1) - Please choose one
Interrupt Vector Table (IVT) in short is a _____ bytes sized table.

►**1024 (Page 10)**
►2048
►3072
►4096

Question : 4 of 10 (Marks: 1) - Please choose one
Hardware Interrupts are _____.

►Preemptive
►**Non-Preemptive (Page 48)**
►Both Preemptive and Non-Preemptive
►None of Given

Question : 5 of 10 (Marks: 1) - Please choose one
Timer interrupt is a _____.

►**Hardware Interrupt (Page 28)**
►Software Interrupt
►Both of these
►None of These

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 6 of 10 (Marks: 1) - Please choose one
The keyboard makes use of interrupt number _____ for its input operations.
►**9 (Page 34)**
►10
►11
►12

Question : 7 of 10 (Marks: 1) - Please choose one
Register can be used to divide frequency is _____

►**Counter Register (Page 69)**
►Accumulator Register
►None of these

Question : 8 of 10 (Marks: 1) - Please choose one
Which port is known as Data Port_____
►**60H (Page 177)**
►61H
►64H
►69H

Question : 9 of 10 (Marks: 1) - Please choose one
LPTs can be swapped.
►**True (Page 92)**
►False

Question : 10 of 10 (Marks: 1) - Please choose one
PPI is used to perform parallel communication
►**True (Page 81)**
►False

Question : 1 of 10 (Marks: 1) - Please choose one
_____is used to control the printer via the BIOS

►Int 16H
►**Int 17H (Page 84)**
►Int 18H
►Int 19H

اطمینان قلب چاہتے ہو تو حسد سے دور رہو

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 2 of 10 (Marks: 1) - Please choose one
There are two main types of interrupts namely _____.

►PC based and Window based
►Hardware based and Kernal based
►**Hardware interrupts and Software interrupts (Page 10)**
►None of the given

Question : 3 of 10 (Marks: 1) - Please choose one
To set the interrupt vector means is to change the double word sized interrupt vector within the IVT.

►**True (Page 22)**
►False

Question : 4 of 10 (Marks: 1) - Please choose one
The service number is usually placed in the _____ register.
►AL
►CL
►**AH (Page 26)**
►AX

Question : 5 of 10 (Marks: 1) - Please choose one
The keyboard makes use of interrupt number _____ for its input operations.
►**9 (Page 34)**
►10
►11
►12

Question : 6 of 10 (Marks: 1) - Please choose one
The service _____ is called the keyboard hook service.

►15H/2FH
►**15H/4FH (Page 44)**
►15H/FFH

Question : 7 of 10 (Marks: 1) - Please choose one
The BIOS interrupt _____ can be used to configure RTC.

►**1AH (Page 136)**
►2AH
►3AH
►4AH

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 8 of 10 (Marks: 1) - Please choose one
PPI stands for
►Parallel Programmable interface
►Peripheral Programmable interface (Page 76)
►Port Programmable interface
►None of the given

Question : 9 of 10 (Marks: 1) - Please choose one
Int _____ is used to control the printer via the BIOS.
►17H (Page 84)
►18H
►20H
►21H

Question : 10 of 10 (Marks: 1) - Please choose one
Counter register can be used to divide clock signal.
►True (Page 69)
►False

Question : 1 of 10 (Marks: 1) - Please choose one
The bit number _____ of the coprocessor control word is the interrupt enable flag.

►7 (Page 168)
►8
►9
►6

Question : 2 of 10 (Marks: 1) - Please choose one
There are _____ kinds of serial communication.

►2 (Page 105)
►3
►4
►5

Question : 3 of 10 (Marks: 1) - Please choose one
_____ store the base address for LPT1.

►40:00H
►40:02H
►40:08H (Page 92)
►40:1AH

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 4 of 10 (Marks: 1) - Please choose one
The amount of memory above conventional memory (extended memory) can be determined using the service _____.

►**15H/88H (Page 162)**
►16H/88H
►17H/88H
►21H/88H

Question : 5 of 10 (Marks: 1) - Please choose one
The output on the monitor is controlled by a controller called _____within the PC.

►**Video controller (Page 30)**
►Bus controller
►Ram controller
►None of the given

Question : 6 of 10 (Marks: 1) - Please choose one
Interrupt 9 usually reads the _____ from keyboard.

►ASCII code
►**Scan code (Page 34)**
►Both ASCII and Scan code
►None of the above

Question : 7 of 10 (Marks: 1) - Please choose one
NMI Stand for

►**Non Maskable Interrupt (Page 46)**
►Non Multitude Interrupt
►Non Maskable Instruction
►None of Given

Question : 8 of 10 (Marks: 1) - Please choose one
A single interrupt controller can arbitrate among ____ different devices.

►4
►6
►**8 (Page 47)**
►10

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 9 of 10 (Marks: 1) - Please choose one
The microprocessor package has many signals for data. Below are some in Correct priority order (Higher to Lower).

►**Reset,Hold,NMI,INTR (Page 46)**
►NMI, INTR,Hold,Reset
►INTR,NMI,Reset,Hold
►None of the Given

Question : 10 of 10 (Marks: 1) - Please choose one
The _____function initialize the COM port whose number is passed as parameter using BIOS services.

►Initializecom()
►**Initialize() (Page 125)**
►Recievechar()
►None of these option

Question : 1 of 10 (Marks: 1) - Please choose one
There are two type of communication synchronous and Anti Synchronous
►True
►**False (Page 105)**

Question : 2 of 10 (Marks: 1) - Please choose one
REGS is a Union

►**True**
►False

Question : 3 of 10 (Marks: 1) - Please choose one
Keyboard Status Byte is located at the address

► 0040:0000H
► 0040:0013H
► 0040:0015H
► **0040:0017H (Page 29)**

Question : 4 of 10 (Marks: 1) - Please choose one
If we use keep (0, 1000) in a TSR program, the memory allocated to it is

► 64000 bytes
► 32000 bytes
► **16000 bytes**
► 80000 bytes

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 5 of 10 (Marks: 1) - Please choose one
Maximum number of interrupts in a standard PC is

► 64
► 128
► **256 (Page 10)**
► 512

Question : 6 of 10 (Marks: 1) - Please choose one
The -------- function receive a byte and COM port number is passed as parameter using BIOS service

►Receivebyte ();
►Receive ();
►**Receivechar (); (Page 125)**
►None of the given option

Question : 7 of 10 (Marks: 1) - Please choose one
---------- whenever receive indicates the start of communication …….. whenever receive indicates the end of communication

►**XON\XOFF (Page 135)**
►XOFF\XON
►XON\YOFF
►YON\XOFF

Question : 8 of 10 (Marks: 1) - Please choose one
---------- is used to set time from RTC

►1A\02H
►**1A\03H (Page 138)**
►1A\04H
►1A\05H

Question : 9 of 10 (Marks: 1) - Please choose one
Set the Interrupt vector means to change the double word sized interrupt vector within IVT.
►**True (Page 22)**
►False

Question : 10 of 10 (Marks: 1) - Please choose one
If keyboard buffer is empty the head and tail points at the same location.
►**True (Page 55)**
►False

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Question : 1 of 10 (Marks: 1) - Please choose one
Standard PC can have _____ PPI.
►1
►4 **(Page 84)**
►8
►16

Question : 2 of 10 (Marks: 1) - Please choose one
By cascading two DMAs ____ bits can be transferred.
►4
►8
►16 **(Page 186)**
►32

Question : 3 of 10 (Marks: 1) - Please choose one
PPI interconnection _____ bits is cleared to indicate low nibble is being sent.
►D1
►D2
► D3
► D4 **(Page 101)**

Question : 4 of 10 (Marks: 1) - Please choose one
Display device (Monitor) performs _____ I/O.

►memory mapped **(Page 30)**
►Isolated
►Both of above
►None of these

Question : 5 of 10 (Marks: 1) - Please choose one
Timer interrupt occurs _____ times every second by means of hardware.

►18.2 **(Page 28)**
►16.2
►15.2
►14.2

Question : 6 of 10 (Marks: 1) - Please choose one
An I/O device cannot be directly connected to the busses so controller is placed between CPU and I/O.
►True **(Page 83)**
►False

Question : 7 of 10 (Marks: 1) - Please choose one
Tail of keyboard should get to get the start of buffer.
►**True (Page 55)**
►False

Question : 8 of 10 (Marks: 1) - Please choose one
____ No. of bytes are used to store the character in the keyboard buffer.
► 1
►**2 (Page 54)**
►4
► 8

Question : 9 of 10 (Marks: 1) - Please choose one
We have set the bit No. 7 of IMR(Interrupt Mask Register) to unmask the Interrupt so that interrupt _____ can occur at ____ line.

►**0xf ,IRQ 7**
► 0xa, IRQ 6
► 0x8, IRQ 5
►0x6, IRQ 2

Question : 10 of 10 (Marks: 1) - Please choose one
If we want to produce the grave voice from speaker phone then we have to load the ____ divisor values at Port ____.

►high, 0x42
►low, 0x22
►high, 0x22
►**low, 0x42**

اپنی مرضی اور اللہ کی مرضی میں فرق کا نام غم ہے
وہ لوگ مبارک ہیں جو الفاظ سے نصیحت نہیں کرتے بلکہ عمل سے کرتے ہیں
ہر چیز کی ایک پہچان ہوتی ہے اور عقلمند کی پہچان غور و فکر کرنا ہے اور غور و فکر کی پہچان خاموشی ہے

Muhammad Moaaz Siddiq – MCS(4th)
Moaaz.pk@gmail.com
Campus:- Institute of E-Learning & Moderen Studies (IEMS) Samundari

Topic 70: Binary Search Using Heaps

The example is formulated using two heaps. One is a node heap and the other is a data heap. Node heap is used to build a tree, while data heap is used to store keys. Recursive functions are used for allocating nodes and scanning nodes as tree is a recursive structure. Data in file is read in a record and the key is used to lexically build a tree. The tree only contains the key entries. The file is ultimately sorted by an In-order traversal of the tree.

```
/* Chapter 5, sortBT command.   Binary Tree version. */
/* sort files
                Sort one or more files.
                This limited implementation sorts on the first field only.
                The key field length is assumed to be fixed length (8-characters).
                The data fields are varying length character strings. */
/* This program illustrates:
                1.      Multiple independent heaps; one for the sort tree nodes,
                   the other for the records.
                2.      Using HeapDestroy to free an entire data structure in a single operation.
                3.      Structured exception handling to catch memory allocation errors. */
/* Technique:
                1.      Scan the input file, placing each key (which is fixed size)
                   into the binary search tree and each record onto the data heap.
                2.      Traverse the search tree and output the records in order.
                3.      Destroy the heap and repeat for the next file. */

#include "Everything.h"
#define KEY_SIZE 8
                /* Structure definition for a tree node. */
typedef struct _TREENODE {
                struct _TREENODE *Left, *Right;
                TCHAR key[KEY_SIZE];
                LPTSTR pData;
} TREENODE, *LPTNODE, **LPPTNODE;
#define NODE_SIZE sizeof (TREENODE)
#define NODE_HEAP_ISIZE 0x8000
#define DATA_HEAP_ISIZE 0x8000
#define MAX_DATA_LEN 0x1000
#define TKEY_SIZE KEY_SIZE * sizeof (TCHAR)
#define STATUS_FILE_ERROR 0xE0000001        // Customer exception
LPTNODE FillTree (HANDLE, HANDLE, HANDLE);
BOOL Scan (LPTNODE);
int KeyCompare (LPCTSTR, LPCTSTR), iFile; /* for access in exception handler */
BOOL InsertTree (LPPTNODE, LPTNODE);
int _tmain (int argc, LPTSTR argv[])
{
```

```
            HANDLE hIn = INVALID_HANDLE_VALUE, hNode = NULL, hData =
NULL;
            LPTNODE pRoot;
            BOOL noPrint;
            CHAR errorMessage[256];
            int iFirstFile = Options (argc, argv, _T ("n"), &noPrint, NULL);
            if (argc <= iFirstFile)
                  ReportError (_T ("Usage: sortBT [options] files"), 1, FALSE);
                                    /* Process all files on the command line. */
for (iFile = iFirstFile; iFile < argc; iFile++) __try {
                                                      /* Open the input file. */
                  hIn = CreateFile (argv[iFile], GENERIC_READ, 0, NULL,
OPEN_EXISTING, 0, NULL);
                  if (hIn == INVALID_HANDLE_VALUE)
                        RaiseException (STATUS_FILE_ERROR, 0, 0, NULL);

                  __try {
                        /* Allocate the two growable heaps. */
                        hNode = HeapCreate (
HEAP_GENERATE_EXCEPTIONS | HEAP_NO_SERIALIZE, NODE_HEAP_ISIZE, 0);
                        hData = HeapCreate (
HEAP_GENERATE_EXCEPTIONS | HEAP_NO_SERIALIZE, DATA_HEAP_ISIZE, 0);
/* Process the input file, creating the tree. */
                        pRoot = FillTree (hIn, hNode, hData);
                        /* Display the tree in key order. */
                        if (!noPrint) {
                                    _tprintf (_T ("Sorted file: %s\n"), argv[iFile]);
                                    Scan (pRoot);
                        }
                  } __finally { /* Heaps and file handle are always closed */
                        /* Destroy the two heaps and data structures. */
                        if (hNode != NULL) HeapDestroy (hNode);
                        if (hData != NULL) HeapDestroy (hData);
                        hNode = NULL; hData = NULL;
                        if (hIn != INVALID_HANDLE_VALUE) CloseHandle (hIn);
                        hIn = INVALID_HANDLE_VALUE;
                  }
            } /* End of main file processing loop and try block. */
            /* Handle the exceptions that we can expect - Namely, file open error or out of
memory. */
            __except ( (GetExceptionCode() == STATUS_FILE_ERROR ||
GetExceptionCode() == STATUS_NO_MEMORY)
                                    ? EXCEPTION_EXECUTE_HANDLER :
EXCEPTION_CONTINUE_SEARCH)
            {
```

```
                    _stprintf (errorMessage, _T("\n%s %s"), _T("sortBT error on file:"),
argv[iFile]);
                    ReportError (errorMessage, 0, TRUE);
                }
                return 0;
}
LPTNODE FillTree (HANDLE hIn, HANDLE hNode, HANDLE hData)
/* Scan the input file, creating a binary search tree in the
                hNode heap with data pointers to the hData heap. */
/* Use the calling program's exception handler. */
{
                LPTNODE pRoot = NULL, pNode;
                DWORD nRead, i;
                BOOL atCR;
                TCHAR dataHold[MAX_DATA_LEN];
                LPTSTR pString;
                                                /* Open the input file. */
while (TRUE) {
                    pNode = HeapAlloc (hNode, HEAP_ZERO_MEMORY, NODE_SIZE);
                    pNode->pData = NULL;
                    (pNode->Left) = pNode->Right = NULL;
                                                /* Read the key. Return if done. */
                    if (!ReadFile (hIn, pNode->key, TKEY_SIZE,
                                        &nRead, NULL) || nRead != TKEY_SIZE)
                                                /* Assume end of file on error. All records
must be just the right size */
                        return pRoot;   /* Read the data until the end of line. */
                    atCR = FALSE;               /* Last character was not a CR. */
                    for (i = 0; i < MAX_DATA_LEN; i++) {
                        ReadFile (hIn, &dataHold[i], TSIZE, &nRead, NULL);
                        if (atCR && dataHold[i] == LF) break;
                        atCR = (dataHold[i] == CR);
}
                    dataHold[i - 1] = _T('\0');
                    /* dataHold contains the data without the key.
                        Combine the key and the Data. */
                    pString = HeapAlloc (hData, HEAP_ZERO_MEMORY,
                                        (SIZE_T)(KEY_SIZE + _tcslen (dataHold) + 1) *
TSIZE);
                    memcpy (pString, pNode->key, TKEY_SIZE);
                    pString[KEY_SIZE] = _T('\0');
                    _tcscat (pString, dataHold);
                    pNode->pData = pString;
                                                /* Insert the new node into the search tree. */
                    InsertTree (&pRoot, pNode);
} /* End of while (TRUE) loop */
```

```
                return NULL; /* Failure */
}
BOOL InsertTree (LPPTNODE ppRoot, LPTNODE pNode)
/* Insert the new node, pNode, into the binary search tree, pRoot. */
{
                if (*ppRoot == NULL) {
                        *ppRoot = pNode;
                        return TRUE;
                }
                if (KeyCompare (pNode->key, (*ppRoot)->key) < 0)
                        InsertTree (&((*ppRoot)->Left), pNode);
                Else
InsertTree (&((*ppRoot)->Right), pNode);
                return TRUE;
}
int KeyCompare (LPCTSTR pKey1, LPCTSTR pKey2)
/* Compare two records of generic characters.
                The key position and length are global variables. */
{
                return _tcsncmp (pKey1, pKey2, KEY_SIZE);
}
static BOOL Scan (LPTNODE pNode)
/* Scan and print the contents of a binary tree. */
{
                if (pNode == NULL)
                        return TRUE;
                Scan (pNode->Left);
                _tprintf (_T ("%s\n"), pNode->pData);
                Scan (pNode->Right);
                return TRUE;
}
```

Topic 71: Memory-Mapped Files

Memory Mapping

Dynamic memory is allocated from the paging file. The paging file is controlled by the Operating System's (OS) virtual memory management system. Also the OS controls the mapping of virtual address onto physical memory. Memory mapped files help to directly map virtual memory space onto a normal file.

Advantages of Memory Mapped IO

There is no need to invoke direct file Input Output (IO) operations. Any data structure placed in file will be available for later use as well. It is convenient and efficient to use in-memory algorithms for sorting, searching etc. Large files could be processed as if they are placed in

memory. File processing is faster than ReadFile() and WriteFile(). There is no need to manage buffers for repetitive operation on a file. This is more optimally done by OS. Multiple processes can share memory space by mapping their virtual memory space onto a file. For file operations, page file space is not needed.

Other considerations

Windows also use memory mapping while implementing Dynamic Link Libraries (DLLs) and loading & executing executable (EXE) files. It is strongly recommended to use SHE exception handling while dealing with memory mapped file to look for EXCEPTION_IN_PAGE_ERROR exceptions.

Topic 72: File Mapping Objects

File Mapping Objects

In order to perform memory mapped file IO operations, file mapping objects need to be created. This object uses the file handle of an open file. The open file or part of the file is mapped onto the address space of the process. File mapping objects are assigned names so that they are also available to other processes. Moreover, these mapping objects also require protection and security attributes and a size. The API used for this purpose is CreateFileMapping().

HANDLE CreateFileMapping(HANDLE hFile,

LPSECURITY_ATTRIBUTES lpFileMappingAttributes,

DWORD flProtect,

DWORD dwMaximumSizeHigh,

DWORD dwMaximumSizeLow, LPCTSTR lpName);

hFile is the open handle to file compatible with protection flag dwProtect

INVALID_HANDLE_VALUE refers to the paging file

LPSECURITY_ATTRIBUTES allow the mapping object to be secured

dwProtect specifies the mapping file access with following attributes

PAGE_READONLY - It means that page can only be read within the mapped region. It can neither be written nor executed. hFile must have GENERIC_READ access.

PAGE_READWRITE - Provides full access to object if the hFile has GENERIC_READ and GENERIC_WRITE access.

PAGE_WRITECOPY - Means that when a mapped region changes, a private copy is written to the paging file and not to the original file.

dwMaximumSizeHigh and dwMaximumSizeLow specify the size of the mapping object. If set to 0, then the current file size is used. Carefully specify this size in the following cases:

· If the file size is expected to grow, then use the expected file size.

· Do not map a region beyond this limit. Once the size is assigned the mapping region cannot grow.

· An attempt to create a mapping on a zero length file will fail.

· The mapping size needs to be specified in the form of two 32-bit values rather than one 64-bit value.

· lpMapName is the name of the map that can also be used by other processes. Set this to NULL if you do not mean to share the map.

Topic 73: Open Existing File Mapping Objects

Open Existing File Mapping Objects

Previously, we discussed that a file mapping can be assigned a shared name by using CreateFileMapping(). This shared name can be used to open existing file maps using OpenFileMapping(). A file map created by a certain process can be subsequently used by other processes by referring to the object by name. The operation may fail if the name does not exist.

HANDLE OpenFileMapping(DWORD dwDesiredAccess,

BOOL bInheritHandle,

LPCSTR lpMapName);

dwDesiredAccess is checked against the accessed assigned by CreateFileMapping();

lpMapName is the name of the mapping assigned by CreateFileMapping().

Topic 74: Mapping Objects to Process Address Space

Mapping into Process Space

Previously, file mapping was created or a handle to already created mapping handle was obtained. The next step is to assign a process address space to file mapping. In case of heaps, first heap was created and then HeapAlloc() was used to allocate space within heap. Similarly once the file mapping is created, MapViewOfFile() is used to define file view block.

LPVOID MapViewOfFile(HANDLE hFileMappingObject,

DWORD dwDesiredAccess,

DWORD dwFileOffsetHigh,

DWORD dwFileOffsetLow,

SIZE_T dwNumberOfBytesToMap);

hFileMappingObject is file mapping object previously obtained by CreateFileMapping().

dwDesiredAccess should be compatible with access rights of file mapping object. The three flag commonly used are:

FILE_MAP_WRITE

FILE_MAP_READ

FILE_MAP_ALL_ACCESS

dwFileOffsetHigh and dwFileOffsetLow give the starting address of the file from where the mapping starts. To start the mapping from the start of a file, set both as zero. This value should be specified in multiples of 64K.

dwNumberOfBytesToMap shows the number of bytes of file to map. Set as zero to map the whole file.

If the function is successful it returns the base address of the mapped region. If the function fails, the return value is NULL.

Closing Mapping Handle

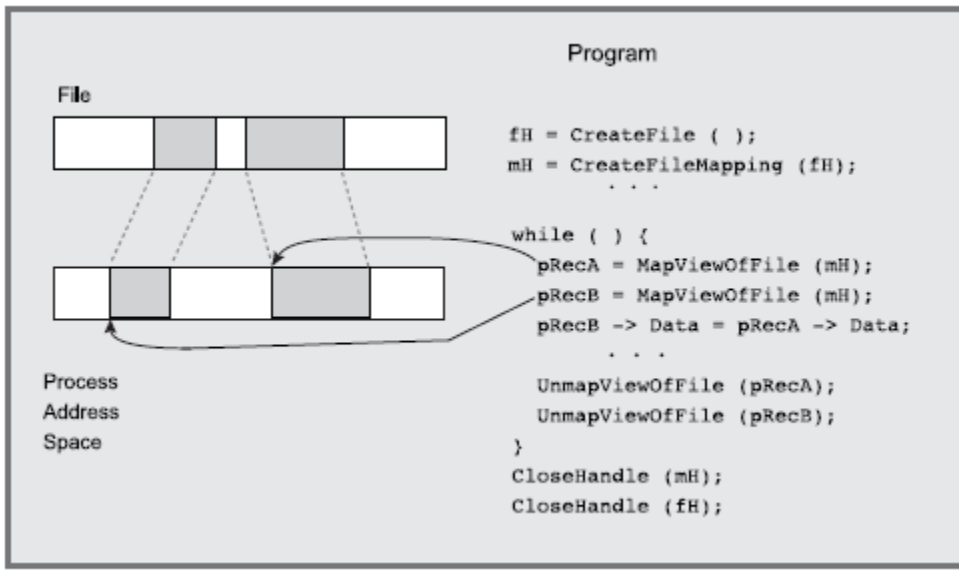
As it is necessary to release Heap blocks with HeapFree(), it is also necessary to unmap file views. File views are unmapped using UnmapViewOfFile().

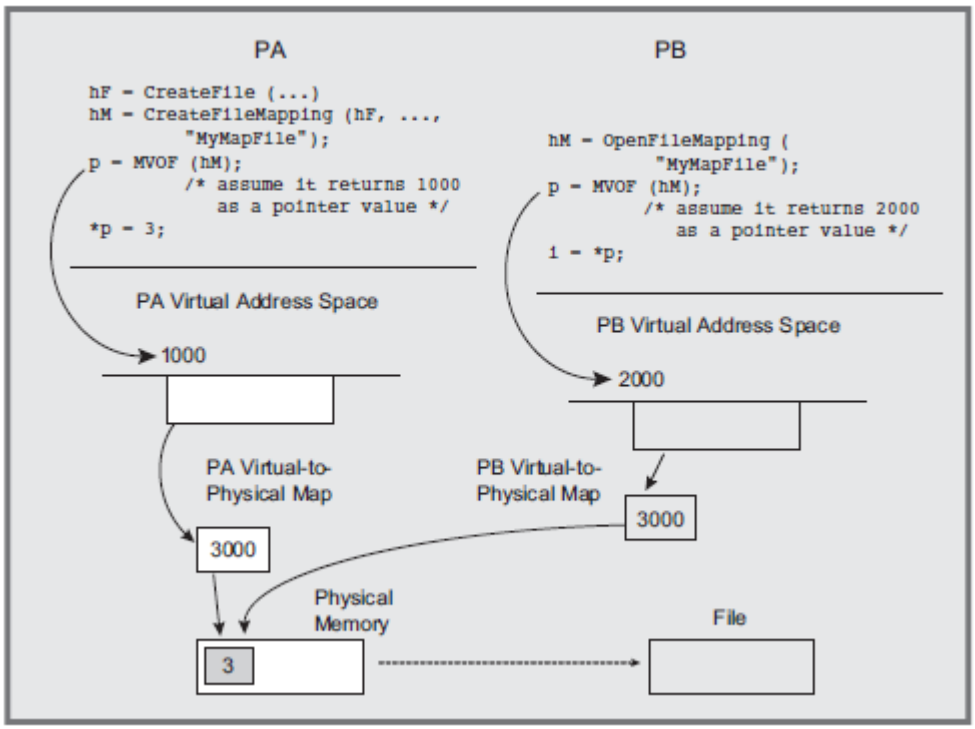
BOOL UnmapViewOfFile(LPCVOID lpBaseAddress);

lpBaseAddress is the pointer to the base address of the mapped view. If the function fails, the return value is zero.

Flushing File View

The file view can be flushed using the FlushViewOfFile() API. This will force the OS to writeback the dirty pages of the file on to disk. In case two processes try to access a file at a time such that one uses file mapping and other uses ReadFile() and WriteFile(). Then both processes may not receive the same view. Changes made through file maps might still be in memory and may not be accessible through ReadFile or WriteFile unless they are flushed. To get a uniform view, it is necessary that all the processes use file maps.





Topic 75: More About File Making

File Mapping Limitations

In Win32, it is not possible to map files bigger than 2-3 GB. Also the entire 3GB might not be available for merely file space. The above limitation is alleviated in Win64. File mapping cannot be extended. You need to know the size of a map in advance. Customized functions would be required to allocate memory within the mapped region.

File Mapping Procedure

The following minimum steps need to be taken while working with mapped files:

· Open File with at least GENERIC_READ access.

· If the file is new then set its length as some non-zero value using SetFilePointerEx() followed by SetEndOfFile().

· Map the file using CreateFileMapping() or OpenFileMapping ().

· Create one or more map views using MapViewOfFile().

· Access file through memory references.

· Unmap and then again Map file view to change regions.

· Use SEH to catch EXCEPTIO_IN_PAGE_ERROR exceptions.

· In the end, unmap file view with UnmapViewOfFile() and use CloseHandle() to close map and file handles.

Topic 76: Sequential File Access Using file mapping

Sequential File Access

Accessing a file through file mapping presents visible advantages. Although the setting up of fileviews might be programmatically complex, the advantages are far bigger. The processing time may reduce 3 folds as compared to conventional file operations while dealing with sequential files. These advantages may only seem to disappear if the size of input and output files is too large. The example is a simple Ceasar cipher application. It sequentially processes all the characters with the file. It simply substitutes each character by shifting it a few places in the ASCII set.

```
/* Chapter 5.
cci_fMM.c function: Memory Mapped implementation of the
simple Caeser cipher function. */
#include "Everything.h"
BOOL cci_f (LPCTSTR fIn, LPCTSTR fOut, DWORD shift)
/* Caesar cipher function.
*       fIn:                    Source file pathname.
*       fOut:                   Destination file pathname.
*       shift:                  Numeric shift value */
{
BOOL complete = FALSE;
                HANDLE hIn = INVALID_HANDLE_VALUE, hOut =
INVALID_HANDLE_VALUE;
                HANDLE hInMap = NULL, hOutMap = NULL;
                LPTSTR pIn = NULL, pInFile = NULL, pOut = NULL, pOutFile = NULL;
                __try {
                        LARGE_INTEGER fileSize;
                        /* Open the input file. */
                        hIn = CreateFile (fIn, GENERIC_READ, 0, NULL,
                                OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
                        if (hIn == INVALID_HANDLE_VALUE)
                                ReportException (_T ("Failure opening input file."), 1);
/* Get the input file size. */
                        if (!GetFileSizeEx (hIn, &fileSize))
                                ReportException (_T ("Failure getting file size."), 4);
                        /* This is a necessar, but NOT sufficient, test for mappability on 32-bit
systems S
                         * Also see the long comment a few lines below */
                        if (fileSize.HighPart > 0 && sizeof(SIZE_T) == 4)
                                ReportException (_T ("This file is too large to map on a Win32
system."), 4);
                        /* Create a file mapping object on the input file. Use the file size. */
                        hInMap = CreateFileMapping (hIn, NULL, PAGE_READONLY, 0, 0,
NULL);
                        if (hInMap == NULL)
                                ReportException (_T ("Failure Creating input map."), 2);
/* Map the input file */
                        /* Comment: This may fail for large files, especially on 32-bit systems
                         * where you have, at most, 3 GB to work with (of course, you have much
less
                         * in reality, and you need to map two files.
                         * This program works by mapping the input and output files in their
entirety.
                         * You could enhance this program by mapping one block at a time for
each file,
```

```
                        * much as blocks are used in the ReadFile/WriteFile implementations.
This would
                        * allow you to deal with very large files on 32-bit systems. I have not
taken
                        * this step and leave it as an exercise.
                        */
                        pInFile = MapViewOfFile (hInMap, FILE_MAP_READ, 0, 0, 0);
                        if (pInFile == NULL)
                                ReportException (_T ("Failure Mapping input file."), 3);
/*  Create/Open the output file. */
                        /* The output file MUST have Read/Write access for the mapping to
succeed. */
                        hOut = CreateFile (fOut, GENERIC_READ | GENERIC_WRITE,
                                0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL,
NULL);
                        if (hOut == INVALID_HANDLE_VALUE) {
                                complete = TRUE; /* Do not delete an existing file. */
                                ReportException (_T ("Failure Opening output file."), 5);
                        }
                        /* Map the output file. CreateFileMapping will expand
                        the file if it is smaller than the mapping. */
                        hOutMap = CreateFileMapping (hOut, NULL, PAGE_READWRITE,
fileSize.HighPart, fileSize.LowPart, NULL);
                        if (hOutMap == NULL)
ReportException (_T ("Failure creating output map."), 7);
                        pOutFile = MapViewOfFile (hOutMap, FILE_MAP_WRITE, 0, 0,
(SIZE_T)fileSize.QuadPart);
                        if (pOutFile == NULL)
                                ReportException (_T ("Failure mapping output file."), 8);
                        /* Now move the input file to the output file, doing all the work in
memory. */
                        __try
                        {
                                CHAR cShift = (CHAR)shift;
                                pIn = pInFile;
                                pOut = pOutFile;
                                while (pIn < pInFile + fileSize.QuadPart) {
                                        *pOut = (*pIn + cShift);
                                        pIn++; pOut++;
}
                                complete = TRUE;
                        }
                        __except(GetExceptionCode() == EXCEPTION_IN_PAGE_ERROR ?
EXCEPTION_EXECUTE_HANDLER : EXCEPTION_CONTINUE_SEARCH)
                        {
                                complete = FALSE;
```

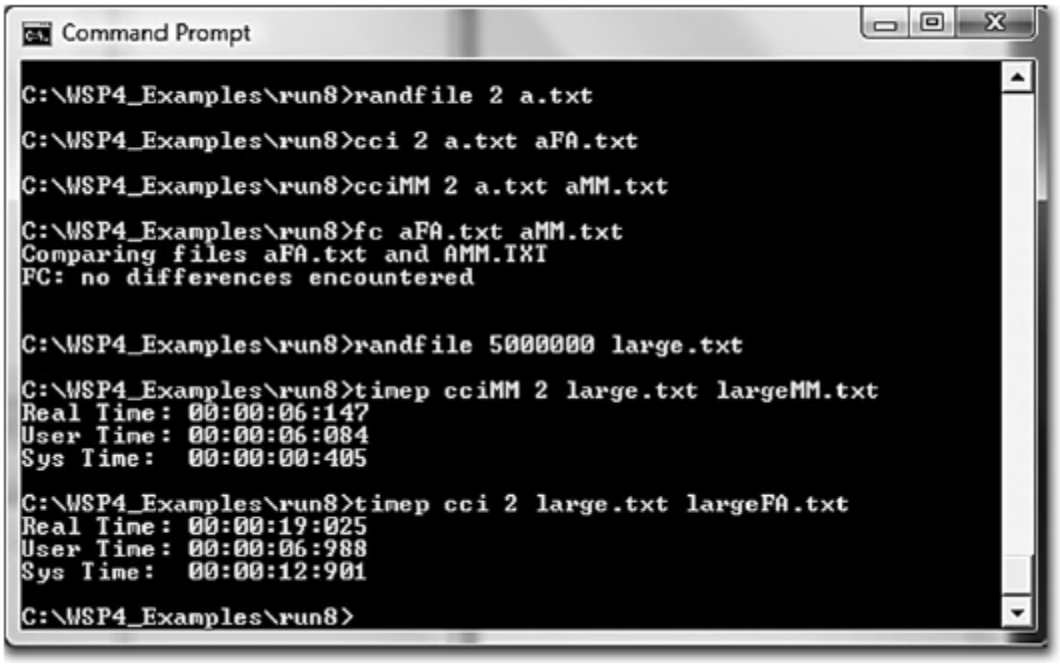
```
                        ReportException(_T("Fatal Error accessing mapped file."), 9);
                    }
                    /* Close all views and handles. */
                    UnmapViewOfFile (pOutFile); UnmapViewOfFile (pInFile);
                    CloseHandle (hOutMap); CloseHandle (hInMap);
                    CloseHandle (hIn); CloseHandle (hOut);
                    return complete;

}
            __except (EXCEPTION_EXECUTE_HANDLER) {
                    if (pOutFile != NULL) UnmapViewOfFile (pOutFile); if (pInFile !=
NULL) UnmapViewOfFile (pInFile);
                    if (hOutMap != NULL) CloseHandle (hOutMap); if (hInMap != NULL)
CloseHandle (hInMap);
                    if (hIn != INVALID_HANDLE_VALUE) CloseHandle (hIn); if (hOut !=
INVALID_HANDLE_VALUE) CloseHandle (hOut);
                    /* Delete the output file if the operation did not complete successfully. */
                    if (!complete)
                        DeleteFile (fOut);
                    return FALSE;               }
}
```

Topic 77: Sorting a Memory-Mapped File

Content:

Another advantage of memory mapping is the ability to use convenient memory based algorithms to process files. Sorting data in memory, for instance, is much easier than sorting records in a file.

Program explained in topic 77 sorts a file with fixed-length records. This program, called sortFL, is similar to Program explaining example of sorting with binary search tree that it assumes an 8-byte sort key at the start of the record, but this example is restricted to fixed records.



The sorting is performed by the ***<stdlib.h>*** C library function ***qsort***. Notice that ***qsort*** requires a programmer-defined record comparison function.

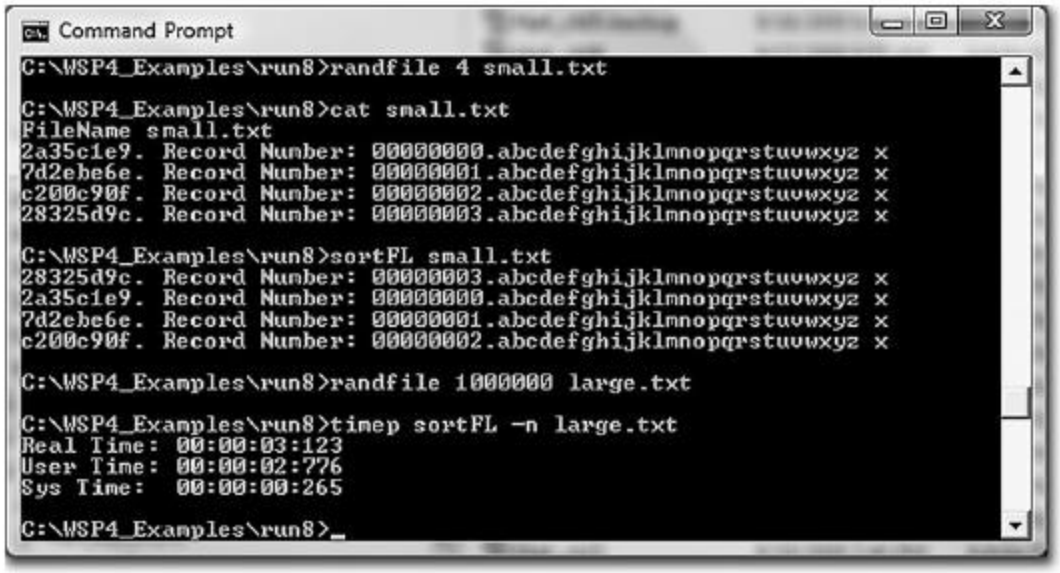
This program structure is straightforward. Simply create the file mapping on a temporary copy of the input file, create a single view of the file, and invoke qsort. There is no file I/O. Then the sorted file is sent to standard output using ***_tprintf***, although a null character is appended to the file map.

Exception and error handling are omitted in the listing but are in the Examples solution on the recommended book's Website.

Program: Sorting a File with Memory Mapping

```
#include<everything.h>
typedef struct  _RECORD
{
        TCHAR key[KEY_SIZE];
        TCHAR data[DATALEN]
} RECORD;
```

```
#define RECSIZE sizeof (RECORD)

int _tmain (int argc, LPTSTR argv[])
{
   HANDLE hFile = INVALID_HANDLE_VALUE, hMap = NULL;
   LPVOID pFile = NULL;
   LARGE_INTEGER fileSize;
   TCHAR tempFile[MAX_PATH];
   LPTSTR pTFile;

   /* Create the name for a temporary file to hold a copy of
      the file to be sorted. Sorting is done in the temp file.
   _stprintf (tempFile, _T ("%s.tmp"), argv[1]);
   CopyFile (argv[1], tempFile, TRUE);

   /* Map the temporary file and sort it in memory. */
   hFile = CreateFile (tempFile, GENERIC_READ | GENERIC_WRITE,
         0, NULL, OPEN_EXISTING, 0, NULL);
   GetFileSizeEx (hFile, &fileSize);
   fileSize.QuadPart += 2;
   hMap = CreateFileMapping (hFile, NULL, PAGE_READWRITE,
         fileSize.HighPart, fileSize.LowPart, NULL);
   pFile = MapViewOfFile (hMap, FILE_MAP_ALL_ACCESS, 0, 0, 0);

   qsort (pFile, FsLow / RECSIZE, RECSIZE, KeyCompare);
                          /* KeyCompare is as in Program 5-2. '

   /* Print the sorted file. */
   pTFile = (LPTSTR) pFile;
   pTFile[fileSize.QuadPart/TSIZE] = '\0';
   _tprintf (_T ("%s"), pFile);
   UnmapViewOfFile (pFile);
   CloseHandle (hMap);
   CloseHandle (hFile);
   DeleteFile (tempFile);
   return 0;
}
```



Topic 78: Using Base Pointer

Content:

File maps are convenient, as the preceding examples demonstrate. Suppose, however, that the program creates a data structure with pointers in a mapped file and expects to access that file in the future. Pointers will all be relative to the virtual address returned from MapViewOfFile, and they will be meaningless when mapping the file the next time. The solution is to use based pointers, which are actually offsets relative to another pointer. The Microsoft C syntax, available in Visual C++ and some other systems, is:

type _based (base) declarator

Here are two examples.

LPTSTR pInFile = NULL;

DWORD _based (pInFile) *pSize;

TCHAR _based (pInFile) *pIn;

Notice that the syntax forces use of the *, a practice that is contrary to Windows convention but which the programmer could easily fix with a typedef.

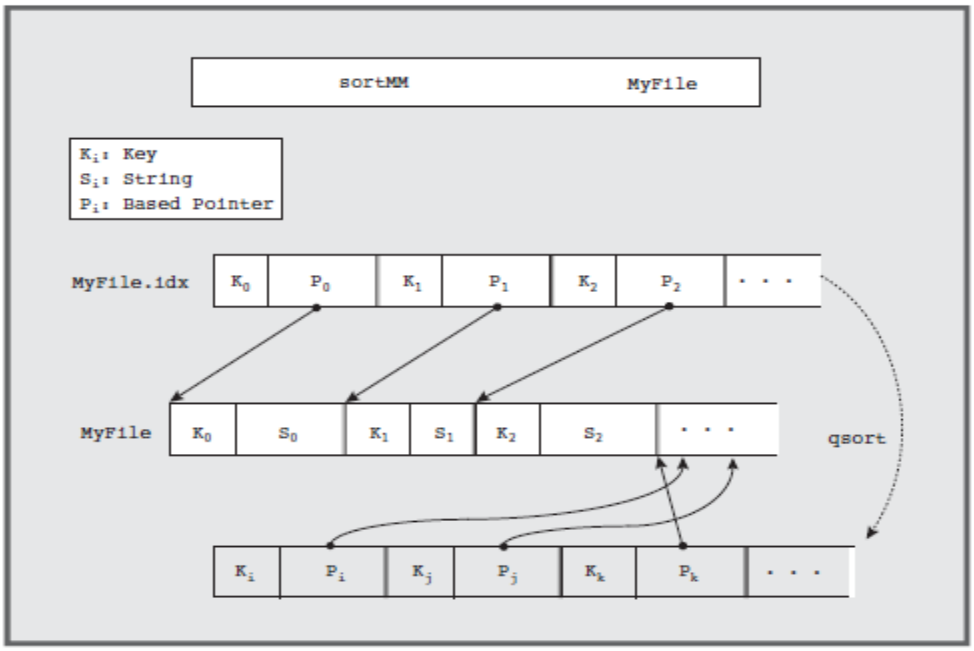
Topic 79: Base Pointer Example

Previously we developed a program that sorted record stored in memory mapped file. Sorting huge files each time they are accessed is not feasible, so permanent indexes are maintained on required key.

Using the address returned by MapViewOfFile() for maintaining indexes is meaningless as the address is liable to change in each call to API. A simple methodology is to maintain an array of record and then build an index for the records. And subsequently use the index to access records directly.

Following example program implements this concept in the given way:

The program uses record of varying sizes in a file. It uses the first field of each record as the key of 8 characters. There are two file mapping. One mapping maps the original file and the other maps the index file. Each record in index file contains a key and the pointer location into the original file for the record containing that key. Once index file is created it can be easily used later. Subsequently, index file records can be sorted for faster searching. The input file remains unchanged. Pictorial Representation of the example is attached below;



Topic 80: Dynamic Link Library

We have previously seen the example use of memory mapped files in windows. This a fundamental feature of windows. Windows itself uses this feature while working with Dynamic Link Libraries (DLLs). DLLs are one of the most important components of windows on which many high-level technologies depend like COM.

Static Linking

The conventional approach is to gather all the source code and library functions attach them and encapsulate them into a single executable file. This approach is simple but has few disadvantages.

- The executable image will be large as it contains all library functions. Hence it will consume more disk space and will require large physical memory to run.
- If a library function updates the whole program will require recompilation.
- There can be many programs that require a library function. Each program will have static copy of its own. Hence, resources requirement will increase.
- It will reduced portability as a program compiled with certain environment setting will run same functions in different environment where some other version might be.

How DLLs Solve this Problem

Using DLLs the library functions are not linked at compile time. They are linked at program load time (implicit linking) or at run time (explicit linking).

As a result the size of executable package is smaller. DLLs can be easily used to create shared libraries which can be used by multiple programs concurrently. Only a single copy of shared DLLs is placed in memory. All the processes sharing the DLL map the DLL space onto their program space. Each program will have its own copy of DLL global variables.

New versions or updates can be simply supported by just providing a new DLL without the need of recompiling main code. The library runs in the same processes as the calling the program.

Importance of DLLs

DLLs are used in almost all modern operating systems. DLLs are most important in case of windows as they are used to implement OS interfaces. The entire Windows API is supported by a set of DLLs which are invoked to call kernel services. The DLL code can be shared by multiple processes. DLL function when invoked by a process runs in process space. Therefore, it can use resources of the calling process such as file handles and thread stack. DLLs should be written in Thread-safe manner. A DLL exports variables as well as function entry points.

Topic 81: Implicit Linking

Implicit linking is the easier of the two techniques. Functions defined in a DLL are collected and build as DLL. The build process builds a .LIB file which is a stub for actual code. The stub is linked to the calling program at build time. It provides a place holder for each function in the DLL.

The place holder/stub will call the original function in the DLL. This file should be placed in common user library directory for the project. The build process also constructs the DLL that contain the original binary image for the functions. This File is usually placed in the same directory as the application.

Function interfaces defined in DLLs should be exported carefully.

Topic 52: Exceptions Handling Sequence

Exception Handling sequence for python is given below:

try:
  You do your operations here;
  .....
except ExceptionI:
  If there is ExceptionI, then execute this block.
except ExceptionII:
  If there is ExceptionII, then execute this block.
  .....
else:
  If there is no exception then execute this block.

Topic 53: Floating-Point Exceptions

IEEE defines a standard for floating-point exceptions it is called IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754). This standard have defined five types of floating-point exception:

- Invalid operation
- Division by zero
- Overflow
- Underflow
- Inexact calculation

Topic 54: Errors and Exceptions

Errors	Exception
Errors are usually raised by the environment in which the application is running.	Exceptions are caused by the code of the app the code belongs to.
It is not possible to recover from an error.	The use of **try-catch** blocks can handle exceptions and recover the system from exception.
Errors occur at run-time and are unknown by the compiler.	Exceptions may or may not be caught by the compiler.
Programmers include an explicit test to check for error,for example whether a file read/write operation has failed.	An exception could occur nearly anywhere, and it is practical to test for an exception.

Topic 55: Treating Errors as Exceptions

Programmer can use ReportError function to report the error and let windows treat it and terminate the process.But it have its own limitations.

• A fatal error terminates the entire process when only a single thread should terminate.

• The programmer may require to continue program execution rather than terminate the process.

• Synchronization resources, such as events or semaphores, will not be released in most of the circumstances.

Topic 56: Termination Handlers

Termination handler provides same functionality as an exception handler,but it is executed when a thread leaves a block as a result normal program flow as well as when an exception occurs. A termination handler cannot diagnose an exception.
A termination handler consists of the following elements.

- A guarded body of code
- A block of termination code to be executed when the flow of control leaves the guarded body

Termination handlers are declared in language-specific syntax. Using the Microsoft C/C++ Optimizing Compiler, they are implemented using **__try** and **__finally**.

The guarded body of code can be a block of code, a set of nested blocks, or an entire procedure or function.
The termination block is executed when the flow of control leaves the guarded body, regardless of whether the guarded body terminated normally or abnormally.

Topic 57: Better Programs with Termination Handlers

Termination and exception handlers allow you to make your program more robust by both simplifying recovery from errors and exceptions and helping to ensure that resources and file locks are freed at critical junctures.

Consider the below given code where we are checking for invalid handle and ReportingException.

If (hin == INVALID_HANDLE_VALUE)

ReportException(argv[iFile],1);
If(!GetFileSizeEx(hIn, &fsize) || fsize.HighPart > 0)
ReportException(_T("FIle is Too Large",1));

Topic 58: Filter Functions

The filter function identifies the exception type and based on the type of the exception, the handler can treat each exception differently. In the following program, the exception handling and termination of a program are illustrated using a filter function.

The program generates an exception based on the type of the exception entered by the user. The floating point exceptions are enabled with the help of controlfp function and old status is saved in the fpOld. The try block mentions the different cases of exception generation with the help of a switch statement.

In the except clause, there is a filter function calling another function GetExceptionInformation(). eCategory is a reference variable whose value determines the type of exception being thrown. For the outer try, there is a finally clause used for any abnormal termination of the program. The old value of the floating point mask is also restored at this stage.

Now we will see how the values in the ecategory reference variable are placed.

User generated exceptions are identified based on the masking operation generating zero as a result.

Topic 59: Console Control Handles

Console control handlers are quite similar to the mechanism of exception handlers. Normal exceptions respond to several asynchronous events like division by zero, invalid page fault etc., but they do not respond to console related events like Ctrl+C. Console control handlers can detect and handle the events that are console related. The API SetConsoleCtrlHandler() is used to add console handlers.

BOOL SetConsoleCtrlHandler(PHANDLER_ROUTINE HandlerRoutine, BOOL Add);

The API takes the address of the HandlerRoutine and Add Boolean as parameters. There can be a number of handler routines if the Add parameter is set as TRUE. If the HandlerRoutine parameter is set as NULL and Add is TRUE, then the Ctrl-C signal will be ignored.

BOOL HandlerRoutine (DWORD dwCtrlType);

The return type of HanlderRoutine should be Boolean taking only one parameter of type DWORD. The HandlerRoutine() function takes a DWORD as a parameter and returns a Boolean value.

dwCtrlType identifies the signal and its values can be:

1. CTRL_C_EVENT: Indicates that a Ctrl-C sequence was entered through keyboards.

2. CTRL_CLOSE_EVENT: Indicates that the console window is being closed.

3. CTRL_BREAK_EVENT: Indicates Ctrl-Break sequence.

4. CTRL_LOGOFF_EVENT: Indicates the user is trying to logoff.

5. CTRL_SHUTDOWN_EVENT: Indicates the user is trying to shut down.

The handler routine will be invoked if a console exception is detected. Handler routine runs in an independent thread within the process. Raising an exception within the handler routine will not interfere with the working of the original routine that created the handler routine. Signals apply to the whole process, while exception applies to a single thread.

Usually signal handlers are used to perform cleanup tasks whenever a shutdown, close or logoff events are detected. Signal Handler would return a TRUE value in case it takes care of the task or it may return FALSE. In case of FALSE, the next handler in the chain is invoked. Signal handler chain is invoked in the reverse order of which they are set up in and the system signal handler is the last one in this chain.

Topic 60: Console Control Handler Example

We have a simple program that set up a console control handler and starts beeping in a loop. The control handler is invoked whenever a console event occurs. The handler handles the event likewise and clears an exitFlag to end the loop of the main function.

```
/* Chapter 4. CNTRLC.C */
/* Catch Cntrl-C signals. */
#include "Everything.h"
static BOOL WINAPI Handler(DWORD cntrlEvent);
static BOOL exitFlag = FALSE;
int _tmain(int argc, LPTSTR argv[])

/* Beep periodically until signaled to stop. */
{
        /* Add an event handler. */
    if (!SetConsoleCtrlHandler(Handler, TRUE))
```

```
                ReportError(_T("Error setting event handler"), 1, TRUE);

        while (!exitFlag) { /* This flag is detected right after a beep, before a handler exits */
                Sleep(4750); /* Beep every 5 seconds; allowing 250 ms of beep time. */
                Beep(1000 /* Frequency */, 250 /* Duration */);
        }
}
        _tprintf(_T("Stopping the main program as requested.\n"));
        return 0;
}
BOOL WINAPI Handler(DWORD cntrlEvent)
{
        switch (cntrlEvent) {
                /* The signal timing will determine if you see the second handler message */
                case CTRL_C_EVENT:
                        _tprintf(_T("Ctrl-C received by handler. Leaving in 5 seconds or
                                less.\n"));
                        exitFlag = TRUE;
                        Sleep(4000); /* Decrease this time to get a different effect */
                        _tprintf(_T("Leaving handler in 1 second or less.\n"));
                        return TRUE; /* TRUE indicates that the signal was handled. */
                case CTRL_CLOSE_EVENT:
                        _tprintf(_T("Close event received by handler. Leaving the handler in 5
                                seconds or less.\n"));
                        exitFlag = TRUE;
                        Sleep(4000); /* Decrease this time to get a different effect  */
                        _tprintf(_T("Leaving handler in 1 second or less.\n"));
                        return TRUE; /* Try returning FALSE. Any difference? */
                default:
                        _tprintf(_T("Event: %d received by handler. Leaving in 5 seconds or
                                less.\n"), cntrlEvent);
                        exitFlag = TRUE;
                        Sleep(4000); /* Decrease this time to get a different effect */
                        _tprintf(_T("Leaving handler in 1 seconds or less.\n"));
                        return TRUE; /* TRUE indicates that the signal was handled. */
        }
}
```

The program can be terminated by the user either by closing the console or with a Ctrl-C. The handler will register with windows using the SetConsoleCtrlHandler(Handler, TRUE) function. The handler will be activated upon the occurrence of any console event. If the registration of any handler fails due to any reason, then an error message will be printed.

Topic 61: Vectored Exceptions Handling

Exception handling functions can be directly associated with an exception just like console handlers. In case a vectored exception is set up, then windows first looks for Vectored Exception Handlers (VEH) and then unwinds the stack.

No __try and __catch keywords are required with VEH and they are like console control handlers. Windows provides a set of APIs for VEH management as follows.

PVOID WINAPI AddVectoredExceptionHandler(ULONG FirstHandler, PVECTORED_EXCEPTION_HANDLER VectoredHandler);

The given API has two parameters; FirstHandler is the parameter used to specify the order in which the handler executes. Non-zero value indicates that it will be the first one to execute and zero specifies it to be the last. If there are more than one handlers setup with zero value, then they will be invoked in the order they are added using AddVectoredExceptionHandler(). Return value is NULL in case of failure, otherwise it returns the Handle to the Vectored Exception Handler.

A call to RemoveVectoredExceptionHandler() is used to remove the vectored exception from a chain. Following is the structure of a vectored exception handler to be set up by the AddVectoredExceptionHandler().

LONG WINAPI VectorHandler(PEXCEPTION_POINTERS ExceptionInfo);

PEXCEPTION_POINTER is a pointer to EXCEPTION_POINTER structure that was previously retrieved by GetExceptionInformation().

Pitfalls of Vectored Exceptions

The exception handler function should be fast and must return as quickly as possible, therefore it should not have a lot of code. The VEH should neither perform any blocking operation like the Sleep() function nor use any synchronization objects. Typically a VEH would access exception information structure, do some minimal processing, and set a few flags.

VEH uses the same return values as SHE.

EXCEPTION_CONTINUE_EXECUTION : No more handlers invoke, exception stack does not unwind, and the execution continues from the point where the exception occurred.

EXCEPTION_CONTINUE_SEARCH : The next VEH handler is probed, if there are no additional VEH handlers, then SEH stack is unwound.

Topic 62: Memory Management

Dynamic Memory

Need of dynamic memory arises whenever dynamic data structures like search tables, trees, linked lists etc. are used. Windows provides a set of APIs for handling dynamic memory allocation.

Windows also provides memory mapped files which allows direct movement of data to and from user space and files without the use of file APIs. Memory Mapped files can help conveniently handle dynamic data structure and make file handling faster because they are treated just like memory. It also provides a mechanism for memory sharing among processes.

Windows Memory Management

Windows essentially uses two API platforms i.e. Win32 and Win64.

The Win32 API uses pointers of size 32 bits, hence the virtual space is 2^32. All data types have been optimized for 32 bit boundaries. Win64 uses a virtual space of 2^64 (16 Exabytes).

A good strategy is to design an application in such a way that it could run in both modes without any change in code.

Win32 makes at least half of the virtual space (8GB) accessible to a process and the rest of the space is reserved by the system for shared data, code, and drivers etc. Overall Windows provides a large memory space available to user programs and hence requires optimal management.

Topic 63: Windows Memory Management Overview

Dynamic Memory

Need of dynamic memory arises whenever dynamic data structures like search tables, trees, linked lists etc. are used. Windows provides a set of APIs for handling dynamic memory allocation.

Windows also provides memory mapped files which allows direct movement of data to and from user space and files without the use of file APIs. Memory Mapped files can help conveniently handle dynamic data structure and make file handling faster because they are treated just like memory. It also provides a mechanism for memory sharing among processes.

Windows Memory Management

Windows essentially uses two API platforms i.e. Win32 and Win64.

The Win32 API uses pointers of size 32 bits, hence the virtual space is 2^32. All data types have been optimized for 32 bit boundaries. Win64 uses a virtual space of 2^64 (16 Exabytes).

A good strategy is to design an application in such a way that it could run in both modes without any change in code.

Win32 makes at least half of the virtual space (8GB) accessible to a process and the rest of the space is reserved by the system for shared data, code, and drivers etc. Overall Windows provides a large memory space available to user programs and hence requires optimal management.

Further information about the parameters of Windows Memory Management can be probed using the following API.

VOID GetSystemInfo(LPSYSTEM_INFO lpSysInfo)

The API returns a pointer to SYSTEM_INFO structure. The structure contains various information regarding the system like page size, granularity, and application's physical memory address space.

Topic 64: Introduction to Heaps

A programmer allocates memory dynamically from a heap. Windows maintains a pool of heaps and a process can have many heaps. Traditionally, one heap is considered enough. But several heaps may be required to make a program more efficient.

In case a single heap is sufficient, then a runtime library function for heap allocation like malloc(), free(), calloc(), realloc() might be enough.

Heap is a windows object and hence is accessed by a handle. Whenever you require allocating memory from heap, you need a heap handle. Every process in windows has a default heap which can be accessed through the following API.

HANDLE GetProcessHeap(VOID)

The API returns a handle to the process heap. NULL is returned in case of failure and not INVALID_HANDLE_VALUE.

However, due to a number of reasons it would be desirable to have more than one heap. Sometimes it is convenient to have distinct heaps for different data structures.

Separate Heaps

Following are the benefits of separate heaps:

1. If a distinct heap is assigned to each thread, then each thread will only be able to use the memory allocated to each thread.

2. Separation of memory space among threads reduces memory contention.

3. Fragmentation is reduced when one fixed size data structure is allocated from a single heap.

4. Allocating a single heap among each thread simplifies synchronization.

5. If a single heap contains complex data structures, then they can be easily de-allocated with a single API call by de-allocating the entire heap. We will not need complex de-allocation algorithms in such cases.

6. Small heaps for a single data structure reduces the chances of page faults as per the principle of locality.

Topic 65: Creating Heaps

We can create a new heap using HeapCreate() API and its size can be set to zero. The API adjusts the heap size to the nearest multiple of page size. Memory is committed to the heap initially, rather than on demand. In case the memory requirements increase than the initial, more pages will automatically be allocated to the heap up to maximum size allowed.

If the required memory is not known, then deferring memory commitment is a good practice as heap is a limited resource. Following is the syntax of the API used to create new heaps.

HANDLE HeapCreate(DWORD flOptions, SIZE_T dwInitialSize, SIZE_T dwMaximumSize);

dwMaximumSize if non-zero, determines the maximum limit of the heap memory set by the user. Heap is not grow-able beyond this point. In case it's zero, then the heap is grow-able to the extent of the virtual memory space available for the heap.

dwInitialSize is the initial size of the heap set by the programmer. SIZE_T is used to enable portability. Based on the win32 or win64 platforms, SIZE_T will be 32 or 64 bit wide.

flOptions is usually a combination of the following three flags predefined in Windows:

1. HEAP_GENERATE_EXCEPTIONS: This flag will raise exceptions in case there is any failure while allocating memory to heap. Exceptions are not generated by CreateHeap(), rather they may occur at the time of allocation.

2. HEAP_NO_SERIALIZE: This option provides a slight performance improvement when serialization of data is not required.

3. HEAP_CREATE_ENABLE_EXECUTE: It is an advanced option that allows you to execute code from heap. Usually an exception will occur if data from heap is interpreted as code due to the data execution prevention (DEP) restriction by Windows.

CloseHandle() is not appropriate to dispose of a heap handle rather HeapDestroy() is used as follows.

BOOL HeapDestroy(

HANDLE hHeap

);

hHeap is the handle to a previously created heap. Do not use the handle obtained from GetProcessHeap() because it may raise an exception. This is an easy way to get rid of all the contents of the heap including complex data structures.

Topic 66: Managing Heap Memory

Once a heap is created, it does not allocate memory that is directly available to the program. Rather, it only creates a logical structure of heap that will be used to allocate new memory blocks. Memory blocks are allocated using heap memory allocation APIs like HeapAlloc() and HeapReAlloc().

LPVOID HeapAlloc(HANDLE hHeap, DWORD dwFlags, SIZE_T dwBytes);

hHeap is the handle of the heap from which memory is to be allocated. dwFlags are quite similar to the flags used in HeapCreate().

HEAP_GENERATE_EXCEPTIONS: This flag will raise exceptions in case there is any failure while allocating memory to heap. Exceptions are not generated by CreateHeap(), rather they may occur at the time of allocation.

HEAP_NO_SERIALIZE: This option provides a slight performance improvement when serialization of data is not required.

The first flag HEAP_GENERATE_EXCEPTION is ignored if it is already set in HeapCreate().

HEAP_ZERO_MEMORY: Specifies that the allocated memory will be initialized to zero.

dwBytes is the size of the memory block to be allocated. For non-grow-able heap, its 0x7FFF8 approximately equivalent to 0.5 MB.

The return value of the function is LPVOID. This is the address of the allocated memory block. Use this pointer in a formal way and there is no need to make any reference to the Heap handle.

In case HEAP_GENERATE_EXCEPTION flag is set, an exception is generated if failure occurs. The exception is either STATUS_NO_MEMORY or STATUS_ACCESS_VIOLATION.

If the exception flag is not set, then NULL is returned by HeapAlloc() and the GetLastError() does not work on HeapAlloc().

BOOL HeapFree(HANDLE hHeap, DWORD dwFlags,

_Frees_ptr_opt_ LPVOID lpMem);

hHeap is the heap handle from which memory is to be allocated. dwFlags should be 0 or set to HEAP_NO_SERIALIZE. lpMem should be the pointer previously returned by HeapAlloc() or HeapReAlloc().

Return value of FALSE will indicate a failure. GetLastError() can be used to get the error.

LPVOID HeapReAlloc(HANDLE hHeap, DWORD dwFlags,

_Frees_ptr_opt_ LPVOID lpMem, SIZE_T dwBytes);

hHeap is the heap handle from which memory is to be allocated. dwFlags: HEAP_GENERATE_EXCEPTION and HEAP_NO_SERIALIZE are quite similar to the flags used in HeapAlloc().

HEAP_ZERO_MEMORY: only the newly allocated memory is set to zero (in case dwBytes is greater than the previous allocation).

HEAP_REALLOC_IN_PLACE_ONLY: It indicates the newly allocated block should lie contiguously to the previously allocated block.

lpMem: specifies the pointer to the block previously allocated to the same heap hHeap.

dwBytes: It refers to the block size to be allocated that can be lesser or greater than the previous allocation. But the same restriction as HeapAlloc applies i.e. the block size cannot be greater than 0x7FFF8.

Topic 67: Heap Size and Serialization

Some programs may require to determine the size of allocated blocks in the Heap. The size of the allocated block is determined using the API HeapSize() as follows.

```
SIZE_T HeapSize(
  HANDLE  hHeap,
  DWORD   dwFlags,
  LPCVOID lpMem
);
```

The function returns the size of the block or zero in case of failure. The only valid dwFlag is HEAP_NO_SERIALIZE.

Serialization

Serialization is required when dealing with concurrent threads using some common resource. Serialization is not required if threads are autonomous and there is no possibility of concurrent threads disrupting each other.

The flag HEAP_NO_SERIALIZE provides an estimated performance advantage of 16%. Using this flag can improve performance if there are no conflicts. This flag can be conveniently used if:

a. The program is single threaded.

b. Each thread has its own heap that is insulated from other threads.

c. The program has its own mutual exclusion mechanism.

Heap Exceptions

Heap exceptions are enabled using the flag HEAP_GENERATE_EXCEPTION. This allows the program to close open "handles" before a program terminates. There can be two scenarios with this option:

a. STATUS_NO_MEMORY: It means there is no more memory to be allocated because the heap has reached its limits.

b. STATUS_ACCESS_VIOLATION: It indicates that the heap has been corrupted. Either the heap has been accessed as code, or has been accessed beyond its bounds.

There are some other functions that can be used while working with heaps. For example,

HeapSetInformation() can be used to enable low fragmentation mode. It can also be used to allow termination of a thread upon heap's corruption.

HeapCompact() allows finding the largest allocated block in the heap.

HeapValidate() allows detecting heap corruption.

HeapWalk() enumerates the allocated blocks on heap.

HeapLock() and HeapUnLock() allow serializing access to the heap.

Topic 68: Using Heaps

Using Memory Management APIs

Till now we have used the Memory Management APIs for allocating, reallocating and deallocating heaps. We can also get the size of a heap through an API.

A typical methodology of dealing with heaps should be to get a heap handle either using HeapCreate() or GetProcessHeap(). Use the handle obtained from the above to allocate memory blocks from the heap using HeapAlloc(). If some block needs to be deallocated, use HeapFree(). Before the program is terminated or when the heap is not required, use HeapDestroy() to dispose of the heap.

It is convenient not to mix up windows heap API and Run Time Library functions. Anything allocated with C library functions should also be deallocated with C library functions.

Topic 69: Working with Heaps

A Sorting File example using heaps

The example is formulated using two heaps. The first one will be a node heap, while the other is a record heap. Node heap will be used to build a tree, while the data heap will be used to store keys.

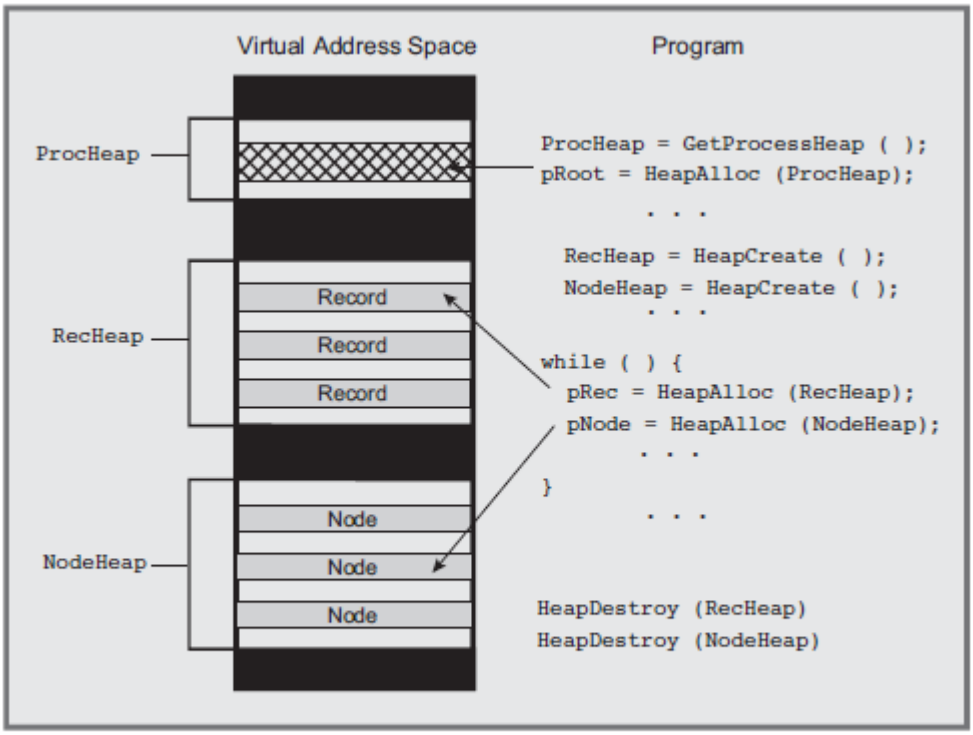


Figure 5–2 Memory Management in Multiple Heaps

Following are the three heaps as shown in the figure:

1. ProcHeap

2. RecHeap

3. NodeHeap

ProcHeap contains the root address, but RecHeap stores the records. NodeHeap on the other hand stores the nodes when they are created. Sorting will be performed in the NodeHeap that gives a reference to be searched in the RecHeap. The data structure will be maintained in the NodeHeap. At the end, all the heaps will be destroyed except ProcHeap because it is created using GetProcessHeap().

Topic 82: **Exporting and Importing Interfaces**

For a DLL function to be useful for an application exporting it, it must be declared as exportable. This can be done by using .DEF file or by using __declspec (dllexport) storage modifier.

__declspec (dllexport)DWORD MyFunction (…)

The build process will generate a .LIB file and .DLL file. The .LIB files will a stub for function calls while DLL file will hold the actual code. Similarly the calling program should use the __declspec (dllimport) Storage modifier.

Following Macro can be used with the library file code

#if defined (MYLIBRARY_EXPORTS)

#define LIBSPEC _declspec(dllexport)

#elif defined (__cplusplus)

#define LIBSPEC extern “C” __declspec(dllimport)

#else

#define LIBSPEC __declspec (dllimport)

#endif

LIBSPEC SWORD MyFunction (…)

If you are using Visual C++ compiler then this task is automatically performed by the compiler. When building the calling program you need to specify the .LIB file. When executing the calling program the .DLL file must be placed in the same directory as the application. Following is the default DLL search safe order for explicit and implicit linking.

- Directory containing the application
- The system directory. This path can be determined by GetSystemDirectory()
- The windows directory (GetWindowsDirectory())
- The current directory
- Directories specified by the PATH environment variable

Topic 83: **Explicit Linking**

Explicit linking requires the program to explicitly a specific DLL to be loaded or freed. Once the DLL is loaded then the program obtains the address of the specific entry point and uses that address as a pointer in function call.

The function is not declared in the calling program. Rather a pointer to a function is declared. The functions required are:

HMODULE LoadLibrary(LPCTSTR lpLibFileName);

HMODULE LoadLibraryEx(LPCTSTR lpLibFileName,

HANDLE hFile,

DWORD dwFlags);

HMODULE is NULL in case of failure.

File name need not mention the extension. The file path must be valid. See MSDN for details of dwFlags Once the DLL is loaded. The programmer needs to obtain an entry point (procedure address) into the DLL. This is done using:

FARPROC GetProcAddress(HMODULE hModule, LPCSTR lpProcName);

hModule is the module handle obtained for the DLL. lpProcName cannot b UNICODE. It is the name of the function whose entry point is to be obtained. Its return type is FARPROC. This is the far address of the function. A C type function pointer declaration can be easily used to invoke the function in DLL and pass its parameters.

Topic 84: **Explicit Linking a file conversion function**

Previously, we have studied many file conversion functions. Some used memory mapped IO, some used file operations. Some performed file conversion some performed file encryption.

Now, we take a look at how we can encapsulate these functions in a DLL and invoke them explicitly.

Following example shows explicit linking of these functions.

HMODULE hDLL;

FARPROC pcci;

TCHAR YNResp[5] = YES;

if (argc < 5)

ReportError (_T("Usage: cciEL shift file1 file2 DllName"), 1, FALSE);

/* Load the cipher function. */

```
hDLL = LoadLibrary (argv[4]);

 if (hDLL == NULL)

ReportError (_T("Failed loading DLL."), 4, TRUE);

 /* Get the entry point address. */

 pcci = GetProcAddress (hDLL, _T("cci_f"));

if (pcci == NULL)

ReportError (_T ("Failed of find entry point."), 5, TRUE);

cci_f = (BOOL (__cdecl *)(LPCTSTR, LPCTSTR, DWORD)) pcci;

 /* Call the function. */
```

Topic 85: **DLL Entry Point**

DLL Entry point can be specified optionally when you create a DLL. The code at entry point executes whenever a process attaches to the DLL. In case of implicit linking the DLL attaches at the time of process start and detaches when process ends. In case of explicit linking the process attaches when LoadLibrary()/LoadLibraryEx() is invoked. Also the process detaches when FreeLibrary() is called.

LoadLibraryEx() can also suppress the execution of entry point. Further, entry point is also invoked whenever a thread attaches to the DLL.

The Entry point in DLL is defined as:

```
BOOL DllMain(

HINSTANCE hDll,

DWORD Reason,

LPVOID lpReserved)
```

hDll corresponds to the handle returned by LoadLibrary(). lpReserved if NULL, represent explicit attachment else it represents implicit attachment. Reason will have on the four values

DLL_PROCESS_ATTACH

DLL_THREAD_ATTACH

DLL_THREAD_DETACH

DLL_PROCESS_DETACH

Based on the value of Reason the programmer can decide whether to initialize or free resources. All the calls to DllMain are serialized by the system. Serialization is critically important as DllMain() is supposed to perform initialization operations for each thread. There should be no blocking calls, IO calls or wait functions as they will indefinitely block other threads. A call to other DLLs from DllMain() cannot be performed except for few exceptions. LoadLibrary() and LoadLibraryEx() should never be called from DllMain() as it will create more DLL entry points. DisableThreadLibraryCalls() can be used to disable thread attach/detach calls for a specified instance.

Topic 86: DLL Version Management

Complications

Many times a DLL is upgraded to provide more features and new symbols. Also, multiple processes share a single implementation of DLL. This strength of DLL may also lead to some complications.

- If the new DLL has changed interfaces this render problems for older programs that have not been updated for newer version.
- Application that require newer updated functionality may sometime link with older DLL version.

Version Management

One intuitive resolution to the problem can be by using different directory for each version. But there are several solutions. Use DLL version number as part of the .DLL and .LIB file names eg. Utils_4_0.DLL and Utils_4_0.LIB. Applications requiring DLLs can determine the version requirements and subsequently access files with distinct filename using implicit or explicit linking.

Microsoft has introduced a concept of side by side DLLs or assemblies. This solution requires application to declare its DLL requirements using XML.

Querying DLL Information

Microsoft DLLs support a callback function that version information and more regarding a DLL. This callback function can be used dynamically to query the information regarding DLL. The works as follows:

HRESULT CALLBACK DllGetVersion(

DLLVERSION *pdvi

)

Information regarding the DLL is placed in the DLLVERSION structure. It contains a field cbSize which is the size of the structure,

dwMajorVersion

dwMinorVersion

dwBuilderNumber

dwPlatformID

dwPlatformID can be set of DLLVER_PLATFORM_NT if DLL cannot run on windows 9x or to DLLVER_PLATFORM_WINDOWS if there are no restrictions. cbSize should be set to sizeof(DLLVERSION)

Topic 87: Windows Processes and Threads

Windows Process Management is all about:

- Multitasking Systems
- Multiprocessor systems
- Thread as a unit of execution

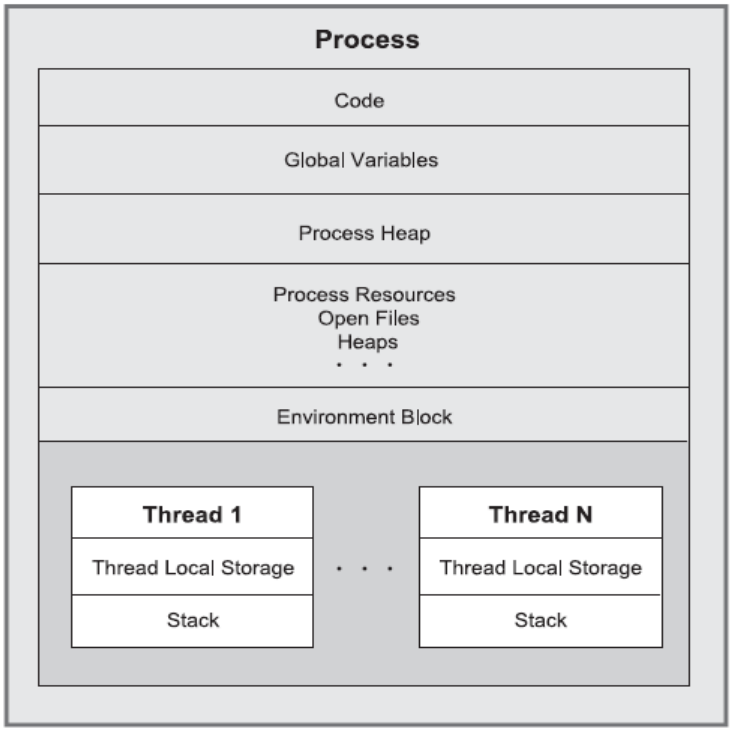
Resources of Processes

From programmer's perspective each process has resources such as one or more threads distinct virtual address space. Although, processes can share memory and files but the process itself lie in an individual virtual memory space.

- One or more code segment including DLLs
- One or more data segment including global variables
- Environment block containing environment variables (current path)

Resources of Threads

Each thread is a unit within the process. Nevertheless, it has resources of its own. Stack: for procedure calls, interrupts, exception handling, and auto variables. Thread Local Storage (TLS): An array like collection of pointers enabling a thread to allocate storage to create its unique data environment. An argument on stack unique for the created thread. A structure containing the context (internal registers status) of the thread.



Topic 88: **Process Creation**

The most fundamental process management function in windows is CreateProcess(). Windows does not have any structure that keeps account of the parent-child processes. The process that creates a child process is considered as parent process. CreateProcess() has 10 parameters. It does not returns a HANDLE. Rather two handles, one for process and one for thread is returned in a parameter struct.

It creates a process with single primary thread. Windows does not have any structure that keeps account of the parent-child processes. The process that creates a child process is considered as parent process.

CreateProcess() has 10 parameters. It does not return a HANDLE. Rather two handles, one for process and one for thread is returned in a parameter struct. One must be very careful while closing both these handles when they are not needed. Closing the thread handle does not terminate the thread it only deletes the reference to the thread within the process.

BOOL CreateProcess(LPCTSTR lpApplicationName, LPTSTR lpCommandLine,

LPSECURITY_ATTRIBUTES lpProcessAttributes, LPSECURITY_ATTRIBUTES

lpThreadAttributes, BOOL bInheritHandles, DWORD dwCreationFlags, LPVOID

lpEnvironment, LPCTSTR lpCurrentDirectory, LPSTARTUPINFO lpStartupInfo,

LPPROCESS_INFORMATION lpProcessInformation);

lpApplicationName and lpCommandLine specify the program name and the command line parameters. lpProcessAttributes and lpThreadAttributes points to the process and thread's security attribute structure. bInheritHandles specifies whether the new process inherits copies of the calling process's inheritable handles. dwCreationFlags combines several flags

CREATE_SUSPENDED: indicates that the primary thread is a suspended thread and will continue if the process invokes ResumeThread()

DETACHED_PROCESS and CREATE_NEW_CONSOLE are mutual exclusive, First flag creates a process without console and second one creates a process with console

CREATE_UNICODE_ENVIRONMENT: should be set if Unicode is being used.

CREATE_NEW_PROCESS_GROUP : Indicates that the new process is the root of new process group. All processes in the same root group receive the control signal if they share the same console.

lpEnvironment : points to an environment block for the new process. If NULL, the process uses the parent's environment.

lpCurDir Specifies the drive and directory of new process. If NULL parent working directory is used.

lpStartupInfo : specifies the main window appearance and standard device handles for new programs.

lpProcInfo is the pointer to the structure containing handles and id for process and thread.

Topic 89: Why Processes need IDs and Handles

IDs are unique to processes for their entire lifetime. ID is invalidated when a process is destroyed. Although, it may be reused by other newly created processes. Alternately, a process can have many handles with different security attributes.

Some process management functions require handles while others require IDs. Handles are required for general purpose handle based functions. Just like file handles and handles for other resources the process handles need to be closed when not required.

The process obtains environment and other information from CreateProcess() call. Once the process has been created then changing this information may have no effect on the child. For example the parent process may change its working directory but it will not have effect on child unless the child changes its own working directory. Processes are entirely independent.

Topic 90: Specifying the Executable Image and the Command Line

Executable Image

Either lpApplicationName or lpCommandLine specifies the executable image. Usually lpCommandLine is used, in which case lpApplicationName is set to NULL. However if lpApplicationName is specified there are rules governing it.

lpApplicationName

lpApplicationName should not be NULL. It should specify the full name of the application including the path. Or use the current path in which case current drive and directory will be used. Include the extension i.e. .EXE or .BAT in the file name. For long names quotes within the string are not required.

lpCommandLine

lpApplicationName should be NULL. Tokens within string are delimited by spaces. First token is the program image name. If the name does not contain the path of the image then the following search sequence is followed.

- Directory of current process image
- The current Directory
- Windows System directory which can be retrieved from GetSystemDirectory()
- Windows Directory which is retrievable from GetWindowsDirectory(). Directories as specified in Environment variable PATH

Command Line

A process can obtain its command line from the usual argv mechanism. Alternately, in windows it can call GetCommandLine(). It's important to know that command line is not a constant string. A program can change the command line. It's advisable that the program makes changes on a copy of command line.

Topic 91: Inheritable Handles

Mostly, a child process requires access to a resource referenced in parent by a handle. If the handle is inheritable then the child can directly receive a copy of open handles in parent. For example standard input and output handles are shared in this pattern. This inheriting of handles is accomplished in this manner. The bInheritHandles flag in the CreateProcess() call determines whether the child process will inherit copies of parent open handles. Also it is necessary to make individual handles inheritable. Use the SECURITY_ATTRIBUTES structure to specify this. It has a flag bInheritFlag which should be set to TRUE. Also the nLength should be set to sizeof(SECURITY_ATTRIBUTES)

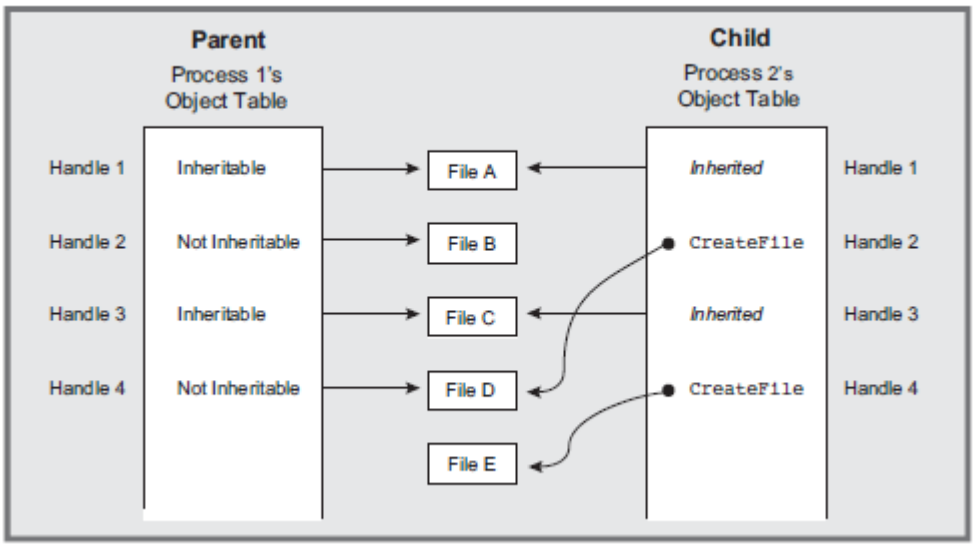
The following code illustrates how this is done programmatically.

```
HANDLE h1, h2, h3;
SECURITY_ATTRIBUTES sa =
    {sizeof(SECURITY_ATTRIBUTES), NULL, TRUE };
...
h1 = CreateFile (..., &sa, ... ); /* Inheritable. */
h2 = CreateFile (..., NULL, ... ); /* Not inheritable. */
h3 = CreateFile (..., &sa, ...);
    /* Inheritable. You can reuse sa. */
```

Topic 92: Passing Inheritable Handles

We have only made the handles inheritable. However, we need to pass the actual values of handles to the child process. Either this is accomplished through InterProcess Communication (IPC) Or it is passed on the child process by setting it up in the STARTUPINFO struct. The latter is a preferred policy as it allows IO redirection and no changes are required in child process. Another approach is to convert the file handles into text and pass them through the command line to the child process.

The handles if are already inheritable then they will readily accessible to the child. Inherited handles are distinct copies. Parent and child might be accessing same file with different file pointer. Each process should close handles.



Content Development

Topic No - 96 (Waiting for a Process)

Content:

The synchronization can be attained easily by the wait process. Windows provides a general-purpose wait function. Which can be used to wait for a single object and also multiple objects in a group. Windows send a signal to the waiting process when the process terminates

We have two API's

i)DWORD waitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds);

ii)DWORD waitForMultipleObjects(DWORD nCount, const HANDLE*lpHandles, BOOL bWaitAll,DWORD dwMilliseconds);

hHandle is the single object for which the process waits.

lpHandle is the array of handles for which the process waits.

nCount is the number of objects in an array. Should not exceed MAXIMUM WAIT OBJECTS

dwMilliseconds is the timeout period for wait . 0 for no wait and INFINITE FOR indefinite wait.

bWaitAll describes if it's necessary to wait for all the objects to get free.

The possible return values are

WAIT_OBJECT_0

WAIT_OBJECT_0+n

WAIT_TIMEDOUT

WAIT_FAILED

WAIT_ABANDONED_0

Topic No - 97 (Environment Block)

Content:

Environment Block string

-An Environment Block is associated with each process.

-The EB contains string regarding the environment of the process of the form Name = Value

-Each string is NULL-terminated

-PATH is an example of an environment string.

-Environment variables can get and set using these API's

DWORD GetEnvironmentVatiable (LPCTSTR lpNAME,LpTSTR lpBuffer, DWORD nSize);

And

BOOL SetEnviromentVariable(LPCTSTR ipName, LPCTSTR lpValue);

To share the parent environment with the child process set lpEnvironment to NULL in the call to CreateProcess()

Any process can modify the environment variables and make new ones

lpName is the variable name. On setting the string the value is modified if the variable already exists. If it does not exist

then a new variable is created and assigned the value. IF the value is NULL then the variable is deleted.

"=" si reserved and cannot be used in the variable name.

GetEnvironmentVariable() return the length of variable string or 0 in case of failure.

lf LpValue is not as long as the value specified by the count then the actual length of the string is returned.

The access rights given in Create Process () are usually

Process_All_Access. But there are several options

PROCESS_QUERY_INFORMATION

CREATE_PROCESS

PROCESS_TERMINATE

PROCESS_SET_INFORMATION

DUPLICATE_HANDLE

CREATE_HANDLE

it can be useful to limit some processes right like giving PROCESS_TERMINATE right to parent process only.

Topic No - 98 (A Pattern Searching Example)

Content:

A pattern searching example

This example uses the power of windows multitasking to search a specific pattern among numerous files.

The process take the specific pattern along with filenames through command line.

The standard output file is specified as inheritable in new process start-up info structure.

Wait functions are used for synchronizations

As soon as the search end the results are displayed one at a time.

Wait functions are limited for 64 handles so program works accordingly.

The program uses the exit code to identify whether the process has detected a match or not.

A Pattern Searching Example

The Example

```
#include "Everything.h"

int _tmain (int argc, LPTSTR argv[])

/*      Create a separate process to search each file on the

        command line. Each process is given a temporary file,

        in the current directory, to receive the results. */

{

        HANDLE hTempFile;

        SECURITY_ATTRIBUTES stdOutSA = /* SA for inheritable handle. */
```

```
                        {sizeof (SECURITY_ATTRIBUTES), NULL, TRUE};
TCHAR commandLine[MAX_PATH + 100];
        STARTUPINFO startUpSearch, startUp;
        PROCESS_INFORMATION processInfo;
        DWORD exitCode, dwCreationFlags = 0;
        int iProc;
        HANDLE *hProc;  /* Pointer to an array of proc handles. */
        typedef struct {TCHAR tempFile[MAX_PATH];} PROCFILE;
        PROCFILE *procFile; /* Pointer to array of temp file names. */
#ifdef UNICODE
   dwCreationFlags = CREATE_UNICODE_ENVIRONMENT;
#endif
if (argc < 3)
                ReportError (_T ("Usage: grepMP pattern files."), 1, FALSE);
        /* Startup info for each child search process as well as
                the child process that will display the results. */
        GetStartupInfo (&startUpSearch);
        GetStartupInfo (&startUp);
        /* Allocate storage for an array of process data structures,
                each containing a process handle and a temporary file name. */
procFile = malloc ((argc - 2) * sizeof (PROCFILE));
        hProc = malloc ((argc - 2) * sizeof (HANDLE));
        /* Create a separate "grep" process for each file on the
                command line. Each process also gets a temporary file
```

```
                name for the results; the handle is communicated through
                the STARTUPINFO structure. argv[1] is the search pattern. */
        for (iProc = 0; iProc < argc - 2; iProc++) {
                /* Create a command line of the form: grep argv[1] argv[iProc + 2] */
/* Allow spaces in the file names. */
                _stprintf (commandLine, _T ("grep \"%s\" \"%s\""), argv[1], argv[iProc + 2]);
                /* Create the temp file name for std output. */
                if (GetTempFileName (_T ("."), _T ("gtm"), 0, procFile[iProc].tempFile) == 0)
                        ReportError (_T ("Temp file failure."), 2, TRUE);
                /* Set the std output for the search process. */
hTempFile = /* This handle is inheritable */
                        CreateFile (procFile[iProc].tempFile,
                                /** GENERIC_READ | Read access not required **/ GENERIC_WRITE,
                                FILE_SHARE_READ | FILE_SHARE_WRITE, &stdOutSA,
                                CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
                if (hTempFile == INVALID_HANDLE_VALUE)
                        ReportError (_T ("Failure opening temp file."), 3, TRUE);
                /* Specify that the new process takes its std output
                        from the temporary file's handles. */
startUpSearch.dwFlags = STARTF_USESTDHANDLES;
                startUpSearch.hStdOutput = hTempFile;
                startUpSearch.hStdError = hTempFile;
                startUpSearch.hStdInput = GetStdHandle (STD_INPUT_HANDLE);
                /* Create a process to execute the command line. */
```

```
                if (!CreateProcess (NULL, commandLine, NULL, NULL,
                                TRUE, dwCreationFlags, NULL, NULL, &startUpSearch, &processInfo))
                        ReportError (_T ("ProcCreate failed."), 4, TRUE);
                /* Close unwanted handles */
                CloseHandle (hTempFile); CloseHandle (processInfo.hThread);
/* Save the process handle. */
                hProc[iProc] = processInfo.hProcess;
        }
        /* Processes are all running. Wait for them to complete, then output
                the results - in the order of the command line file names. */
        for (iProc = 0; iProc < argc-2; iProc += MAXIMUM_WAIT_OBJECTS)
                WaitForMultipleObjects (min(MAXIMUM_WAIT_OBJECTS, argc - 2 - iProc),
                                &hProc[iProc], TRUE, INFINITE);
/* Result files sent to std output using "cat".
                Delete each temporary file upon completion. */
        for (iProc = 0; iProc < argc - 2; iProc++) {
                if (GetExitCodeProcess (hProc[iProc], &exitCode) && exitCode == 0) {
                        /* Pattern was detected - List results. */
                        /* List the file name if there is more than one file to search */
                        if (argc > 3) _tprintf (_T("%s:\n"), argv[iProc+2]);
                        _stprintf (commandLine, _T ("cat \"%s\""), procFile[iProc].tempFile);
if (!CreateProcess (NULL, commandLine, NULL, NULL,
                                TRUE, dwCreationFlags, NULL, NULL, &startUp, &processInfo))
                                ReportError (_T ("Failure executing cat."), 0, TRUE);
```

```
                else {
                        WaitForSingleObject (processInfo.hProcess, INFINITE);
                        CloseHandle (processInfo.hProcess);
                        CloseHandle (processInfo.hThread);
                }
        }
CloseHandle (hProc[iProc]);
        if (!DeleteFile (procFile[iProc].tempFile))
                ReportError (_T ("Cannot delete temp file."), 6, TRUE);
    }
    free (procFile); free (hProc);
    return 0;
}
```

Topic No - 99 (Working in Multiprocessor Environment)

Content:

Working in Multiprocessor Environment

Multiprocessor Environment

All the processes run concurrently in windows environment.

If the system is uniprocessor the processor time is multiplexed among multiple processes in an interleaved manner

If the system is multiprocessor then windows scheduler can run process threads on separate processors.

There would be substantial performance gain in this case

The performance gain will not be linear because of dependencies among processes (wait and signal)

From programming point of view it is essential to understand this potential of windows so that programs can be designed optimally.

Alternately, constraining the processing to a specific processor is also possible.

This is accomplished by processor affinity mask

Subsequently, it is possible to create independent threads within a process which can be scheduled on separate processor.

Topic No - 100 (Process Execution Times)

Process Execution Times

Process Times

Windows API provides a very simple mechanism for determining the amount of time a process has consumed.

The function used for this purpose is GetProcessTime()

BOOL GetProcessTimes(HANDLE hProcess,

LPFILETIME lpCreationTime,

LPFILETIME lpExitTime,

LPFILETIME lpKernelTime,

LPFILETIME lpUserTime);

Process handle can be of the current process or a terminated one.

Elapsed time can be computed by subtracting creation time from exit time.

FILETIME is a 64-bit value

GetThreadTime() can be used similarly and requires a hread handle.

Topic No - 101 (Example of Process Execution Times)

Content:

Example of Process Execution Times

Example

This example prints the elapsed, real, and user times.

It uses the API GetCommandLine()to get the command line as a single string

It then uses the SkipArg() function the skip past the executable name.

Process Execution Times example

#include "Everything.h"

```
int _tmain (int argc, LPTSTR argv[])

{

        STARTUPINFO startUp;

        PROCESS_INFORMATION procInfo;

        union {             /* Structure required for file time arithmetic. */

                LONGLONG li;

                FILETIME ft;

        } createTime, exitTime, elapsedTime;



FILETIME kernelTime, userTime;

        SYSTEMTIME elTiSys, keTiSys, usTiSys;

        LPTSTR targv, cLine = GetCommandLine ();

        OSVERSIONINFO windowsVersion;

        HANDLE hProc;



        targv = SkipArg(cLine, 1, argc, argv);

        /*  Skip past the first blank-space delimited token on the command line */

        if (argc <= 1 || NULL == targv)

                ReportError (_T("Usage: timep command ..."), 1, FALSE);

/* Determine is this is Windows 2000 or NT.  */
```

```
        windowsVersion.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
        if (!GetVersionEx (&windowsVersion))
                ReportError (_T("Can not get OS Version info. %d"), 2, TRUE);

        if (windowsVersion.dwPlatformId != VER_PLATFORM_WIN32_NT)
                ReportError (_T("This program only runs on Windows NT kernels"), 2, FALSE);

/* Determine is this is Windows 2000 or NT.  */
        windowsVersion.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
        if (!GetVersionEx (&windowsVersion))
                ReportError (_T("Can not get OS Version info. %d"), 2, TRUE);

        if (windowsVersion.dwPlatformId != VER_PLATFORM_WIN32_NT)
                ReportError (_T("This program only runs on Windows NT kernels"), 2, FALSE);
GetStartupInfo (&startUp);

        /* Execute the command line and wait for the process to complete. */
        if (!CreateProcess (NULL, targv, NULL, NULL, TRUE,
                NORMAL_PRIORITY_CLASS, NULL, NULL, &startUp, &procInfo))
                ReportError (_T ("\nError starting process. %d"), 3, TRUE);
        hProc = procInfo.hProcess;
```

```
        if (WaitForSingleObject (hProc, INFINITE) != WAIT_OBJECT_0)

ReportError (_T("Failed waiting for process termination. %d"), 5, TRUE);;

        if (!GetProcessTimes (hProc, &createTime.ft,
                &exitTime.ft, &kernelTime, &userTime))
                        ReportError (_T("Can not get process times. %d"), 6, TRUE);

        elapsedTime.li = exitTime.li - createTime.li;
        FileTimeToSystemTime (&elapsedTime.ft, &elTiSys);
        FileTimeToSystemTime (&kernelTime, &keTiSys);
        FileTimeToSystemTime (&userTime, &usTiSys);
        _tprintf (_T ("Real Time: %02d:%02d:%02d.%03d\n"),

elTiSys.wHour, elTiSys.wMinute, elTiSys.wSecond,
                elTiSys.wMilliseconds);
        _tprintf (_T ("User Time: %02d:%02d:%02d.%03d\n"),
                usTiSys.wHour, usTiSys.wMinute, usTiSys.wSecond,
                usTiSys.wMilliseconds);
        _tprintf (_T ("Sys Time:  %02d:%02d:%02d.%03d\n"),
                keTiSys.wHour, keTiSys.wMinute, keTiSys.wSecond,
```

```
                keTiSys.wMilliseconds);


        CloseHandle (procInfo.hThread); CloseHandle (procInfo.hProcess);

        return 0;

}
```

Topic No - 102 (Generating Console events)

Content:

Example of Process Execution Times

Example

This example prints the elapsed, real and user times.

It uses the API GetCommandLine()to get the command line as a single string

It than uses the SkipArg() function the skip past the executable name.

Process Execution Times example

```
#include "Everything.h"


int _tmain (int argc, LPTSTR argv[])

{
```

```
        STARTUPINFO startUp;
        PROCESS_INFORMATION procInfo;
        union {             /* Structure required for file time arithmetic. */
                LONGLONG li;
                FILETIME ft;
        } createTime, exitTime, elapsedTime;

FILETIME kernelTime, userTime;
        SYSTEMTIME elTiSys, keTiSys, usTiSys;
        LPTSTR targv, cLine = GetCommandLine ();
        OSVERSIONINFO windowsVersion;
        HANDLE hProc;

        targv = SkipArg(cLine, 1, argc, argv);
        /*  Skip past the first blank-space delimited token on the command line */
        if (argc <= 1 || NULL == targv)
                ReportError (_T("Usage: timep command ..."), 1, FALSE);
/* Determine is this is Windows 2000 or NT.  */
        windowsVersion.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
        if (!GetVersionEx (&windowsVersion))
                ReportError (_T("Can not get OS Version info. %d"), 2, TRUE);

        if (windowsVersion.dwPlatformId != VER_PLATFORM_WIN32_NT)
                ReportError (_T("This program only runs on Windows NT kernels"), 2, FALSE);
```

```
/* Determine is this is Windows 2000 or NT.  */
        windowsVersion.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
        if (!GetVersionEx (&windowsVersion))
                ReportError (_T("Can not get OS Version info. %d"), 2, TRUE);

        if (windowsVersion.dwPlatformId != VER_PLATFORM_WIN32_NT)
                ReportError (_T("This program only runs on Windows NT kernels"), 2, FALSE);
GetStartupInfo (&startUp);

        /* Execute the command line and wait for the process to complete. */
        if (!CreateProcess (NULL, targv, NULL, NULL, TRUE,
                NORMAL_PRIORITY_CLASS, NULL, NULL, &startUp, &procInfo))
                ReportError (_T ("\nError starting process. %d"), 3, TRUE);
        hProc = procInfo.hProcess;
        if (WaitForSingleObject (hProc, INFINITE) != WAIT_OBJECT_0)

ReportError (_T("Failed waiting for process termination. %d"), 5, TRUE);;

        if (!GetProcessTimes (hProc, &createTime.ft,
                &exitTime.ft, &kernelTime, &userTime))
                        ReportError (_T("Can not get process times. %d"), 6, TRUE);

        elapsedTime.li = exitTime.li - createTime.li;
```

```
        FileTimeToSystemTime (&elapsedTime.ft, &elTiSys);
        FileTimeToSystemTime (&kernelTime, &keTiSys);
        FileTimeToSystemTime (&userTime, &usTiSys);
        _tprintf (_T ("Real Time: %02d:%02d:%02d.%03d\n"),

elTiSys.wHour, elTiSys.wMinute, elTiSys.wSecond,
                elTiSys.wMilliseconds);
        _tprintf (_T ("User Time: %02d:%02d:%02d.%03d\n"),
                usTiSys.wHour, usTiSys.wMinute, usTiSys.wSecond,
                usTiSys.wMilliseconds);
        _tprintf (_T ("Sys Time:  %02d:%02d:%02d.%03d\n"),
                keTiSys.wHour, keTiSys.wMinute, keTiSys.wSecond,
                keTiSys.wMilliseconds);

        CloseHandle (procInfo.hThread); CloseHandle (procInfo.hProcess);
        return 0;
}
```

Topic No - 103 (Simple job Management Shell)

Content:

Generating Console Events

Usage and Significance

Terminating another process can be problematic as the terminating process does not get a chance to release resources.

SEH does not help either as there is no mechanism that can be used to raise an exception in another process.

Console control event allows sending a message from one process to another.

Usually, a handler is set up in a process to catch such signals. Subsequently, the handler generates an exception.

It is hence possible for a process to generate an event in another process.

CreateProcess() can be used with the flag CREATE_NEW_PROCESS_GROUP.

This creates a process group in which the created process is the root and all the subsequently created processes by the parent are in the new group.

One process in a group can generate a CTRL_C_EVENT or CTRL_BREAK_EVENT in a group identified by the root process ID.

The target process should have the same console as the process generating the event.

More specifically, the calling process cannot have its own console using CREATE_NEW_CONSOLE or DETACHED_PROCESS flags.

BOOL GenerateConsoleCtrlEvent(DWORD dwCtrlEvent,

DWORD dwProcessGroupId);

dwCtrlEvent can be CTRL_C_EVENT or CTRL_BREAK_EVENT

dwProcessGroupId is the id of the process group

Topic No - 104 (Get a job Number)

Content:

Simple Job Management Shell

Job Management Shell

This job shell will allow three commands to run

jobbg

jobs

kill

The job shell parses the command line and then calls the respective function for the given command.

The shell uses a user-specific file keeping track of process ID and other related information

Several shells can run concurrently and use this shared file.

Also, concurrency issues can be encountered as several shells can try to use the same file.

File locking is used to make the file access mutually exclusive.

```
/*      Chapter 6 */
/* JobObjectShell.c One program combining three
job management commands:
        Jobbg   - Run a job in the background
        jobs    - List all background jobs
        kill    - Terminate a specified job of the job family
                        There is an option to generate a console
                        control signal.
        This implementation enhances JobShell with a time limit on each process.
                There is a time limit on each process, in seconds, in argv[1] (if present)
                0 or omitted means no process time limit
*/
#include "Everything.h"
#include "JobManagement.h"
```

```
#define MILLION 1000000
static int Jobbg (int, LPTSTR *, LPTSTR);
static int Jobs  (int, LPTSTR *, LPTSTR);
static int Kill  (int, LPTSTR *, LPTSTR);
HANDLE hJobObject = NULL;

JOBOBJECT_BASIC_LIMIT_INFORMATION basicLimits = {0, 0, JOB_OBJECT_LIMIT_PROCESS_TIME};

int _tmain (int argc, LPTSTR argv[])
{
LARGE_INTEGER processTimeLimit;
        BOOL exitFlag = FALSE;
        TCHAR command[MAX_COMMAND_LINE], *pc;
        DWORD i, localArgc;
        TCHAR argstr[MAX_ARG][MAX_COMMAND_LINE];
        LPTSTR pArgs[MAX_ARG];

        /* NT Only - due to file locking */
        if (!WindowsVersionOK (3, 1))
                ReportError (_T("This program requires Windows NT 3.1 or greater"), 1, FALSE);
        hJobObject = NULL;
        processTimeLimit.QuadPart = 0;
        if (argc >= 2) processTimeLimit.QuadPart = _ttoi(argv[1]);
        basicLimits.PerProcessUserTimeLimit.QuadPart = processTimeLimit.QuadPart *
MILLION;
```

```
	hJobObject = CreateJobObject(NULL, NULL);
	if (NULL == hJobObject)
		ReportError(_T("Error creating job object."), 1, TRUE);
	if (!SetInformationJobObject(hJobObject, JobObjectBasicLimitInformation, &basicLimits,
sizeof(JOBOBJECT_BASIC_LIMIT_INFORMATION)))
		ReportError(_T("Error setting job object information."), 2, TRUE);

	for (i = 0; i < MAX_ARG; i++)
		pArgs[i] = argstr[i];

	_tprintf (_T("Windows Job Mangement with Windows Job Object.\n"));
	while (!exitFlag) {
_tprintf (_T("%s"), _T("JM$"));
		_fgetts (command, MAX_COMMAND_LINE, stdin);
		pc = _tcschr (command, _T('\n'));
		*pc = _T('\0');

		GetArgs (command, &localArgc, pArgs);
		CharLower (argstr[0]);

		if (_tcscmp (argstr[0], _T("jobbg")) == 0) {
			Jobbg (localArgc, pArgs, command);
		}
		else if (_tcscmp (argstr[0], _T("jobs")) == 0) {
			Jobs (localArgc, pArgs, command);
		}
```

```
                else if (_tcscmp (argstr[0], _T("kill")) == 0) {
                        Kill (localArgc, pArgs, command);
}
                else if (_tcscmp (argstr[0], _T("quit")) == 0) {
                        exitFlag = TRUE;
                }
                else _tprintf (_T("Illegal command. Try again.\n"));
        }

        CloseHandle (hJobObject);

        return 0;
}

/* Jobbg: Execute a command line in the background, but
        the job identity in the user's job file, and exit.
        Related commands (jobs, fg, kill, and suspend) can be used to manage the jobs. */
/* jobbg [options] command-line
                -c: Give the new process a console.
                -d: The new process is detached, with no console.
        These two options are mutually exclusive.
        If neither is set, the background process shares the console with jobbg. */

/* These new features this program illustrates:
                1. Creating detached and with separate consoles.
                2. Maintaining a Jobs/Process list in a shared file.
```

```
                3. Determining a process status from a process ID.*/


/* Standard include files. */
/* - - - - - - - - - - - - - - - - - - - - - - - - */


int Jobbg (int argc, LPTSTR argv[], LPTSTR command)
{
/*      Similar to timep.c to process command line. */
        /*      - - - - - - - - - - - - - - - - - - - - - */
        /*      Execute the command line (targv) and store the job id,
                the process id, and the handle in the jobs file. */

        DWORD fCreate;
        LONG jobNumber;
        BOOL flags[2];

        STARTUPINFO startUp;
        PROCESS_INFORMATION processInfo;
        LPTSTR targv = SkipArg (command, 1, argc, argv);

        GetStartupInfo (&startUp);
/* Determine the options. */
        Options (argc, argv, _T ("cd"), &flags[0], &flags[1], NULL);

                /* Skip over the option field as well, if it exists. */
                /* Simplifying assumptions: There's only one of -d, -c (they are mutually exclusive.
```

```
                Also, commands can't start with -. etc. You may want to fix this. */
        if (argv[1][0] == _T('-'))
                targv = SkipArg (command, 2, argc, argv);

        fCreate = flags[0] ? CREATE_NEW_CONSOLE : flags[1] ? DETACHED_PROCESS : 0;

                /* Create the job/thread suspended.
                        Resume it once the job is entered properly. */
if (!CreateProcess (NULL, targv, NULL, NULL, TRUE,
                        fCreate | CREATE_SUSPENDED | CREATE_NEW_PROCESS_GROUP,
                        NULL, NULL, &startUp, &processInfo)) {
                ReportError (_T ("Error starting process."), 0, TRUE);
        }
        if (hJobObject != NULL)
        {
                if (!AssignProcessToJobObject(hJobObject, processInfo.hProcess)) {
                        ReportError(_T("Could not add process to job object. The process will be
terminated."), 0, TRUE);
                        TerminateProcess (processInfo.hProcess, 4);
                        CloseHandle (processInfo.hThread);
                        CloseHandle (processInfo.hProcess);
                        return 4;
}
        }

                /* Create a job number and enter the process Id and handle
```

```
                    into the Job "database" maintained by the
                    GetJobNumber function (part of the job management library). */

        jobNumber = GetJobNumber (&processInfo, targv);
        if (jobNumber >= 0)
                ResumeThread (processInfo.hThread);
        else {
                TerminateProcess (processInfo.hProcess, 3);
                CloseHandle (processInfo.hThread);
                CloseHandle (processInfo.hProcess);
                ReportError (_T ("Error: No room in job control list."), 0, FALSE);
                return 5;
}

        CloseHandle (processInfo.hThread);
        CloseHandle (processInfo.hProcess);
        _tprintf (_T (" [%d] %d\n"), jobNumber, processInfo.dwProcessId);
        return 0;
}

/* Jobs: List all running or stopped jobs that have
        been created by this user under job management;
        that is, have been started with the jobbg command.
        Related commands (jobbg and kill) can be used to manage the jobs. */
/* These new features this program illustrates:
        1. Determining process status.
```

```
	2. Maintaining a Jobs/Process list in a shared file.

	3. Obtaining job object information

*/



int Jobs (int argc, LPTSTR argv[], LPTSTR command)

{

	JOBOBJECT_BASIC_ACCOUNTING_INFORMATION basic info;



	if (!DisplayJobs ()) return 1;

	/* Dispaly the job information */

	if (!QueryInformationJobObject(hJobObject, JobObjectBasicAccountingInformation, &basicInfo,
sizeof(JOBOBJECT_BASIC_ACCOUNTING_INFORMATION), NULL)) {

		ReportError(_T("Failed QueryInformationJobObject"), 0, TRUE);

		return 0;

	}

	_tprintf (_T("Total Processes: %d, Active: %d, Terminated: %d.\n"),

		basicInfo.TotalProcesses, basicInfo.ActiveProcesses,

basicInfo.TotalTerminatedProcesses);

	_tprintf (_T("User time all processes: %d.%03d\n"),

		basicInfo.TotalUserTime.QuadPart / MILLION, (basicInfo.TotalUserTime.QuadPart %
MILLION) / 10000);



	return 0;

}
```

```
/* kill [options] jobNumber
        Terminate the process associated with the specified job number. */
/* This new features this program illustrates:
        1. Using TerminateProcess
        2. Console control events */
/* Options:
        -b  Generate a Ctrl-Break
        -c  Generate a Ctrl-C
Otherwise, terminate the process. */

/* The Job Management data structures, error codes,
        constants, and so on are in the following file. */

int Kill (int argc, LPTSTR argv[], LPTSTR command)
{
        DWORD processId, jobNumber, iJobNo;
        HANDLE hProcess;
        BOOL cntrlC, cntrlB, killed;

        iJobNo = Options (argc, argv, _T ("bc"), &cntrlB, &cntrlC, NULL);

        /* Find the process ID associated with this job. */

jobNumber = _ttoi (argv[iJobNo]);
        processId = FindProcessId (jobNumber);
        if (processId == 0)          {
```

```
                ReportError (_T ("Job number not found.\n"), 0, FALSE);
                return 1;
        }
        hProcess = OpenProcess (PROCESS_TERMINATE, FALSE, processId);
        if (hProcess == NULL) {
                ReportError (_T ("Process already terminated.\n"), 0, FALSE);
                return 2;
        }



if (cntrlB)
                killed = GenerateConsoleCtrlEvent (CTRL_BREAK_EVENT, 0);
        else if (cntrlC)
                killed = GenerateConsoleCtrlEvent (CTRL_C_EVENT, 0);
        else
                killed = TerminateProcess (hProcess, JM_EXIT_CODE);
        if (!killed) {
                ReportError (_T ("Process termination failed."), 0, TRUE);
                return 3;
        }
                WaitForSingleObject (hProcess, 5000);
        CloseHandle (hProcess);


        _tprintf (_T ("Job [%d] terminated or timed out\n"), jobNumber);
        return 0;
}
```

Topic No - 105 (Listing Background jobs)

Content: Content: Content: Job Listing

The job management function used for the purpose is DisplayJobs()

The function opens the file.

Looks up into the file and acquires the status of the processes listed in the file.

Also displays the status of the processes listed and other information.

BOOL DisplayJobs (void)

```
/* Scan the job database file, reporting on the status of all jobs.
        In the process remove all jobs that no longer exist in the system. */
{
        HANDLE hJobData, hProcess;
        JM_JOB jobRecord;
        DWORD jobNumber = 0, nXfer, exitCode;
        TCHAR jobMgtFileName[MAX_PATH];
        OVERLAPPED regionStart;

        if ( !GetJobMgtFileName (jobMgtFileName) )
                return FALSE;
        hJobData = CreateFile (jobMgtFileName, GENERIC_READ | GENERIC_WRITE,
                FILE_SHARE_READ | FILE_SHARE_WRITE,
                NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

```
if (hJobData == INVALID_HANDLE_VALUE)
                return FALSE;

        /* Read records and report on each job. */
        /*  First, create an exclusive lock on the entire
                file as entries will be changed. */
        regionStart.Offset = 0;
        regionStart.OffsetHigh = 0;
        regionStart.hEvent = (HANDLE)0;
        LockFileEx (hJobData, LOCKFILE_EXCLUSIVE_LOCK, 0, 0, 0, ®ionStart);

        __try {
        while (ReadFile (hJobData, &jobRecord, SJM_JOB, &nXfer, NULL) && (nXfer > 0)) {
                jobNumber++; /* jobNumber won't exceed MAX_JOBS_ALLOWED as the file is

not allowed to grow that long */
                hProcess = NULL;
                if (jobRecord.ProcessId == 0) continue;
                                        /* Obtain process status. */
                hProcess = OpenProcess (PROCESS_ALL_ACCESS, FALSE, jobRecord.ProcessId);
                if (hProcess != NULL) {
                        GetExitCodeProcess (hProcess, &exitCode);
                        CloseHandle (hProcess);
                }
                                        /* Show the job status. */
```

```
                _tprintf (_T (" [%d] "), jobNumber);
                if (NULL == hProcess)
                        _tprintf (_T (" Done"));
                else if (exitCode != STILL_ACTIVE)
                        _tprintf (_T ("+ Done"));
else _tprintf (_T ("      "));
                                        /* Show the command. */
                _tprintf (_T (" %s\n"), jobRecord.CommandLine);

                /* Remove processes that are no longer in the system. */

                if (NULL == hProcess) { /* Back up one record. */
                        /* SetFilePoiner is more convenient here than SetFilePointerEx since the move is
a short shift from the current position */
                        SetFilePointer (hJobData, -(LONG)nXfer, NULL, FILE_CURRENT);
                                        /* Set the job number to 0. */
                        jobRecord.ProcessId = 0;
                        if (!WriteFile (hJobData, &jobRecord, SJM_JOB, &nXfer, NULL))
                                ReportError (_T ("Rewrite error."), 1, TRUE);

}
        } /* End of while. */
        } /* End of try.   */

        __finally {                     /* Release the lock on the file. */
                UnlockFileEx (hJobData, 0, 0, 0, ®ionStart);
```

```
                if (NULL != hProcess) CloseHandle (hProcess);
        }

        CloseHandle (hJobData);
        return TRUE;
}
```

Topic No - 106 (Finding a Process Id)

Content:

Finding a Process Id

The job management function used for the purpose is FindProcessId()

It obtains the process id of the given job number

It simply looks up into the file based on job number and reads the record at the specific location.

DWORD FindProcessId (DWORD jobNumber)

```
/* Obtain the process ID of the specified job number. */
{
        HANDLE hJobData;
```

```
        JM_JOB jobRecord;
        DWORD nXfer, fileSizeLow;
        TCHAR jobMgtFileName[MAX_PATH+1];
        OVERLAPPED regionStart;
        LARGE_INTEGER fileSize;

        if ( !GetJobMgtFileName (jobMgtFileName) ) return 0;
        hJobData = CreateFile (jobMgtFileName, GENERIC_READ,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
if (hJobData == INVALID_HANDLE_VALUE) return 0;
        /* Position to the correct record, but not past the end of file */
        /* As a variation, use GetFileSize to demonstrate its operation. */
        if (!GetFileSizeEx (hJobData, &fileSize) ||
                (fileSize.HighPart != 0 || SJM_JOB * (jobNumber - 1) > fileSize.LowPart
                 || fileSize.LowPart > SJM_JOB * MAX_JOBS_ALLOWED))
                        return 0;
        fileSizeLow = fileSize.LowPart;
        /* SetFilePoiner is more convenient here than SetFilePointerEx since the the file is known to be
"short" ( < 4 GB). */
        SetFilePointer (hJobData, SJM_JOB * (jobNumber - 1), NULL, FILE_BEGIN);

        /* Get a shared lock on the record. */
        regionStart.Offset = SJM_JOB * (jobNumber - 1);
        regionStart.OffsetHigh = 0; /* Assume a "short" file. */
        regionStart.hEvent = (HANDLE)0;
```

```
        LockFileEx (hJobData, 0, 0, SJM_JOB, 0, &regionStart);

        if (!ReadFile (hJobData, &jobRecord, SJM_JOB, &nXfer, NULL))
                ReportError (_T ("JobData file error"), 0, TRUE);

        UnlockFileEx (hJobData, 0, SJM_JOB, 0, &regionStart);
        CloseHandle (hJobData);

        return jobRecord.ProcessId;
}
```

Topic No - 107 (Job Objects)

Content: Job Objects
Windows provides the provision of grouping a number of processes into a job object.
Resource limits can be specified for each job object.
This also helps maintain accounting information
Creating Job Objects
Firstly a job object is created using CreateJobObject(). It uses a name and security attributes.
Also, OpenJobObject() is used with named object and CloseHandle() is used to close the object.
Once the job object is created it can be assigned processes using AssignProcesstoJobObject()
Creating Job Objects
Once a process is assigned to a job object it cannot be assigned to another job object.
Child processes are automatically assigned the same job object as the parent unless the CREATE_BREAKAWAY_FROM_JOB flag is used at creation.
Control limits are specified to the processes in job object using:
BOOL SetInformationJobObject(HANDLE hJob,
JOBOBJECTINFOCLASS JobObjectInformationClass,
LPVOID lpJobObjectInformation, DWORD cbJobObjectInformationLength);

hJob is handled to the existing job object

JobObjectInformationClass is the class of the limits you wish to set regarding per process time, working set limit, the limit on active processes, priority, processor affinity, etc.

lpJobObjectInformation is the structure congaing the actual information as it uses a different structure for each class.

Topic No - 108 (Using Job Objects)
Content: Using Job Objects
In this example job objects are used to limit process execution time and obtain user time statistics.
The simple JobShell program discussed previously is modified.
A command line time limit argument is added.

```
/* Chapter 6 */
/* JobObjectShell.c One program combining three
job management commands:
Jobbg - Run a job in the background
jobs - List all background jobs
kill - Terminate a specified job of job family
  There is an option to generate a console
  control signal.
This implemenation enhances JobShell with a time limit on each process.
There is a time limit on each process, in seconds, in argv[1] (if present)
0 or omitted means no process time limit
*/

#include "Everything.h"
#include "JobManagement.h"

#define MILLION 1000000
static int Jobbg (int, LPTSTR *, LPTSTR);
static int Jobs  (int, LPTSTR *, LPTSTR);
static int Kill  (int, LPTSTR *, LPTSTR);
HANDLE hJobObject = NULL;

JOBOBJECT_BASIC_LIMIT_INFORMATION basicLimits = {0, 0, JOB_OBJECT_LIMIT_PROCESS_TIME};

int _tmain (int argc, LPTSTR argv[])
{
LARGE_INTEGER processTimeLimit;
BOOL exitFlag = FALSE;
```

```
TCHAR command[MAX_COMMAND_LINE], *pc;
DWORD i, localArgc;
TCHAR argstr[MAX_ARG][MAX_COMMAND_LINE];
LPTSTR pArgs[MAX_ARG];
/* NT Only - due to file locking */
if (!WindowsVersionOK (3, 1))
ReportError (_T("This program requires Windows NT 3.1 or greater"), 1,
FALSE);
hJobObject = NULL;
processTimeLimit.QuadPart = 0;
if (argc >= 2) processTimeLimit.QuadPart = _ttoi(argv[1]);
basicLimits.PerProcessUserTimeLimit.QuadPart = processTimeLimit.QuadPart * MILLION;

hJobObject = CreateJobObject(NULL, NULL);
if (NULL == hJobObject)
ReportError(_T("Error creating job object."), 1, TRUE);
if (!SetInformationJobObject(hJobObject, JobObjectBasicLimitInformation, &basicLimits,
sizeof(JOBOBJECT_BASIC_LIMIT_INFORMATION)))
ReportError(_T("Error setting job object information."), 2, TRUE);

for (i = 0; i < MAX_ARG; i++)
pArgs[i] = argstr[i];

_tprintf (_T("Windows Job Mangement with Windows Job Object.\n"));
while (!exitFlag) {
_tprintf (_T("%s"), _T("JM$"));
_fgetts (command, MAX_COMMAND_LINE, stdin);
pc = _tcschr (command, _T('\n'));
*pc = _T('\0');

GetArgs (command, &localArgc, pArgs);
CharLower (argstr[0]);

if (_tcscmp (argstr[0], _T("jobbg")) == 0) {
Jobbg (localArgc, pArgs, command);
}
else if (_tcscmp (argstr[0], _T("jobs")) == 0) {
Jobs (localArgc, pArgs, command);
}
else if (_tcscmp (argstr[0], _T("kill")) == 0) {
Kill (localArgc, pArgs, command);
}
```

```
else if (_tcscmp (argstr[0], _T("quit")) == 0) {
exitFlag = TRUE;
}
else _tprintf (_T("Illegal command. Try again.\n"));
}
CloseHandle (hJobObject);
return 0;
}

/* Jobbg: Execute a command line in the background, put
the job identity in the user's job file, and exit.
Related commands (jobs, fg, kill, and suspend) can be used to manage the jobs. */

/* jobbg [options] command-line
-c: Give the new process a console.
-d: The new process is detached, with no console.
These two options are mutually exclusive.
If neither is set, the background process shares the console with jobbg. */

/* These new features this program illustrates:
1. Creating detached and with separate consoles.
2. Maintaining a Jobs/Process list in a shared file.
3. Determining a process status from a process ID.*/
/* Standard includes files. */
/* - - - - - - - - - - - - - - - - - - - - - - - - */

int Jobbg (int argc, LPTSTR argv[], LPTSTR command)
{
/* Similar to timep.c to process command line. */
/* - - - - - - - - - - - - - - - - - - - - - - */
/* Execute the command line (targv) and store the job id,
the process id, and the handle in the jobs file. */
DWORD fCreate;
LONG jobNumber;
BOOL flags[2];
STARTUPINFO startUp;
PROCESS_INFORMATION processInfo;
LPTSTR targv = SkipArg (command, 1, argc, argv);
GetStartupInfo (&startUp);

/* Determine the options. */
Options (argc, argv, _T ("cd"), &flags[0], &flags[1], NULL);
```

```
/* Skip over the option field as well, if it exists. */
/* Simplifying assumptions: There's only one of -d, -c (they are mutually exclusive.
  Also, commands can't start with -. etc. You may want to fix this. */
if (argv[1][0] == _T('-'))
targv = SkipArg (command, 2, argc, argv);

fCreate = flags[0] ? CREATE_NEW_CONSOLE : flags[1] ? DETACHED_PROCESS : 0;
/* Create the job/thread suspended.
Resume it once the job is entered properly. */
if (!CreateProcess (NULL, targv, NULL, NULL, TRUE,
fCreate | CREATE_SUSPENDED | CREATE_NEW_PROCESS_GROUP,
NULL, NULL, &startUp, &processInfo)) {
ReportError (_T ("Error starting process."), 0, TRUE);
}
if (hJobObject != NULL)
{
if (!AssignProcessToJobObject(hJobObject, processInfo.hProcess)) {
ReportError(_T("Could not add process to job object. The process will be terminated."), 0, TRUE);
TerminateProcess (processInfo.hProcess, 4);
CloseHandle (processInfo.hThread);
CloseHandle (processInfo.hProcess);
return 4;
}
}

/* Create a job number and enter the process Id and handle
into the Job "database" maintained by the
GetJobNumber function (part of the job management library). */
jobNumber = GetJobNumber (&processInfo, targv);
if (jobNumber >= 0)
ResumeThread (processInfo.hThread);
else {
TerminateProcess (processInfo.hProcess, 3);
CloseHandle (processInfo.hThread);
CloseHandle (processInfo.hProcess);
ReportError (_T ("Error: No room in job control list."), 0, FALSE);
return 5;
}
CloseHandle (processInfo.hThread);
CloseHandle (processInfo.hProcess);
_tprintf (_T (" [%d] %d\n"), jobNumber, processInfo.dwProcessId);
```

```
return 0;
}
/* Jobs: List all running or stopped jobs that have
been created by this user under job management;
that is, have been started with the jobbg command.
Related commands (jobbg and kill) can be used to manage the jobs. */
/* These new features this program illustrates:
1. Determining process status.
2. Maintaining a Jobs/Process list in a shared file.
3. Obtaining job object information
 */
int Jobs (int argc, LPTSTR argv[], LPTSTR command)
{
JOBOBJECT_BASIC_ACCOUNTING_INFORMATION basicInfo;
if (!DisplayJobs ()) return 1;
/* Dispaly the job information */
if (!QueryInformationJobObject(hJobObject, JobObjectBasicAccountingInformation, &basicInfo,
sizeof(JOBOBJECT_BASIC_ACCOUNTING_INFORMATION), NULL)) {
ReportError(_T("Failed QueryInformationJobObject"), 0, TRUE);
return 0;
}
_tprintf (_T("Total Processes: %d, Active: %d, Terminated: %d.\n"),
basicInfo.TotalProcesses, basicInfo.ActiveProcesses, basicInfo.TotalTerminatedProcesses);
_tprintf (_T("User time all processes: %d.%03d\n"),
basicInfo.TotalUserTime.QuadPart / MILLION, (basicInfo.TotalUserTime.QuadPart % MILLION) / 10000);

return 0;
}
/* kill [options] jobNumber
Terminate the process associated with the specified job number. */
/* This new features this program illustrates:
1. Using TerminateProcess
2. Console control events */
/* Options:
-b  Generate a Ctrl-Break
-c  Generate a Ctrl-C
Otherwise, terminate the process. */

/* The Job Management data structures, error codes,
constants, and so on are in the following file. */

int Kill (int argc, LPTSTR argv[], LPTSTR command)
```

```
{
DWORD processId, jobNumber, iJobNo;
HANDLE hProcess;
BOOL cntrlC, cntrlB, killed;

iJobNo = Options (argc, argv, _T ("bc"), &cntrlB, &cntrlC, NULL);
/* Find the process ID associated with this job. */

jobNumber = _ttoi (argv[iJobNo]);
processId = FindProcessId (jobNumber);
if (processId == 0) {
ReportError (_T ("Job number not found.\n"), 0, FALSE);
return 1;
}
hProcess = OpenProcess (PROCESS_TERMINATE, FALSE, processId);
if (hProcess == NULL) {
ReportError (_T ("Process already terminated.\n"), 0, FALSE);
return 2;
}
if (cntrlB)
killed = GenerateConsoleCtrlEvent (CTRL_BREAK_EVENT, 0);
else if (cntrlC)
killed = GenerateConsoleCtrlEvent (CTRL_C_EVENT, 0);
else
killed = TerminateProcess (hProcess, JM_EXIT_CODE);
if (!killed) {
ReportError (_T ("Process termination failed."), 0, TRUE);
return 3;
}
WaitForSingleObject (hProcess, 5000);
CloseHandle (hProcess);
_tprintf (_T ("Job [%d] terminated or timed out\n"), jobNumber);
return 0;
}
```

Topic No - 109 (Thread Overview)

Content: Threads overview

Thread is an independent unit of execution within a process.

In a multi-threaded system, multiple threads may exist within a process.

Organizing and coordinating these threads is a challenge.

Some programming problems are greatly simplified by the use of threads.

The conventional scenario uses single-threaded processes being run concurrently

This scenario has varying disadvantages

Switching among processes is time-consuming and expensive. Threads can easily allow concurrent processing of the same function hence reducing overheads

Usually, independent processes are not tightly coupled, and hence sharing resources is difficult.

Synchronization and mutual exclusion in single-threaded processes halt computations.

Threads can perform the asynchronously overlapped functions with less programming effort.

The use of multithreading can benefit more with a multiprocessor environment using efficient scheduling of threads.

Topic No - 110 (Thread Issues)

Content: Issues with Threads

Threads share resources within a process. One thread may inadvertently another thread's data.

In some specific cases, concurrency can greatly degrade performance rather than improve.

In some simple single-threaded solutions using multithreading greatly complicates matters and even results in poor performance.

Topic No - 111 (Thread Basics)

Content: Threads Basics

Threads use the space assigned to a process.

In a multi threaded system, multiple threads might be associated with a process

Thread within a process share data and code.

However, individual threads may have their unique data storage in addition to the shared data.

This data is not altogether exclusive.

Programmer must assure that a thread only uses its own data and does not interfere with shared data.

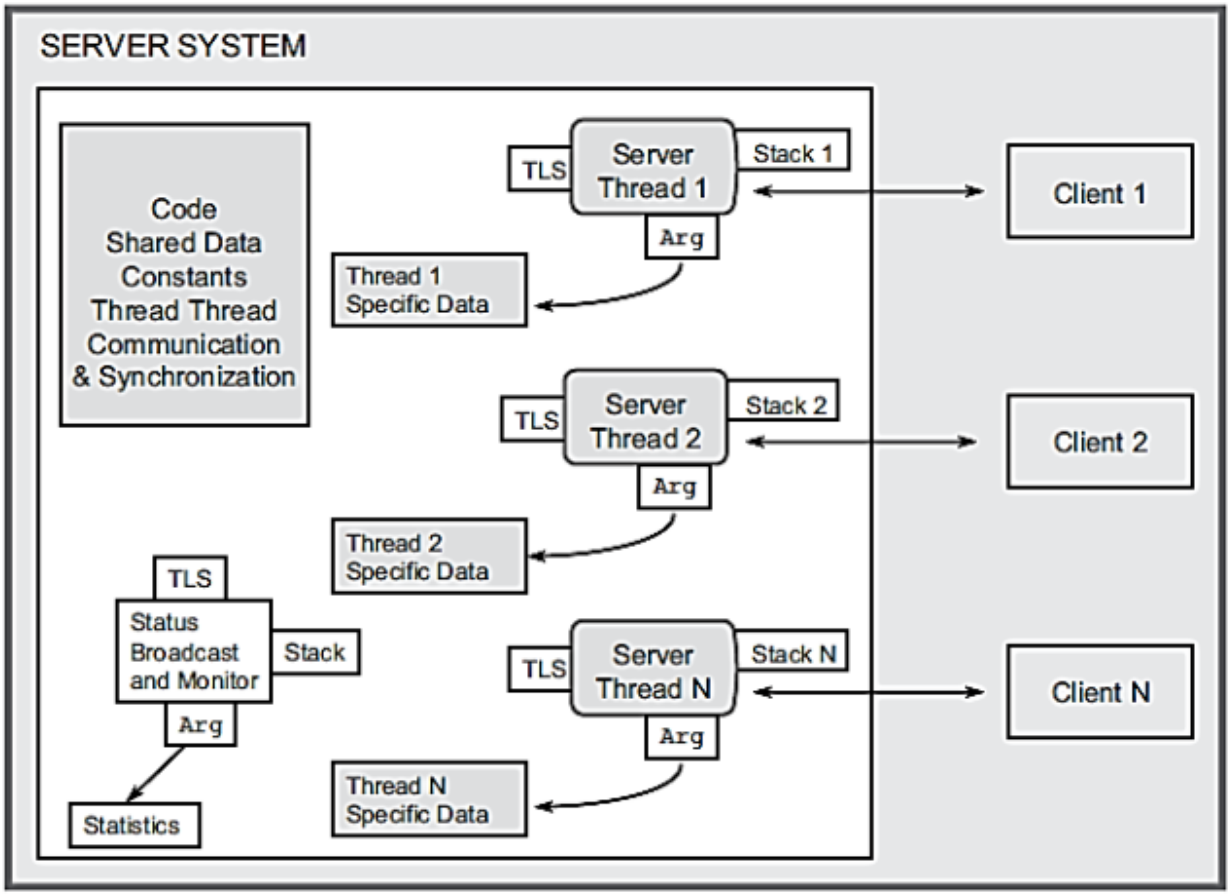
Moreover, each thread has its own stack for function calls.

The calling process usually passes an argument to the thread. These arguments are stored in the thread's stack

Each thread can allocate its own Thread Local Storage(TLS) and set clear values.

TLS is a small pointer array that is only accessible to the thread.

This also assures that a thread will not modify data of any other thread's TLS.



Topic No - 112 (Thread Management)

Content: Thread Management

Like every other resource threads are also treated as objects

The API used to create a thread is CreateThread()

Threads can also be treated as parent and child. Although the OS is unaware of that.

CreateThread() has many unique requirements

The CreateThread() requires the starting address of the thread within the calling process.

It also requires stack space size for the thread which comes from process address space.

Default memory space IMB

One page is committed in start, this grows with requirement.

Requires a pointer to thread argument. This can be any structure.

HANDLE CreateThread(LPSECURITY_ATTRIBUTES lpsa, SIZE_T dwStackSize, LPTHREAD_START_ROUTINE lpStartAddress, LPVOID lpParameter, DWORD dwCreationFlags, LPDWORD lpThreadId);

lpsa is security attributes structure used previously.

dwStackSize is the stack size in bytes.

lpStartAddress is the starting address of the thread function within the calling process of the form:

DWORD WINAPI ThreadFunc(LPVOID)

The function returns a DWORD value which is usually an exit code and accepts a single pointer argument.

lpThreadParam is the pointer passed to the thread and is usually interpreted as a pointer to a structure containing arguments

dwCreationFlags if 0 would been that thread would start readily. If its CREATE_SUSPENDED then the thread will be suspended requiring the use of ResumeThread() to start execution.

lpThreadId is a pointer to a DWORD that will receive the thread identifier. If its kept NULL then no thread identifier will be returned,

Topic No - 113 (Exiting Thread)

Content: Exiting Thread

All threads in a process can exit using a ExitThread() function.

An alternate is that the thread function returns with the exit code.

When a thread exits the thread stack is deallocated and the handle reffering to the thread are invalidated.

If the thread is linked to some DLL then the DllMain() function is invoked with te reason DLL_THREAD_DETACH.

VOID ExitThread(DWORD dwExitCode);

When all the threads exit the process terminates.

One thread can terminate another thread using TerminateThread().

In this case Thread resources will not be deallocated, completion handlers do not execute, no notification is sent to attached DLLs.

Because of these reasons use of TerminateThread() is strongly discouraged.

A thread object will continue to exist even its execution has ended until the last reference to the thread handle is destroyed with CloseHandle().

Any other running thread can retrieve the exit code.

BOOL GetExitCodeThread(HANDLE hThread, LPDWORD lpdwExitCode);

lpdwExitCode contains a pointer to exit code.

The thread is still running the exit code will be STILL_ACTIVE

Topic No - 114 (Thread Identity)

Content: Thread Identity

Thread Ids and handles can be obtained using functions quite similar to one used with processes.

GetCurrentThread()

Non-inheritable pseudohandle to the calling thread.

GetCurrentThreadId()

Obtains thread Id rather than the handle.

GetThreadId()

Obtains thread Id from thread handle

OpenThread()

Creates Thread handle from the thread Id

Content Development

Topic No - 115

More on Thread Management

The functions of thread management that are discussed above are enough to program any useful threading application. However, there are some more functions introduced in the later versions of Windows i.e. Windows XP and Windows 2003 to write a useful and robust program. These functions are described below.

GetProcessIdOfThread()

This function was not available in the earlier version of windows and requires Windows 2003 or later versions. When a thread is specified, this function tells us, to which process a specified thread is linked while returning the id of that process. This function is useful for, mapping thread and process and program that manages or interacts with threads in another process

GetThreadIOPendingFlag()

This function determines whether the thread, specified by its handle, has any outstanding I/O requests. For example, the thread might be blocked for some IO operations. The result is the status at the time that the function is executed; the actual status could change at any time if the target thread completes or initiates an operation.

Suspending and Resuming Threads

In some cases, we might require to pause any running thread or resume any paused thread. For these purposes, Windows maintains a suspend count. Every thread has its separate suspend count. A thread will only run if the suspend count is 0 and if it is not, then the thread is paused and execution of that thread will be stopped until the suspend count becomes 0. One thread can increment or decrement the suspend count of another thread using SuspendThread and ResumeThread. Recall that a thread can be created in the suspended state with a count of 1

DWORD ResumeThread (HANDLE hThread)
DWORD SuspendThread (HANDLE hThread)

Both functions, if successful, return the previous suspend count. 0xFFFFFFFF indicates failure.

Topic No - 116

Waiting for Threads

One thread can wait for another thread to complete or terminate in the same way that threads wait for process termination. Threads and actions should be synchronized so that action can be performed after completing execution. The wait function can be used with the thread handle to check whether a particular thread has completed its execution or not. Window treats thread as an object, and there are two different types of wait function in the windows that can be used for threads as well (Since the thread is also an object), i.e. WaitForSingleObject() or WaitForMultipleObjects(). WaitForSingleObject() is used to wait for a single specified object and WaitForMultipleObjects can be used for more than 1 object, those objects are defined in the form of an array.

For WaitForMultpleObjects, there is a defined limit in windows to wait for execution i.e. MAXIMUM_WAIT_OBJECTS(64), Usually, it is 64 objects. If there are more than 64 objects, we can use multiple calls as well i.e. if there are 100 objects, we can create 2 arrays of 64 and 36 objects and call WaitForMultipleObjects() two times.

The wait function waits for the object, indicated by the handle, to become signaled. In the case of threads, ExitThread() and TerminateThread() set the object to the signaled state, releasing all other threads waiting on the object, including threads that might wait in the future after the thread terminates. Once a thread handle is signaled, it never becomes nonsignaled.

Note that multiple threads can wait on the same object. Similarly, the ExitProcess() function sets the process state and the states of all its threads to signaled.

Topic No – 117

C Library in Threads

Let's assume a scenario, where we are using the window threading function with C library functions concurrently, there might arise a problem of thread safety. For example, strtok() is a C library function, used to extract the token from the specified string, this function uses the global memory space for its internal processing, and if we are using the different copies of that string, they all might use global space. The result achieved from this operation might get compromised and unsatisfactory. In such cases, these types of problems are resolved by using C Library threading functions rather than Windows threading functions. For C Library, Windows C provides a thread-safe library named LIBCMT. This library can be used for thread-relevant functions to program a multithreaded program. So far, we were using the CreateThread and ExitThread functions to create and exit the threads, this library provides us with these equivalent functions i.e. _beginthreadex() and endthreadex() respectively. These C library functions are quite simpler but are not diverse as compared to Windows threading functions. i.e. _beingthreadex() is a simpler function but it does not allow users to specify the security attributes. _endthreadex() does not allow return values and does not pass information regarding the status of the thread. If we are using the windows program and using this C Libray thread function the return value must be type cast to HANDLE to process it further, since the original return type of _beginthreadex() is not HANDLE.

Topic No - 118

Multithreaded Pattern Searching

In this example of pattern searching with multithreading, the program is managing concurrent I/O to multiple files, and the main thread, or any other thread, can perform additional processing before waiting for I/O completion. In this way, we can manage several files in a very simple and efficient way. In the previous examples of pattern searching, we used the multitasking approach, but here we will use the multithreading technique, that enables us to write an optimal program.

In Synchronous Input-Output, suppose an example of the keyboard, when the user presses any keyboard button the execution stops until the button is released, but in asynchronous input-output, a Sound card plays the song at the same time other operations are also performed.

When a read operation is performed, this can be performed concurrently, a file can be read by several processes at the same time or we can also read several files at the same time. But the problem arises when several processes attempt to write a single file. For now, we will be limited to read operation only. This program can be provided several files in which a specific pattern is to be searched. For every file, a separate thread will be created which will search the pattern in that file. Once the pattern is found in the file, it will be reported in a temporary file

```
/* grepMT. */
/* Parallel grep-- multiple thread version. */

#include "Everything.h"
typedef struct { /* grep thread's data structure. */
        int argc;
        TCHAR targv[4][MAX_PATH];
} GREP_THREAD_ARG;

typedef GREP_THREAD_ARG * PGR_ARGS;
static DWORD WINAPI ThGrep (PGR_ARGS pArgs);

int _tmain(int argc, LPTSTR argv[])
{
        GREP_THREADED_ARG *gArg;
        HANDLE *tHandle;
        DWORD threadIndex, exitCode;
        TCHAR commandLine[MAX_COMMAND_LINE);
        int iThrd, threadCount;
        STARTUPINFO startUp;
        PROCESS_INFORMATION processInfo;

        GetStartupInfo(&startUp);

        /* Boss Thread: create seperate "grep" thread for each file. */
```

```
tHandle = malloc((argc-2) * sizeof(HANDLE));
gArg = malloc((argc-2) * sizeof (GREP_THREAD_ARG));

for(iThrd=0, iThrd<argc-2; iThrd++)
{
_tcscpy (gArg[iThrd].targv[1], argv[1]);  /* Pattern */
_tcscpy (gArg[iThrd].targv[2], argv[iThrd + 2]);
GetTempFileName /* Temp file name. */
        (".", "Gre", 0, gArg[iThrd].targv[3];
gArg[iThrd].argc = 4;

/* Create a worker thread to execute the command line. */
tHandle[iThrd] = (HANDLE)_beginthreadex (
        NULL, 0, ThGrep, &gArg[iThrd], 0, NULL);
}

/* Redirect std Output for file listing process. */
startUp.dwFlags = STARTF_USESTDHANDLES;
startUp.hStdOutput = GetStdHandle (STD_OUTPUT_HANDLE);

/* Worker threads ar eall running . Wait for them to complete. */
threadCount = argc - 2;
while (threadCount>0) {
        threadIndex =  WaitForMultipleObjects (
                threadCount, tHandle, FALSE, INFINITE);
        iThrd = (int) threadIndex - int (WAIT_OBJECT_0);
        GetExitCodeThread ( tHandle[iThrd], &exitCode);
        CloseHandle ( tHandle[iThrd]);
        if ( exitCode == 0) {       /* Pattern Found */
                if ( argc > 3) {
                        /* Print file name if more than one . */
                        _tprintf ( _T("\n**Search results - file : %s\n"),
                                gArg[iThrd].targv[2];
                        fflush(stdout);
                }

        /* Use the "cat" program to list the result files. */
        _stprintf ( commandLine, _T("%s%s"), _T("cat "),
                gArg[iThrd].targv[3]);
        CreateProcess(NULL, commandLine, NULL, NULL,
                TRUE, 0, NULL, NULL, &startUP, &processInfo);
        waitForSingleObject ( processInfo.hProcess, INFINITE);
        CloseHandle (processInfo.hProcess);
        CloseHandle (processInfo.hThread);
        }

DeleteFile ( gArg[iThrd].targv[3]);
```

```
        /* Adjust thread and file name arrays. */
        tHandle[iThrd] = tHandle[threadCount - 1];
        _tcscpy(gArg[iThrd].targv[3], gArg[threadCount -1 ].targv[3]);
        _tcscpy(gArg[iThrd].targv[2], gArg[threadCount -1 ].targv[2]);
        threadCount--;
        }
}

/* The form of grep thread function code is :
static  DWORD WINAPI ThGrep (PGR_ARGS pArgs)
{
        ....
} */
```

- Structure of thread argument is created with argument count (argc) and thread argument value (targv) which will be passed to thread. A Thread prototype is also created named as ThGrep which will be used to search the specific patterns in the file.
- STARTUPINFO and PROCESS_INFORMATION structures are created for startup and for creating process respectively
- stratUp information is placed in GetStratupInfo function
- Files are specified using the command line, in which patterns are to be searched, and for every file inputted, a separate thread will run.
- Loop will run till argc-2 times, in argc, the first two parameters will be process name and pattern and 3rd parameter is inputted file names, though, this loop will run, till the number of files inputted. In this loop for every file, the name of the file is copied and its temporary file is created
- Further, arguments that are to be passed to the thread are also prepared, 1st argument will store the pattern which is to be searched, 2nd argument will store the name of the file in which the pattern is to be searched and 4th argument will store the count i.e. how many arguments are stored.
- Thread is created using _beginthreadex, whose name is ThGrep and is passed all the arguments that were created (gArg). With this, the handle of every thread will be stored in the form of an array (tHandle)
- Standard output files, standard error files, and flags are set for startup information
- Total number of threads are stored in ThreadCount
- Another loop is run, in which the Wait function is called to wait for multiple objects specified with a number of threads (threadCount) and handles of all threads (tHandle array). When the wait function is completed, an exitcode will be generated which will tell us why the wait function is stopped, and further, for garbage collection, the handle of the thread is also closed with the CloseHandle function.
- When the wait function was in execution, a process is created with help of the CreateProcess function which has been provided, startup information(startUp), and process information (processInfo).
- Another wait function is also called but this time WaitForSingleObject is called which has been provided the handle of the process named hProcess.
- Once the execution of the process is completed, the handle of the process as well as the

thread are closed.

Topic No – 119

Boss worker and other thread models

In the example of multithreaded pattern searching, there was one main thread, which was running other threads. Each file was assigned a separate thread, which was finding the pattern in that file. This model is more like a boss worker model. The boss worker model is one in which there is one boss and many workers. The Boss assigns work to workers and each worker report result back to the boss. There are many other models, which are used to write an efficient and more understandable multithreaded program depending on the scenarios.

The work crew model is one in which the workers cooperate on a single task, each performing a small piece. They might even divide up the work themselves without direction from the boss. Multithreaded programs can employ nearly every management arrangement that humans use to manage concurrent tasks.

The Client-Server model is mostly used worldwide, in which a client requests the server and the server runs a thread for that client. For every client, a separate thread is run at the server end. In this way, the work is done concurrently rather than sequentially. Another major model is the pipeline model, where work moves from one thread to the next

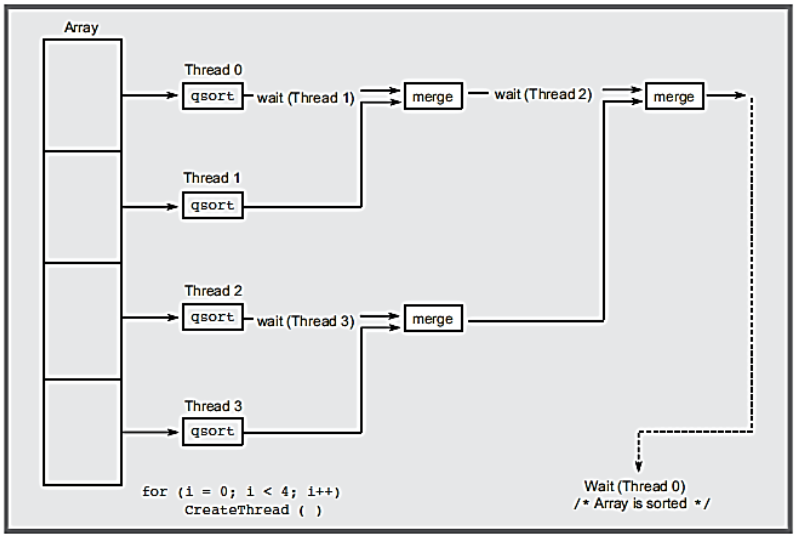
There are many advantages to using these models when designing a multithreaded program, including the following.

- Most models can be used which makes things simpler and expedites programming and debugging efforts.
- Models help you obtain the best performance and avoid common mistakes
- Models naturally correspond to the structure of programming language constructs.
- Maintenance is simplified.
- Troubleshooting is simplified when they are analyzed in terms of a specific model.
- Synchronization and coordination are simplified using well-defined models.

Topic No – 120

MergeSort: Exploiting Multiple Processors

The window is a multiprocessing system i.e. more than 1 process can run simultaneously. This example shows, how multithreading can be used to achieve the optimal performance gain in a multiprocessing system, each process runs a different thread to utilize the optimal resources. The main idea behind this is to subdivide the process into similar tasks so that a separate thread is run for each subtask i.e. A big array is divided into smaller parts, each part is sorted separately with different threads, and all the parts are merged. This will allow parallelism and better performance gain. The strategy implemented in this example is the worker crew model, work is divided into different workers, and all the work is merged in the end. This strategy could also be used, by using multiprocessing instead of multithreading, but the result might not be as efficient as it is with multithreading, because switching overhead is low in multithreading but high in multiprocessing. In the example of MergeSort, each subarray is sorted with qsort() and merged as in the mergesort algorithm. The program code will run most accurately if a number of records are divisible by a number of threads and the number of threads is in the power of two. If the number of processes is equal to the number of threads, this would be the most optimal situation otherwise less optimal. If a list is subdivided into 4 sub-lists and 4 threads are created for these sublists, they all must be created at a suspended state and should only be resumed when all the threads are created. If one thread is completed and the other, which is to be merged, is not created or does not exist, this will occur in a race condition. To avoid the race condition, all the threads should be created with a suspended state and resumed to run all concurrently. The following diagrams explain it further



A Large array is divided into smaller 4 subarrays. For each subarray, a separate thread is created i.e. thread 0, thread 1, thread 2, and thread 3. When thread 0 is sorted it will wait for thread 1 to be sorted, once sorted, they both will be merged. The same happens for thread 2 and thread 3, they are merged when sorted. These 2 merged subarrays are then sorted and merged to form a large sorted array.

Topic No – 121

MergeSort: Exploiting Multiple Processors

```
/* Chapter 7 SortMT. Work crew model
        File sorting with multiple threads and merge sort.
        sortMT [options] nt file. Work crew model.  */
/*  This program is based on sortHP.
        It allocates a block for the sort file, reads the file,
        then creates an "nt" threads (if 0, use the number of
        processors to so sort a piece of the file. It then
        merges the results in pairs. */
/* LIMITATIONS:
        1.      The number of threads must be a power of 2
        2.      The number of 64-byte records must be a multiple
                of the number of threads.
        An exercise asks you to remove these limitations. */
#include "Everything.h"
/* Definitions of the record structure in the sort file. */
#define DATALEN 56  /* Correct length for presdnts.txt and monarchs.txt. */
#define KEYLEN 8
typedef struct _RECORD {
        TCHAR key[KEYLEN];
        TCHAR data[DATALEN];
} RECORD;
#define RECSIZE sizeof (RECORD)
typedef RECORD * LPRECORD;
typedef struct _THREADARG {       /* Thread argument */
        DWORD iTh;          /* Thread number: 0, 1, 3, ... */
        LPRECORD lowRecord;       /* Low Record */
        LPRECORD highRecord;      /* High record */
} THREADARG, *PTHREADARG;
static DWORD WINAPI SortThread (PTHREADARG pThArg);
static int KeyCompare (LPCTSTR, LPCTSTR);
static DWORD nRec; /* Total number of records to be sorted. */
static HANDLE * pThreadHandle;
int _tmain (int argc, LPTSTR argv[])
{

        /* The file is the first argument. Sorting is done in place. */
        /* Sorting is done in memory heaps. */
        HANDLE hFile, mHandle;
        LPRECORD pRecords = NULL;
        DWORD lowRecordNum, nRecTh,numFiles, iTh;
        LARGE_INTEGER fileSize;
        BOOL noPrint;
        int iFF, iNP;
```

```
	PTHREADARG threadArg;
	LPTSTR stringEnd;

	iNP = Options (argc, argv, _T("n"), &noPrint, NULL);
	iFF = iNP + 1;
	numFiles = _ttoi(argv[iNP]);
	if (argc <= iFF)
		ReportError (_T ("Usage: sortMT [options] nTh files."), 1, FALSE);
	/* Open the file and map it */
	hFile = CreateFile (argv[iFF], GENERIC_READ | GENERIC_WRITE,
			0, NULL, OPEN_EXISTING, 0, NULL);
	if (hFile == INVALID_HANDLE_VALUE)
		ReportError (_T ("Failure to open input file."), 2, TRUE);
	// For technical reasons, we need to add bytes to the end.
	/* SetFilePointer is convenient as it's a short addition from the file end */
	if (!SetFilePointer(hFile, 2, 0, FILE_END) || !SetEndOfFile(hFile))
		ReportError (_T ("Failure position extend input file."), 3, TRUE);

	mHandle = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, 0, NULL);
	if (NULL == mHandle)
		ReportError (_T ("Failure to create mapping handle on input file."), 4, TRUE);

	/* Get the file size. */
	if (!GetFileSizeEx (hFile, &fileSize))
		ReportError (_T ("Error getting file size."), 5, TRUE);
	nRec = (DWORD)fileSize.QuadPart / RECSIZE;     /* Total number of records. Note
assumed limit */
	nRecTh = nRec / numFiles;    /* Records per thread. */
	threadArg = malloc (numFiles * sizeof (THREADARG));   /* Array of thread args. */
	pThreadHandle = malloc (numFiles * sizeof (HANDLE));

	/* Map the entire file */
	pRecords = MapViewOfFile(mHandle, FILE_MAP_ALL_ACCESS, 0, 0, 0);

	if (NULL == pRecords)
		ReportError (_T ("Failure to map input file."), 6, TRUE);
	CloseHandle (mHandle);
	/* Create the sorting threads. */
	lowRecordNum = 0;
	for (iTh = 0; iTh < numFiles; iTh++) {
		threadArg[iTh].iTh = iTh;
		threadArg[iTh].lowRecord = pRecords + lowRecordNum;
		threadArg[iTh].highRecord = pRecords + (lowRecordNum + nRecTh);
		lowRecordNum += nRecTh;
		pThreadHandle[iTh] = (HANDLE)_beginthreadex (
			NULL, 0, SortThread, &threadArg[iTh],
```

```
CREATE_SUSPENDED, NULL);
        }
        /* Resume all the initially suspened threads. */
        for (iTh = 0; iTh < numFiles; iTh++)
                ResumeThread (pThreadHandle[iTh]);
        /* Wait for the sort-merge threads to complete. */
        WaitForSingleObject (pThreadHandle[0], INFINITE);
        for (iTh = 0; iTh < numFiles; iTh++)
                CloseHandle (pThreadHandle[iTh]);

        /*  Print out the entire sorted file. Treat it as one single string. */
        stringEnd = (LPTSTR) pRecords + nRec*RECSIZE;
        *stringEnd =_T('\0');
        if (!noPrint) {
           _tprintf (_T("%s"), (LPCTSTR) pRecords);
        }
        UnmapViewOfFile(pRecords);
        // Restore the file length
        /* SetFilePointer is convenient as it's a short addition from the file end */
        if (!SetFilePointer(hFile, -2, 0, FILE_END) || !SetEndOfFile(hFile))

        ReportError (_T("Failure restore input file lenght."), 7, TRUE);
        CloseHandle(hFile);
        free (threadArg); free (pThreadHandle);
        return 0;
} /* End of _tmain. */
static VOID MergeArrays (LPRECORD, DWORD);
DWORD WINAPI SortThread (PTHREADARG pThArg)
{

        ReportError (_T("Failure restore input file lenght."), 7, TRUE);
        CloseHandle(hFile);
        free (threadArg); free (pThreadHandle);
        return 0;
} /* End of _tmain. */
static VOID MergeArrays (LPRECORD, DWORD);
DWORD WINAPI SortThread (PTHREADARG pThArg)
{

        DWORD groupSize = 2, myNumber, twoToI = 1;
                        /* twoToI = 2^i, where i is the merge step number. */
        DWORD_PTR numbersInGroup;
        LPRECORD first;
        myNumber = pThArg->iTh;
        first = pThArg->lowRecord;
        numbersInGroup = (DWORD)(pThArg->highRecord - first);
```

```
        /* Sort this portion of the array. */
        qsort (first, numbersInGroup, RECSIZE, KeyCompare);

        /* Either exit the thread or wait for the adjoining thread. */
        while ((myNumber % groupSize) == 0 && numbersInGroup < nRec) {
                                /* Merge with the adjacent sorted array. */
                WaitForSingleObject (pThreadHandle[myNumber + twoToI], INFINITE);
                MergeArrays (first, numbersInGroup);
                numbersInGroup *= 2;
                groupSize *= 2;
                twoToI *=2;
        }
        return 0;
}
static VOID MergeArrays (LPRECORD p1, DWORD nRecs)
{

        /* Merge two adjacent arrays, each with nRecs records. p1 identifies the first */
        DWORD iRec = 0, i1 = 0, i2 = 0;
        LPRECORD pDest, p1Hold, pDestHold, p2 = p1 + nRecs;
        pDest = pDestHold = malloc (2 * nRecs * RECSIZE);
        p1Hold = p1;
        while (i1 < nRecs && i2 < nRecs) {
                if (KeyCompare ((LPCTSTR)p1, (LPCTSTR)p2) <= 0) {
                        memcpy (pDest, p1, RECSIZE);
                        i1++; p1++; pDest++;
                }

                else {
                        memcpy (pDest, p2, RECSIZE);
                        i2++; p2++; pDest++;
                }
        }
        if (i1 >= nRecs)
                memcpy (pDest, p2, RECSIZE * (nRecs - i2));
        else    memcpy (pDest, p1, RECSIZE * (nRecs - i1));
        memcpy (p1Hold, pDestHold, 2 * nRecs * RECSIZE);
        free (pDestHold);
        return;
}
int KeyCompare (LPCTSTR pRec1, LPCTSTR pRec2)
{
        DWORD i;
        TCHAR b1, b2;
        LPRECORD p1, p2;
        int Result = 0;
```

```
        p1 = (LPRECORD)pRec1;
        p2 = (LPRECORD)pRec2;
        for (i = 0; i < KEYLEN && Result == 0; i++) {
                b1 = p1->key[i];
                b2 = p2->key[i];
                if (b1 < b2) Result = -1;
                if (b1 > b2) Result = +1;
        }
        return  Result;
}
```

- A thread structure is defined which is passed to every thread as an argument, this structure has the ‘iTh’ variable which specifies the thread number, ‘lowRecord’ and ‘highRecord’ which are used to define the starting and ending index of the sub list.
- ‘SortThread’ is the thread name that is to be created
- ‘nRec’ is the total number of threads created
- ‘pThreadHandle' is the handle to the thread
- The file, which has records in it, is opened to read and write. This file is mapped (‘mHandle’) to access that file as memory i.e. in the form of an array.
- Thread argument, which is to be passed as an argument to create a thread, is allocated the memory, that has all three fields that are to be passed
- ‘nRec’ is the total number of records
- ‘nRec/numFiles’ defines the number of records per file
- ‘pRecord’ is the address of the mapped file
- Different threads are created and run to divided the file into chunks and each thread sorts the file. To divide the file, a loop is run till ‘numFiles’ times, and specifies the thread numbers (‘ith’), starting index of a chunk (‘lowRecord’), and ending index of a chunk (‘highRecord’)
- The thread is created with ‘beginThread’ functions which are passed ‘SortThread’ and ‘ThreadArgument’ in a suspended state.
- When all the threads are created, they are resumed with help of a loop.
- For all created threads, we wait till the execution completes. Once the execution is completed handle is closed and the file is unmapped.
- When a thread is run, it is passed ‘QSort’ to sort the records, and then it waits for its adjacent thread to be sorted and completed. Once they are completed, the results are merged. Every thread waits for its adjacent thread to be completed and merges after completion

Performance

- Multithreading gives the best results when the number of processors is the same as the number of threads.
- Additional threads beyond processor count slows down the program
- Performance degrades if there is one processor and memory is low and the array is very large because most of the time threads will be contending for physical memory. But the problem alleviates when the memory is sufficient.

Topic No – 122

Introduction to Parallelism

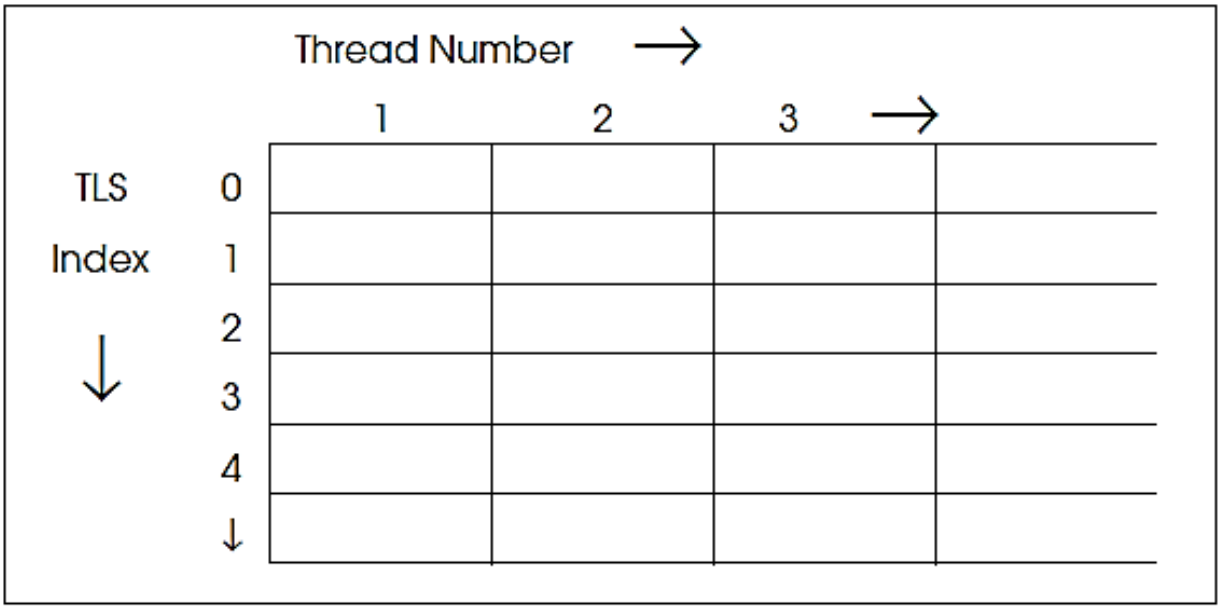
Multiprocessing and multithreading, both are responsible for multiple flows of execution, but the multithreading is optimal. Windows is not only multithreading or multiprocessing it also supports multiprocessors as well. If we have a system with multiprocessors, we should learn to program to use the potential of multiprocessors because the processor's speed has reached its bottleneck i.e. after a certain limit, its speed cannot be enhanced. Parallelization is the key to future performance improvement since we can no longer depend on increased CPU clock rates and since multicore and multiprocessor systems are increasingly common. Previously, various programs have been discussed that unleash the power of parallelism. The properties that enabled parallelism include the following:

- Major task is divided into subtasks and many worker threads were run. Subtasks were divided into worker threads that perform their work. These worker subtasks run independently, without any interaction between them.
- As subtasks are complete, a master program can merge the results of divided subtasks into a single result.
- The programs do not require mutual exclusion of any sort. Only the master worker is synchronized with each worker and waits for them to complete.
- Every worker will work as a separate thread on a separate processor. it is the most optimal situation
- Program performance scales automatically, up to some limit, as you run on systems with more processors; the programs themselves do not, in general, determine the processor count on the host computer. Instead, the Windows kernel assigns worker subtasks to available processors.
- Output remains undisturbed even if the program is serialized.
- If you "serialize" the program the results should get precisely the same as the parallel program. The serialized program is, moreover, much easier to debug.
- The maximum performance improvement is limited by the program's "parallelism," thread management overhead, and computations that cannot be parallelized.

Topic No – 123

Thread Local Storage

A thread is an execution unit. In a multithreading program, one procedure can have several threads. Every thread needs data, that it doesn't want to share with other threads and which is unique i.e. it varies from thread to thread. One technique is to have the creating thread call CreateThread (or beginThreadex) with lpvThreadParm pointing to a data structure that is unique for each thread. The thread can then allocate additional data structures and access them through lpvThreadParm. Windows also provides Thread Local Storage (TLS), which gives each thread its array of pointers. The following figure shows this TLS arrangement.



A function can have many threads i.e. thread 1, thread 2, etc. Every column in the TLS arrangement corresponds to thread numbers and every thread needs variables i.e. TLS index 0,1,2,3 etc. Initially, no TLS indexes (rows) are allocated, but new rows can be allocated and deallocated at any time. Once the row is allocated, it will be allocated to all rows. The primary thread would be a logical choice for TLS space management, however, every thread can access TLS.

- **DWORD TlsAlloc(VOID):** This API is used to allocate the index and it returns the TLS index in the form of the double word. Otherwise returns -1 in case of failure.
- **BOOL TlsFree(DWORD dwIndex)**: Frees the specified index.
- **LPVOID TlsGetValue (DWORD dwTlsIndex) and BOOL TlsSetValue (DWORD dwTlsIndex, LPVOID lpTlsValue):** Provided valid indexes are used. The programmer can access TLS space using these simple GET/SET APIs

Some Cautions

- TLS provides a convenient mechanism for accessing memory that is global within a thread but inaccessible to other threads.

- Global storage of a program is accessible by all threads
- TLS provides a convenient mechanism for accessing memory that is global within a thread but inaccessible to other threads.
- Global storage of a program is accessible by all threads

Topic No – 124

Processes and Thread priorities

In a multitasking or multithreading system, there are a number of processes running, each of which competes for resources like processor, memory, etc. The operating system is responsible for managing the resources. The Windows kernel always runs the highest-priority thread that is ready for execution. A thread is not ready if it is waiting, suspended, or blocked for some reason. Since the threads are dependents on the processes and they receive priority relative to their process priority classes. Process priority classes are set initially when they are created using CreateProcess, and each has a base priority, with values including

- **IDLE_PRIORTY_CLASS** for threads that will run only when the system is idle. This is the lowest priority process.
- **NORMAL_PRIORTY_CLASS** indicating no special scheduling requirements.
- **HIGH_PRIORTY_CLASS** indicating time-critical tasks that should be executed immediately.
- **REALTIME_PRIORTY_CLASS**, the highest possible priority

The priority class of a process can be set and got using **BOOL SetPriorityClass(HANDLE hProcess, DWORD dwPriority) and DWORD GetPriorityClass(HANDLE hProcess)** respectively.

There are some enhances variants of priority levels i.e. **ABOVE_NORMAL_PRIORTY_CLASS (**which is below HIGH_PRIORTY_CLASS) and **BELOW_NORMAL_PRIORTY_CLASS (**which is above IDLE_PRIOTY_CLASS).

PROCESS_MODE_BACKGROUND_BEGIN, which lowers the priority of the process and its threads for background work without affecting the responsiveness of foreground processes and threads. **PROCESS_MODE_BACKGROUND_END** restores the process priority to the value before it was set with PROCESS_MODE_BACKGROUND_BEGIN.

Thread priorities are either absolute or are set relative to the process base priority. At thread creation time, the priority is set to that of the process. The relative thread priorities are in a range of ±2 “points” from the process’s base. The symbolic names of the resulting common thread priorities, starting with the five relative priorities, are:

- THREAD_PRIORTY_LOWEST
- THREAD_PRIORTY_BELOW_NORMAL
- THREAD_PRIORTY_NORMAL
- THREAD_PRIORTY_ABOVE_NORMAL
- THREAD_PRIORTY_HIGHEST
- THREAD_PRIORTY_TIME_CRITICAL is 15, or 31 if the process class is REAL_TIME_PRIORTY_CLASS
- THREAD_PRIORTY_IDLE is 1, or 16 for REAL_TIME_PRIORTY_CLASSES processes

THREAD_MODE_BACKGROUND_BEGIN and **THREAD_MODE_BACKGROUND_END** are similar to **PROCESS_MODE_BACKGROUND_BEGIN** and **PROCESS_MODE_BACKGROUND_END**.

Thread priorities can be set within the range of +-2 of the corresponding process priority using these **BOOL SetThreadPriority(HANDLE hThread, int nPriority)** AND **Int GetThreadPriority(HANDLE hThread).**

Thread priorities are dynamic. They change with the priority of process or windows may also boost thread priority as per need. This feature can be enabled disabled using **SetThreadPriorityBoost()**

Topic No – 125

Thread States

The following figure shows how the executive manages threads and shows the possible thread states. This figure also shows the effect of program actions. Such state diagrams are common to all multitasking OSs and help clarify how a thread is scheduled for execution and how a thread moves from one state to another.

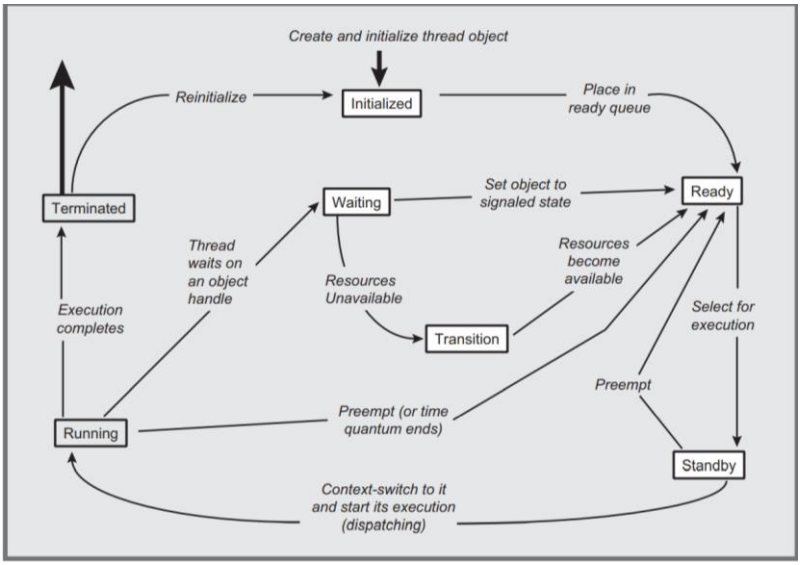


Figure 1 Thread States and Transitions

- A thread is in the running state when it is running on a processor. More than one thread can be in the running state on a multiprocessor computer
- The executive places a running thread in the wait state when the thread performs a wait on a non-signaled handle, such as a thread or process handle. I/O operations will also wait for the completion of a disk or other data transfer, and numerous other functions can cause waiting. It is common to say that a thread is blocked, or sleeping, when in the wait state
- A thread is ready if it could be running. The executive's scheduler could put it in the running state at any time. The scheduler will run the highest-priority ready thread when a processor becomes available, and it will run the one that has been in the ready state for the longest time if several threads have the same high priority. The thread moves through the standby state before entering the ready state.
- The executive will move a running thread to the ready state if the thread's time slice expires without the thread waiting. Executing will also move a thread from the running state to the ready state.
- The executive will place awaiting thread in the ready state as soon as the appropriate handles are signaled, although the thread goes through an intermediate transition state. It is common to say that the thread wakes up.

- A thread, regardless of its state, can be suspended, and a ready thread will not be run if it is suspended. If a running thread is suspended, either by itself or by a thread on a different processor, it is placed in the ready state.
- A thread is in the terminated state after it terminates and remains there as long as there are any open handles on the thread. This arrangement allows other threads to interrogate the thread's state and exit code.
- Normally, the scheduler will place a ready thread on any available processor. The programmer can specify a thread's processor, which will limit the processors that can run that specific thread. In this way, the programmer can allocate processors to threads and prevent other threads from using these processors, helping to assure responsiveness for some threads. The appropriate functions are SetProcessAffinityMask and GetProcessAffinityMas. SetThreadIdealProcessor can specify a preferred processor that the scheduler will use whenever possible; this is less restrictive than assigning a thread to a single processor with the affinity mask.

Topic No – 126

Mistakes while using Threads

There are several factors to keep in mind as you develop threaded programs; lack of attention to a few basic principles can result in serious defects, and it is best to avoid the problems in the first place than try to find them during testing or debugging.

- The essential factor is that the threads execute asynchronously.
- There is no sequencing unless you create it explicitly.
- This asynchronous behavior is what makes threads so useful, but without proper care, serious difficulties can occur.

Here are a few guidelines. There may be a few inadvertent violations, however, which illustrates the multithreaded programming challenges.

- Make no assumptions about the order in which the parent and child threads execute.
- It is possible for a child thread to run to completion before the parent, or, conversely, the child thread may not run at all for a considerable period.
- On a multiprocessor computer, the parent and one or more children may even run concurrently.
- Make sure all the initializations required by a child thread have been performed before calling CreateThread()
- In case a thread has been run but initialization is required then use some technique like thread suspension until data is initialized.
- Failure by the parent to initialize data required by the child is a common cause of “race conditions” wherein the parent “races” the child to initialize data before the child needs it.
- Any thread, at any time, can be preempted, and any thread, at any time, may resume execution
- Do not confuse synchronization and priority. These both are different concepts. Threads are defined as their priorities when created, However Threads are synchronized in such a way that one thread completes its specific purpose, and only then another thread may run its process.
- Even more so than with single-threaded programs, testing is necessary, but not sufficient, to ensure program correctness. It is common for a multithreaded program to pass extensive tests despite code defects. There is no substitute for careful design, implementation, and code inspection.
- Threaded program behavior varies widely with processor speed, number of processors, OS version, and more. Testing on a variety of systems can isolate numerous defects, but the preceding precaution still applies.
- The default stack size for a thread is 1MB. Make sure the size is sufficient as per thread needs.
- Threads should be used only as appropriate. Thus, if there are activities that are naturally concurrent, each such activity can be represented by a thread. If, on the other hand, the activities are naturally sequential, threads only add complexity and performance overhead.

- If you use a large number of threads, be careful, as the numerous stacks will consume virtual memory space and thread context switching may become expensive. In other cases, it could mean more threads than the number of processors.
- Fortunately, correct programs are frequently the simplest and have the most elegant designs. Avoid complexity wherever possible.

Topic No – 127

Timed Waits

In threading, there are some functions that can be used to wait for threads. The Sleep function allows a thread to give up the processor and move from the running to the wait state for a specified period of time. After the completion of the specified time, the thread will move to the ready state and will wait to move on running state. A thread can perform a task periodically by sleeping after carrying out the task.

VOID Sleep (DWord dwMilliseconds)

The time period is specified in milliseconds and can even be INFINITE, in which case the thread will never resume. A 0 value will cause the thread to relinquish the remainder of the time slice; the kernel moves the thread from the running state to the ready state.

The function SwitchToThread() provides another way for a thread to yield its processor to another ready thread if there is one that is ready to run.

Topic No – 128

Fibers

A fiber, as the name implies, is a piece of a thread. More precisely, fiber is a unit of execution that can be scheduled by the application rather than by the operating system. An application can create numerous fibers, and the fibers themselves determine which fiber will execute next. The fibers have independent stacks but otherwise run entirely in the context of the thread on which they are scheduled, having access, for example, to the thread's TLS and any mutexes owned by the thread. Furthermore, fiber management occurs entirely in user space outside the kernel. Fibers can be thought of as lightweight threads, although there are numerous differences. A fiber can execute on any thread, but never on two at one time. Fiber that is meant to run on different threads at different instances should not access thread-specific data from TLS.

Fiber Uses

- Fibers make portability easy.
- Fibers need not wait/block on file locks, mutexes, and pipes. They can pool resources and in case resources are not available they can switch to another fiber.
- Fiber exhibits flexibility. They exist as part of threads but they are not bound to any specific thread. They can run on any thread but only one at a time.
- Fibers are not pre-emptively scheduled. Windows OS is unaware of fibers. They are controlled and managed by fiber DLL.
- Fibers can be used as co-routines. Using this an application can switch between related tasks. Whereas, in the case of threads applications has no control over which thread executes next.
- Major software platform offers the use of fibers and claims its performance advantages.
-

Topic No – 129

Fiber APIs

A set of different API functions are provided that can help create and manage fibers. These functions are

A thread must enable fiber operation by calling

ConvertThreadToFiber() or

ConvertThreadToFiberEx()

After calling this API the thread will now contain a fiber. It will provide a pointer to the fiber data more or less like thread data. This can be used accordingly.

Subsequently, new fibers can be created in this thread using

CreateFiber()

Each new fiber has a start address, stack size, and a parameter. Each fiber is identified by address and not a handle.

Individual fiber can obtain their data by calling

GetFiberData()

Similarly, a fiber can obtain its identity by calling

GetCurrentFiber()

A running fiber gives control to another fiber using

SwitchToFiber()

It uses the address of the other fiber. The context of the current fiber is saved and the context of the other fiber is restored. Fibers must explicitly indicate the next fiber that is to run in the application.

A running fiber gives control to another fiber using

SwitchToFiber()

It uses the address of the other fiber. The context of the current fiber is saved and the context of the other fiber is restored. Fibers must explicitly indicate the next fiber that is to run in the application.

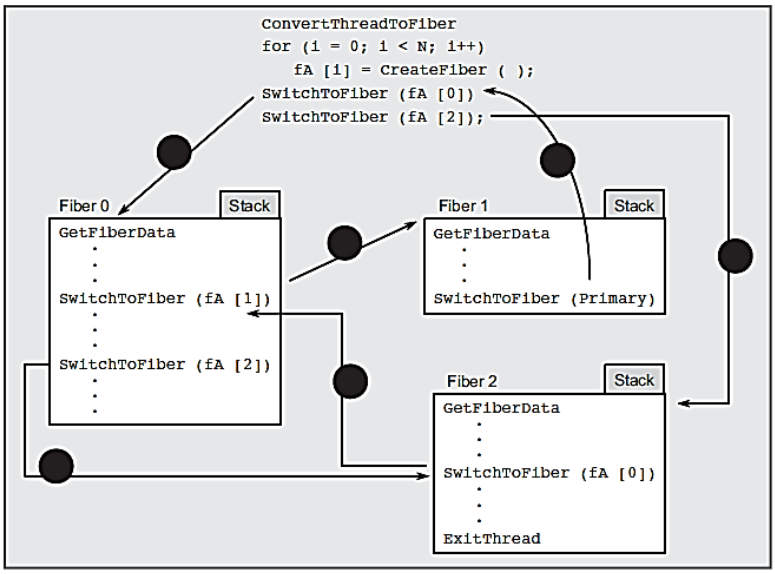
An existing fiber can be deleted using

DeleteFiber()

Topic No – 130

Using Fibers

Previously we discussed several APIs used to manage the fibers, here we will develop a scheme with these APIs to use the fibers. Fiber enables us to control the switching of threads. This scheme firstly converts a thread to fiber and then uses it as the primary fiber. The primary fiber creates other fibers and switching among these fibers is managed by the application.



In the center top, a primary thread is created in which ConvertThreadToFiber is used to convert into fiber and then with the help of a loop number of fibers are created. To convert the execution to a certain fiber SwitchToFiber is used. Primary fiber is switched to Fiber 0 which gets fiber data and then switches to Fiber 1 which also performs the same work i.e. gets data and this fiber switches to primary fiber. The primary fiber starts execution from the point where it was switched before i.e. now it switches to fiber 2 which gets data and switches to fiber 0 and then back to fiber 2 and at the end Thread is closed.

There are two policies

- Master-Slave Scheduling: One fiber decides which fiber to run. Each fiber transfers back the execution to the primary fiber. (Fiber 1)
- Peer to Peer Scheduling: A fiber determine which fiber should be next to execute based on some policy (Fiber 0 and 2)

Topic No – 131

Need for Thread Synchronization

Threads enable concurrent processing and parallelism in multiple processors. However, there are some pros and cons because of this concurrent processing. When many threads may run concurrently. They may need to synchronize for tasks in numerous instances. i.e. in the Boss-worker model, there are one boss thread and many workers threads, the Boss thread waits for all workers to complete execution and compiles all the data. In case if boss executes before the worker completes its execution, this may affect the output we are required to ensure that the Boss will not access the worker's memory unless the worker completes it. Alternately, the workers do not start working unless the boss has created all the workers so that a worker may not try to access the data of another worker which has not been created as yet. When multiple threads are using share data they may require coordination i.e. when a thread is using the data other threads should wait for it. Also, programmers need to ensure that two or more threads are not modifying the same data item simultaneously. In case many threads are modifying elements of the queue, the programmer needs to assure that two or more threads do not attempt to remove an element at the same time. Several programming flaws can leave such vulnerabilities in the code.

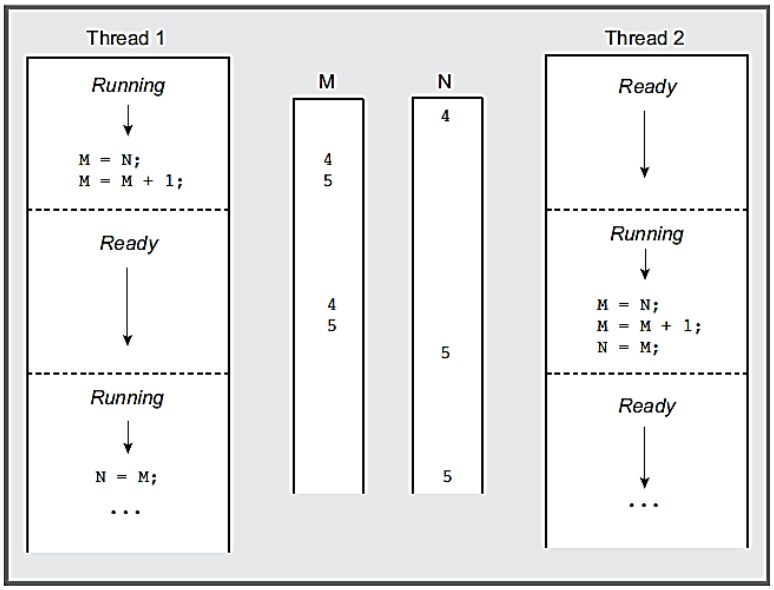
Example

There are two threads i.e. Thread 1 and Thread 2, operation of both of these is the same i.e.

```
Thread 1          Thread 2
{                 {
M = N             M = N
M = M + 1         M = M + 1
N = N             N = N
}                 }
```

Let's suppose that the initial value of N is 4 though the final values after execution of thread 1 will be M=5, N=5. After completion of thread 1 when thread 2 starts executing, the initial value of N is 5 (because of thread 1 execution) thus final values become M=6 and N=6.

When these same threads are run concurrently as shown in the following figure.



When the thread is executing, the value of N is 4 initially. After two instructions thread is switched. Thus the value of M becomes 5 and the value of N is not changed because 3rd instruction is not executed yet. Since the thread is switched therefore thread 2 starts execution and value N is 4 here. It executes all three instructions and values become M=5 and N=5. Again thread is switched and the remaining 1 instruction of thread 1 executes which affects the N variable and because of this final value of N becomes 5.

Final Values in normal processing

M=6 and N=6

Final Values in concurrent processing

M=5 and N=5

Thus we can see in many situations the final output of concurrent processing may not be same as it is in the normal processing

Topic No – 132

A Simple Solution to Critical Section Problem

When problems and errors arise because of parallelism or concurrent processing these types of errors are of critical section problem i.e. A critical resource which is used by number of processes or threads. This critical section problem can be handled by the following ways:

Using Global Variables

This solution uses a global variable named Flag to indicate to all threads that a thread is modifying a variable. A thread turns it TRUE before modifying a variable turns it to FALSE after modifying it. Each thread would check this Flag before modifying the variable. If the Flag is TRUE then if indicates that the variable is being used by some other thread.

```
BOOL Flag = FALSE;
DWORD N;
. . .
DWORD WINAPI ThreadFunc(TH_ARGS pArgs)
{
   . . .
   while (Flag) Sleep (1000);
   Flag = TRUE;
   N++;
   Flag = FALSE;
   . . .
}
```

Even in this case, the thread could be preempted between the time FLAG is tested and the time FLAG is set to TRUE; the first two statements form a critical code region that is not properly protected from concurrent access by two or more threads. Another attempted solution to the critical code region synchronization problem might be to give each thread its own copy of the variable, as follows:

```
DWORD WINAPI ThreadFunc (TH_ARGS pArgs)
{  DWORD N;
   ... N++; ...
}
```

Topic No – 133

Volatile Storage

The volatile stage is a Windows level or Compiler level facility provided when using incrementing operations which change the shared variables to reduce the conflicts that arise because of switching.

Latent Defects

Even if the synchronization problem is somehow resolved there still may remain some latent problems. A thread code may switch to another thread while a variable value has been modified in a register without writing it back. The use of registers for intermediate operations is a compiler optimization technique.

Turning off optimization may adversely affect the performance of the whole program but not always sometimes it may slow the program. ANSI C provides a qualifier volatile for this purpose. A variable with a volatile qualifier will always be accessed from memory for operations and will always be stored in memory after any operation. A volatile qualifier also means that the variable can be accessed at any instance. A volatile qualifier should only be used where necessary as it degrades the performance.

Using volatile

- Use volatile only if a variable is being modified by two or more threads.
- Even if it’s read only for two or more threads but the outcome of the threads depends on its new value.

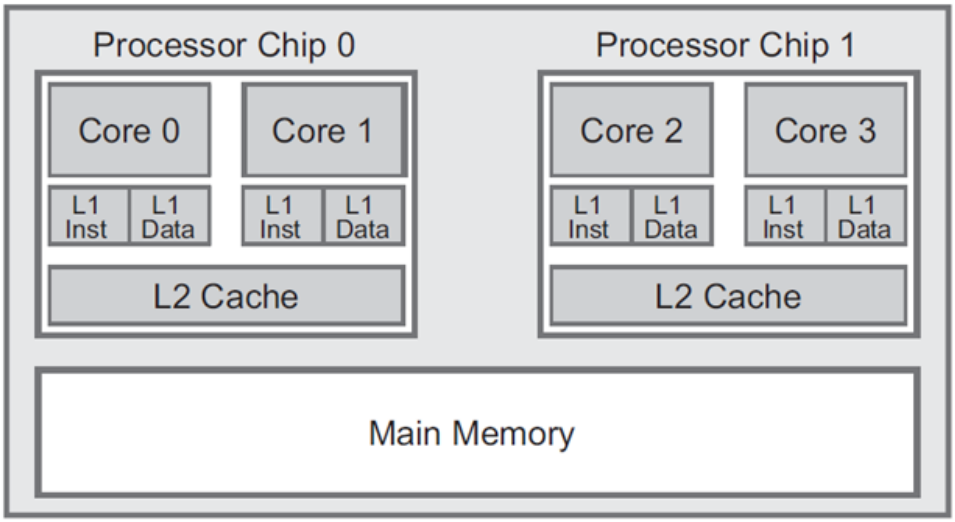
Content Development

Topic No - 134

Memory Architecture and Memory Barriers

Cache Coherency:
The volatile qualifier does not assure that the changes are visible to processors in a desired order. Processor usually hold the values in cache before writing them back to memory. This may alter the sequence in which different processors see the values.

Memory Barriers:
To assure that the memory is accesses in the desired order use memory barrier or memory fences. The interlocked function provide memory barrier. Further, the concept is clarified the diagram showing 4 processors in a system on two dual core chips.



Memory System Architecture

The diagram depicts 4 processor core on two chips. Each core has its own register with intermediate values of variables. Each core has separate Level-1 cache for instruction and data. A common larger L2 cache for cores on each chip. Memory is shared among cores.

Volatile qualifiers only assures that the new data values will be updated in L1. There is no guarantee that the values will be visible to other thread running on different processors. Memory barriers assure that the main memory is updated and cache of all processors is coherent.

For example if core 0 updates a variable N using a memory barrier. If core 3 L1 cache also contains the values of N then the value in its cache is either updated or removed so that core 3 could access the new value coherent value of N. This certainly incurs a huge cost. Moving data within the core registers costs less than a cycle whereas more data from one core to another via main memory can cost 100s of cycles.

Topic No - 135

Interlocked Functions:

Interlocked functions are most suited if variable with volatile scope only need to be incremented, decremented and exchanges. Interlocked functions are simpler and faster and easy to use. However, they do pose the performance drawback as they do generate a memory barrier.

The most basic of these functions are InterlockedIncrement() and InterlockedDecrement().

Long InterlockedIncrement(LONG volatile *Addend)

Long InterlockedDecrement(LONG volatile *Addend)

They both use a 32-bit signed variable as parameter that should be stored at 4-byte boundary in memory. They should be used whenever possible to improve performance.

For 64-bit systems InterlockedIncrement64() and InterlockedDecrement64() can be used with Addend placed at 8-byte boundary.

The function can be used in following way:

InterlockedIncrement(&N);

N should be volatile integer placed on appropriate memory boundary. Function return the new value of N. However some other thread may preempt before the value is returned and change the value of N. Do not call it twice to increment the value by two as the thread may be preempted between both calls. Rather use InterlockedExchangeAdd().

Topic No - 136

Local and Global Storage:

Another criterion for correct thread code is that global storage should not be used for local storage purpose. If a global variable is used to store thread specific data then it will be used by other threads as well. This will result in incorrect behavior no matter how the rest of code is written. The following example is incorrect usage of the global variables.

```
DWORD N;

....

DWORD WINAPI ThreadFunc (TH_ARGS pArgs)

{

        ...

        N = 2 * pArgs->Count; ...

        /* Use N; value is specific to this call to ThreadFunc*/

}
```

N is kept global but is used to store thread specific information from its parameters structure. For example, if many thread functions are running at the same time and each function modifies N with its own parameters, no function will be able to use N properly because N is global and all threads are running at the same time and affecting N's value. Due to this problem, the final results will be incorrect.

It's important to know when to use local and when to use global variables while dealing with these variables. If you know a variable contains thread-specific information, make it local. You can make a variable global if you know the information in it will be used by other threads.

Adhering to such practices can become even more convoluted when single threaded programs are converted to run as multi-threaded programs have threads being run in parallel. For example:

```
 DWORD N=0;

....

for(int i=0; i < MAX; ++i) {

        ...Allocate and initialize pArgs data ...
```

```
        N += ThreadFunc(pArgs);
}
...
DWORD WINAPI ThreadFunc (ARGS pArgs)
{
        DWORD result;
        ...
        result=...;
        return result;
}
```

The code above is for a single-threaded program. The program executes sequentially. The thread function is called repeatedly in a for loop. When a thread is called, the information it returns is stored in N, and the process is repeated for the next thread and so on. Because this is a single-threaded program, only one thread will be executed at a time. If you alter this code to a multi-threaded program, you will face issues.

Topic No - 137

How to write thread Safe Code:

Guide lines for writing thread safe code to ensure that the code runs smoothly in a threaded environment. When more than one thread can run the same code without introducing synchronization issues, it is said to be thread safe.

Variables that are required locally should not be accessible globally. They should be placed on Stack or in Data Structure passed to thread or the thread TLS. If a function is being used by several thread and it contains a variable that is thread specific such as a counter than store it in TLS or thread dedicated DS. Do not store it in global memory.

Avoid race conditions. If some required variables are uninitialized then create suspended threads until variables are initialized. If some condition needs to be met before a code block is executed then make sure the condition is met by waiting on synchronization objects.

Thread should not change the process environment. Changing environment by one thread will effect other threads. Thread should not change standard input or output devices. Also it should not change environment variables. A primary thread may change process environment as an exception. In that case the same environment used by rest of threads.

As a principle primary thread ensures that no other thread changes the process environment. All variable that are meant to be shared among threads are either kept global or static. They are protected by synchronization or interlocked mechanism using memory barriers.

Topic No - 138

Thread Synchronization Objects

Windows support different types of synchronization objects. The objects can be used to enforce synchronization and mutual exclusion.

Synchronization Mechanism

One method that has been used is by WaitForSingleObject() or WaitForMultipleObjects(). Using this method a thread waits for other threads to terminate. Other method used File Locks when multiple processes tries to access a file. Windows provides four different synchronization objects i.e. Mutexes, Semaphores, Events and Critical Section.

Risks in Using Synchronization Objects

There are always inherent risks involved in the use of such objects such as deadlocks. Care is required while using these objects. In higher version of Windows other objects like SRW locks and Condition variables are also available. Other advanced objects are waitable timers and IO completion ports.

Topic No - 139

CRITICAL_SECTION Objects

Critical Section

Critical section is the part of the code that can only be executed by one thread at a time. Windows provides the CRITICAL_SECTION objects as a simple lock mechanism for solving critical section problem. CRITICAL_SECTION Objects are initialized and deleted but do not have handles and are not shared among processes. Only one thread at a time can be in the CS variable, although many threads may enter and leave CS at numerous instances.

The function used to initialize and delete CRITICAL_SECTION are

VOID InitializeCriticalSection(LPCRITICAL_SECTION lpCriticalSection)

VOID DeleteCriticalSection(LPCRITICAL_SECTION lpCriticalSection)

When a thread wants to enter into the critical section it must call EnterCriticalSection() and when it leaves it should call LeaveCriticalSection().

A call to LeaveCriticalSection() must match an EnterCriticalSection(). Multiple threads may call EnterCriticalSection() but only one thread is allowed to enter the critical section while the rest are blocked. A blocked thread can enter the critical section when LeaveCriticalSection() is invoked.

VOID EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection)

VOID LeaveCriticalSection(LPCRITICAL_SECTION lpCriticalSection)

Recursive Critical Section

If one thread has entered a CS it can enter again. Windows maintain a count. Thread will have to leave as many times as it enters. This is implemented to support recursive functions and to make shared library functions thread safe. There is no timeout to EnterCriticalSection(). A thread will remain blocked forever if a matching Leave call is not received. A thread will remain blocked forever if a matching Leave call is not received.

However as thread can always poll to see whether another thread owns a CS using the function:

BOOL TryEnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection)

If the function returns TRUE then it indicates that the current thread now owns the CS. If it returns false then it indicates that the CS is owned by some other thread and it is not safe to enter CS.

Since CRITICAL_SECTION is a user space object therefore it has apparent advantages over other kernel space objects.

Topic No - 140

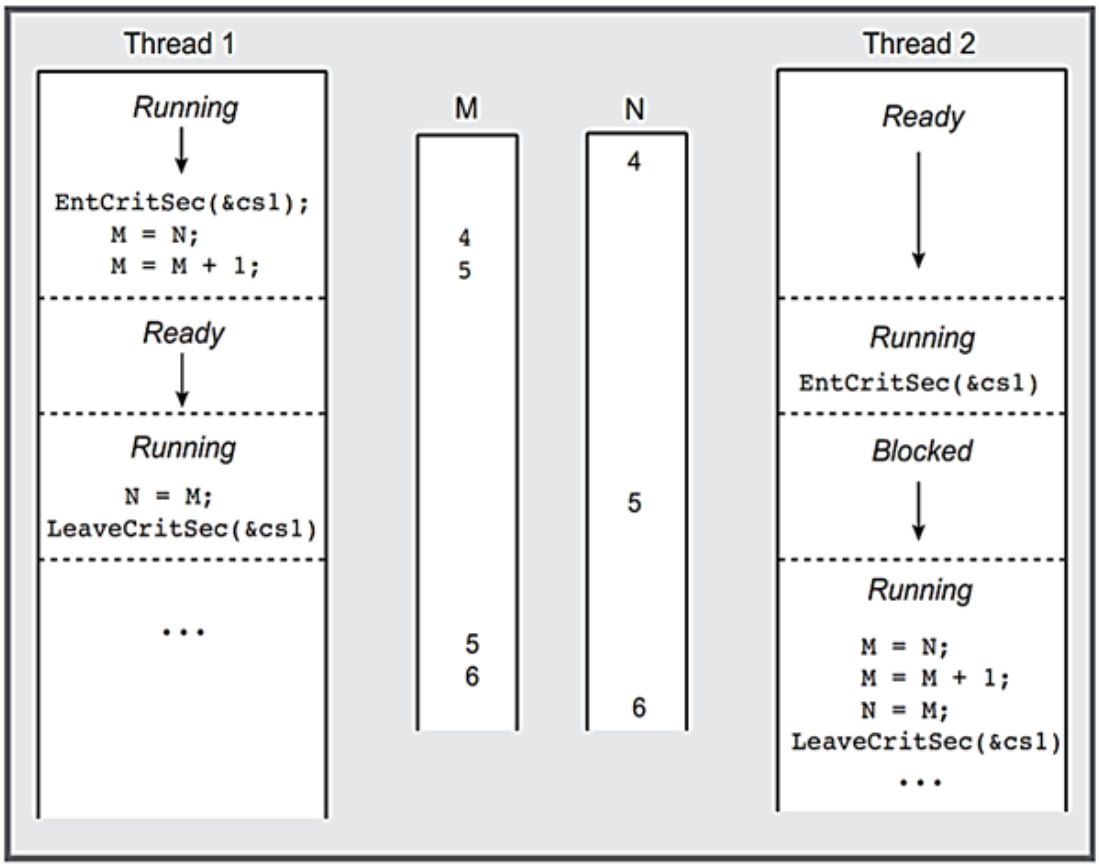
CRITICAL_SECTION for Protecting Shared Variables

In this module, we will study how to use protected shared variables in critical section and how critical section can assist us use protected shared variables.

Using the critical section construct is easy and intuitive. Consider an example of a server that maintains status related information in shared variables, information like number of requests, number of responses and requests currently under process. Since such count variables are shared therefore only one thread can be allowed to modify it at a time.

CRITICAL_SECTION construct can be used easily to ensure this. In this solution also an intermediate variable is used to emphasize the role of CRITICAL_SECTION.

```
 CRITICAL_SECTION cs1;
 volatile DWORD N = 0;
/* N is a global variable, shared by all threads. */
InitializeCriticalSection (&cs1);
    . . .
/* Create one or more threads using ThreadFunc */
    . . .
DWORD WINAPI ThreadFunc (TH_ARGS pArgs);
{
    DWORD M;
    __try {
        EnterCriticalSection (&cs1);
    if (N < N_MAX) { M = N; M += 1; N = M; }
    } __finally {
        LeaveCriticalSection (&cs1)
    }
}
    ...
DeleteCriticalSection (&cs1);
```



Topic No 141:

Protect a Variable with a Single Synchronization Object

In this module, we will discuss a guideline how to protect shared variables, how to use synchronization object, and what kind of mapping is there between synchronization object and variables like one to one or one to many.

Single Synchronization Object

All the variables within the critical section must be guarded by a single object. Using different objects within the same thread or using different objects across numerous threads sharing same data would be incorrect. Shared variables must be protected by a single object across all threads for mutual exclusion to work.

Below is given an example of incorrect use of synchronization object. In this example, two different objects are using the same variable N. Such code may generate incorrect results therefore share all variables using a single object.

```
DWORD WINAPI ThreadFunc (ARGS pArgs)
{
    . . .
    EnterCriticalSection(&cs1);
    N += 5;
    LeaveCriticalSection(&cs1);
    . . .
    InterlockedDecrement(&N);
    . . .
    EnterCriticalSection(&cs2);
    N -= 5;
    LeaveCriticalSection(&cs2);
    . . .
}
```

Topic No 142:

Producer-Consumer Problem

In this module, we will discuss a version of Producer-Consumer problem.

Producer-Consumer Threads

Producer consumer problem is a classical problem in mutual exclusion. It has many versions. Here we describe a simplistic version. This version clearly shows how to build and protect data structures for storing objects. Also we discuss how to establish invariant properties of variables which are always TRUE outside CS. In addition to primary threads there are two more threads: a producer and a consumer thread. The producer periodically creates a message. The message is contained in a table. The consumer on request of the user displays the message. The displayed data must be most recent and no data should be displayed twice. Do not display data while it is being updated by the producer. Do not display old data. Some of the produced messages may never be used and maybe lost. This is also like the pipeline model in which a message moves from one thread to another. The producer also computes a simple checksum of the message. Consumer makes sure by checking the checksum. If the consumer accesses the table while it is being updated the table will be invalid. CS ensures that this does not happen. The invariant is that the checksum is correct for current message contents.

Topic No 143:

Sample Program for Producer-Consumer Problem

Producer-Consumer

Producer consumer problem is a classical problem in mutual exclusion. Based on limitations and function described the following program is developed:

```
/* Chapter 9. simplePC.c */
/* Maintain two threads, a Producer and a Consumer */
/* The Producer periodically creates checksummed mData buffers, */
/* or "message block" which the Consumer displays when prompted */
/* by the user. The consumer reads the most recent complete */
/* set of mData and validates it before display */

#include "Everything.h"
#include <time.h>
#define DATA_SIZE 256
typedef struct MSG_BLOCK_TAG { /* Message block */
        CRITICAL_SECTION mGuard;       /* Guard the message block structure */
        DWORD fReady, fStop;
                /* ready state flag, stop flag   */
        volatile DWORD nCons, mSequence; /* Message block mSequence number         */
        DWORD nLost;
time_t mTimestamp;
        DWORD mChecksum; /* Message contents mChecksum               */
```

```
        DWORD mData[DATA_SIZE]; /* Message Contents                */
} MSG_BLOCK;
/*      One of the following conditions holds for the message block     */
/*        1)    !fReady || fStop                                        */
/*      nothing is assured about the mData              OR              */
/*        2)    fReady && mData is valid                                */
/*                        && mChecksum and mTimestamp are valid         */
/* Also, at all times, 0 <= nLost + nCons <= mSequence                   */
/* Single message block, ready to fill with a new message    */
MSG_BLOCK mBlock = { 0, 0, 0, 0, 0 };
DWORD WINAPI Produce (void *);
DWORD WINAPI Consume (void *);
void MessageFill (MSG_BLOCK *);
void MessageDisplay (MSG_BLOCK *);
int _tmain (int argc, LPTSTR argv[])
{
        DWORD status;
        HANDLE hProduce, hConsume;

        /* Initialize the message block CRITICAL SECTION */
        InitializeCriticalSection (&mBlock.mGuard);
        /* Create the two threads */
        hProduce = (HANDLE)_beginthreadex (NULL, 0, Produce, NULL, 0, NULL);
```

```
        if (hProduce == NULL)
                ReportError (_T("Cannot create Producer thread"), 1, TRUE);
        hConsume = (HANDLE)_beginthreadex (NULL, 0, Consume, NULL, 0, NULL);
        if (hConsume == NULL)
                ReportError (_T("Cannot create Consumer thread"), 2, TRUE);

        /* Wait for the Producer and Consumer to complete */

        status = WaitForSingleObject (hConsume, INFINITE);
        if (status != WAIT_OBJECT_0)
                ReportError (_T("Failed waiting for Consumer thread"), 3, TRUE);
        status = WaitForSingleObject (hProduce, INFINITE);
        if (status != WAIT_OBJECT_0)
                ReportError (__T("Failed waiting for Producer thread"), 4, TRUE);
        DeleteCriticalSection (&mBlock.mGuard);
        _tprintf (_T("Producer and Consumer threads have terminated\n"));
        _tprintf (_T("Messages Produced: %d, Consumed: %d, Lost: %d.\n"),
                mBlock.mSequence, mBlock.nCons, mBlock.mSequence - mBlock.nCons);
        return 0;
}
DWORD WINAPI Produce (void *arg)
/* Producer thread - Create new messages at random intervals */
{
```

```
        srand ((DWORD)time(NULL)); /* Seed the random # generator      */


        while (!mBlock.fStop) {
                /* Random Delay */
                Sleep(rand()/100);


                /* Get the buffer, fill it */


                EnterCriticalSection (&mBlock.mGuard);
                __try {
                        if (!mBlock.fStop) {
                                mBlock.fReady = 0;
                                MessageFill (&mBlock);
                                mBlock.fReady = 1;
                                InterlockedIncrement (&mBlock.mSequence);
                        }
                }
                __finally { LeaveCriticalSection (&mBlock.mGuard); }
        }
        return 0;
}
DWORD WINAPI Consume (void *arg)
{
```

```
CHAR command, extra;
/* Consume the NEXT message when prompted by the user */
while (!mBlock.fStop) { /* This is the only thread accessing stdin, stdout */
        _tprintf (_T("\n**Enter 'c' for Consume; 's' to stop: "));
        _tscanf (_T("%c%c"), &command, &extra);
        if (command == _T('s')) {
                /* ES not needed here. This is not a read/modify/write.
                 * The Producer will see the new value after the Consumer returns */
                mBlock.fStop = 1;
        } else if (command == _T('c')) { /* Get a new buffer to Consume */
                EnterCriticalSection (&mBlock.mGuard);
                __try {
                        if (mBlock.fReady == 0)
                                _tprintf (_T("No new messages. Try again later\n"));
                        else {
                                MessageDisplay (&mBlock);
                                mBlock.nLost = mBlock.mSequence - mBlock.nCons + 1;
                                mBlock.fReady = 0; /* No new messages are ready */
                                InterlockedIncrement(&mBlock.nCons);
                        }
                }
                __finally { LeaveCriticalSection (&mBlock.mGuard); }
        } else {
```

```
                    _tprintf (_T("Illegal command. Try again.\n"));
                }
        }
        return 0;
}
void MessageFill (MSG_BLOCK *msgBlock)
{
        /* Fill the message buffer, and include mChecksum and mTimestamp */
        /* This function is called from the Producer thread while it */
        /* owns the message block mutex*/

        DWORD i;

        msgBlock->mChecksum = 0;
        for (i = 0; i < DATA_SIZE; i++) {
                msgBlock->mData[i] = rand();
                msgBlock->mChecksum ^= msgBlock->mData[i];
        }
        msgBlock->mTimestamp = time(NULL);
        return;
}
void MessageDisplay (MSG_BLOCK *msgBlock)
{
```

```
        /* Display message buffer, mTimestamp, and validate mChecksum  */
        /* This function is called from the Consumer thread while it          */
        /* owns the message block mutex                                       */
        DWORD i, tcheck = 0;


        for (i = 0; i < DATA_SIZE; i++)
                tcheck ^= msgBlock->mData[i];
        _tprintf (_T("\nMessage number %d generated at: %s"),
                msgBlock->mSequence, _tctime (&(msgBlock->mTimestamp)));
        _tprintf (_T("First and last entries: %x %x\n"),
                msgBlock->mData[0], msgBlock->mData[DATA_SIZE-1]);
        if (tcheck == msgBlock->mChecksum)
                _tprintf (_T("GOOD ->mChecksum was validated.\n"));
        else
                _tprintf (_T("BAD  ->mChecksum failed. message was corrupted\n"));


        return;
}
```

Topic No 144:

Mutexes

In this module we will discuss mutexes. Mutex, a short form of Mutual Exclusion. Windows provides an object called a mutex. Using this object we can enforce mutual exclusion.

Mutexes have some advantages beyond CRITICAL_SECTION. Mutexes can be named and have handles. They can also be used for interprocess communication between threads in different processes. For example, if two processes share files through memory maps, then mutexes can be used for protection. Mutexes also allow timeout values. Mutexes can automatically become signaled once abandoned by the terminating thread. A thread gains ownership to mutex by waiting successfully on mutex handle using WaitForSingleObject() or WaitForMultipleObject(). Ownership is released using ReleaseMutex(). A thread should be careful about releasing a thread as soon as possible. A thread can acquire a single mutex several times. A thread will not be blocked if it already has ownership. The recursive property holds for Mutexes also.

Windows functions used to manage mutexes are

- CreateMutex()
- ReleaseMutex()
- OpenMutex()

```
HANDLE CreateMutex (
   LPSECURITY_ATTRIBUTES lpsa,
   BOOL bInitialOwner,
   LPCTSTR lpMutexName)
```

If bInitialOwner is TRUE, handover ownership to the calling thread immediately. However, it is ignored if the mutex already exists as per its name.

lpMutexName is the name of the mutex. The name is assigned as per rules of the Windows namespace.

If NULL is returned it indicates failure.

OpenMutex() is used to open and existing named mutex. The Open operation is followed by Create. The main thread would usually create a mutex while other threads can open it. This construct allows to synchronize threads from different processes.

```
BOOL ReleaseMutex (HANDLE hMutex)
```

Similarly ReleaseMutex() releases the ownership of the Mutex for a thread. It fails if the Mutex is not already owned by the thread.

Topic No 145:

Mutexes, CRITICAL_SECTIONS and Deadlocks

In this module we will discuss deadlocks. Deadlocks may arise when Mutexes or CRITICAL SECTIONS are used.

Using Concurrency Objects

Concurrency objects must be used carefully, otherwise it can lead to a deadlock situation. Deadlocks are a byproduct of concurrency control. It occurs when two or more objects try to lock a resource at the same time.

An Example

For instance, there are two lists (A and B) with the same structure maintained by different worker threads. In one situation, it can be possible that an operation is only allowed if a certain element is either present in both the lists or none. An operation is invalid if an element is present in just one.

In another situation, an element in one list can not be in the other. Based on these situations, concurrency objects are required for both lists. Using a single mutex for both lists will degrade performance by restricting concurrent updates.

Next slide shows an implementation for this situation.

```
static struct {
   /* Invariant: List is a valid list. */
   HANDLE guard; /* Mutex handle. */
   struct ListStuff;
}  ListA, ListB;
...
DWORD WINAPI AddSharedElement (void *arg)
/* Add a shared element to lists A and B. */
{ /* Invariant: New element is in both or neither list. */
   WaitForSingleObject (ListA.guard, INFINITE);
   WaitForSingleObject (ListB.guard, INFINITE);
   /* Add the element to both lists ... */
   ReleaseMutex (ListB.guard);
   ReleaseMutex (ListA.guard);
   return 0;
}
DWORD WINAPI DeleteSharedElement (void *arg)
/* Delete a shared element to lists A and B. */
{
   WaitForSingleObject (ListB.guard, INFINITE);
   WaitForSingleObject (ListA.guard, INFINITE);
   /* Delete the element from both lists ... */
   ReleaseMutex (ListB.guard);
   ReleaseMutex (ListA.guard);
   return 0;
}
```

Avoiding deadlock situation

One method to avoid this situation is to maintain a hierarchy. All threads should follow the same hierarchy. They should acquire and release mutexes in the same order. In the example, the situation can be easily avoided if both the threads acquire mutexes in the same order.

Another technique that can be used is an array of mutexes and using WaitForMultipleObjects() with the fWaitAll flag. In this case, the thread will either acquire both the mutexes or none. This technique is not possible with CRITICAL_SECTION.

Topic No 146:

Mutexes and CS

This module is about Mutexes VS CRITICAL_SECTIONs. Mutexes and CS are similar in the sense that they are used to achieve the same goal. Both can be owned by a single thread, and other threads trying to gain access to a resource shall be denied access until the resource is released. However, there are few advantages of using mutexes against some performance drawbacks.

Advantages of Mutexes

Mutexes that are abandoned due to the abrupt termination of a thread are automatically signaled so that waiting threads are not blocked permanently. However, abrupt termination of a thread indicates a serious programming flaw. Mutex waits can time out, whereas CS wait does not. As they are named, mutexes are shareable over numerous threads in different processes. Threads that create the mutexes can acquire immediate ownership (slight convenience). However, in most cases, CSs will work considerably faster.

Topic No 147:

Semaphores

This module is about semaphores. Semaphore is simply a data structure that is used for concurrency control.

Semaphore Count

Semaphore maintains a count. A semaphore is in a signaled state when the count is greater than 0. Semaphore is unsignaled when the count is 0. Threads use the wait function on semaphore. The count is decremented when a waiting thread is released. The following functions are used: CreateSemaphore(), CreateSemaphoreEx(), OpenSemaphore(), and ReleaseSemaphore().

```
HANDLE CreateSemaphore (
    LPSECURITY_ATTRIBUTES lpsa,
    LONG lSemInitial,
    LONG lSemMax,
    LPCTSTR lpSemName)
```

lSemMax must be 1 or greater, which is the maximum value of semaphore count.

lSemInitial is the initial value.

0<= lSemInitial<= lSemMax.

A NULL returns in case of failure.

```
BOOL ReleaseSemaphore (
    HANDLE hSemaphore,
    LONG cReleaseCount,
    LPLONG lpPreviousCount)
```

lpPreviousCount gives the count preceding the release.

cReleaseCount gives the count after the release and must be greater than 0. The call will fail and return FALSE if it would cause the count to exceed the maximum, and the count will remain unchanged. Also, in this case, the release count will not be valid.

Any thread can release the semaphore, not just the one that acquired its ownership. Hence, there is no concept of abandonment.

Topic No 148:

Using Semaphores

This module is about using semaphores. We have already discussed semaphores. Now we will discuss how to use the structure and functions in a program to enforce mutual exclusion.

Semaphore Concepts

Classically, a semaphore count represents the number of available resources. Sem Maximum represents the number of resources. In a producer-consumer scenario, the producer would place an element in the queue and call ReleaseSemaphore(). The consumer will wait on the semaphore and decrease the count after consuming the item.

Thread Creation

Previously, in many examples, numerous threads were created in a suspended state and were not activated until all the threads were created. This problem can be handled by semaphores without suspending threads. A newly created thread will wait on a semaphore with the semaphore initialized by 0. The boss thread will call ReleaseSemaphore with the count set to the number of threads.

Topic No 149:

Semaphore Limitation

In this module we will discuss limitations of semaphore.

Limitations

Some limitations are encountered while using windows semaphore. How can a thread request to decrease the count by 2 or more? The thread will have to wait twice on the semaphore. Calling wait twice will not be atomic and execution may switch in between.

Below is given a code of two threads, thread1 on the left side and thread2 on the right side. The problem with the below code is that the wait functions are not atomic; they are separate wait functions. Because they are not single functions, switching between these two wait functions (after completion of first wait function) in thread1 may occur, and execution control may reach the thread2 wait function, resulting in a deadlock-like situation. So this is one of the limitations of semaphores.

```
/* hSem is a semaphore handle.
The maximum semaphore count is 2. */
   ...
/* Decrement the semaphore by 2. */
WaitForSingleObject (hSem, INFINITE);
WaitForSingleObject (hSem, INFINITE);
   ...
/* Release two semaphore counts. */
ReleaseSemaphore (hSem, 2, &PrevCount);
```

```
/* hSem is a semaphore handle.
The maximum semaphore count is 2. */
   ...
/* Decrement the semaphore by 2. */
WaitForSingleObject (hSem, INFINITE);
WaitForSingleObject (hSem, INFINITE);
   ...
/* Release two semaphore counts. */
ReleaseSemaphore (hSem, 2, &PrevCount);
```

How can we deal with this limitation? Make the two wait functions atomic by using a CriticalSection to avoid switching during the wait, as shown in the below code.

```
/* Decrement the semaphore by 2. */
EnterCriticalSection (&csSem);
WaitForSingleObject (hSem, INFINITE);
WaitForSingleObject (hSem, INFINITE);
LeaveCriticalSection (&csSem);
   ...
ReleaseSemaphore (hSem, 2, &PrevCount);
```

Other Solutions

Another solution one can suggest is the use of WaitForMultipleObjects() with an array holding multiple references to the same semaphore. However, this solution will fail readily as the call to WaitForMultipleObjects() fails when it detects duplicate objects in the array. Secondly, all the handles may get signals even if the count is 1.

Topic No 150:

Events

In this module, we will discuss Events. Events are synchronization objects like Mutexes, Semaphores and Critical Section that can be used for concurrency control.

Events can indicate to other threads that a specific condition now holds, such as some message being available. Multiple threads can be released from wait simultaneously when an event is triggered. Events are classified as either manual-reset or auto-reset. Event property is set using CreateEvent(). A manual-reset event can signal several waiting threads simultaneously and can be reset. An auto-reset event signals a single waiting thread, and the event is reset automatically. The functions used to handle events are CreateEvent(), CreateEventEx(), OpenEvent(), SetEvent(), ResetEvent() and PulseEvent().

```
HANDLE CreateEvent (
    LPSECURITY_ATTRIBUTES lpsa,
    BOOL bManualReset,
    BOOL bInitialState,
    LPTCSTR lpEventName)
```

The function creates an event. bManualReset is set to TRUE to set a manual reset event. If bInitialState is TRUE, the event is set to a signaled state. A named event can be opened using OpenEvent() from any process.

```
BOOL SetEvent (HANDLE hEvent)
BOOL ResetEvent (HANDLE hEvent)
BOOL PulseEvent (HANDLE hEvent)
```

The above three functions are used to control events. A thread can signal the event by using SetEvent(). In the case of auto-reset, a single thread is released out of many. Event automatically returns to a non-signaled state. If no threads are waiting, the event remains in a signaled state until a thread waits on it. In this case, the thread will be immediately released.

If the event is manual-reset, it remains signaled until a thread explicitly calls ResetEvent(). Meanwhile, all the waiting threads are released. Consequently, other threads may also wait to be released immediately before the reset.

PulseEvent() releases all threads currently waiting for a manual reset. The event is automatically reset. In the case of an auto-reset event, PulseEvent() releases a single waiting thread, if any.

Topic No 151:

Event Usage Models

In this module, we will discuss Event Usage Models. Event can be implemented using different types of models. These models are designed according to the usage.

Combinations

There can be four distinct ways to use events. These four models culminate from combinations of SetEvent(), PulseEvent() and use of auto and manual events. Each combination proves useful depending on the situation. The combinations are highlighted in the table.

	Auto-Reset Event	Manual-Reset Event
`SetEvent`	Exactly one thread is released. If none is currently waiting on the event, the first thread to wait on it in the future will be released immediately. The event is automatically reset.	All currently waiting threads are released. The event remains signaled until reset by some thread.
`PulseEvent`	Exactly one thread is released, but only if a thread is currently waiting on the event. The event is then reset to nonsignaled.	All currently waiting threads, if any, are released, and the event is then reset to nonsignaled.

Understanding Events

An auto-reset event can be conceptualized as a door that automatically shuts when opened. Whereas the manual reset does not shuts automatically when opened. Hence, PulseEvent() can be considered as a door that is open which shuts when a single thread passes through in the case of auto-reset. While in the case of a manual reset, multiple waiting threads can pass through. SetEvent(), on the other hand, simply opens the door and releases a thread.

Topic No 152:

Producer-Consumer Solution Using Events

In this module, we will try to solve Producer-Consumer problem using Events.

Producer Consumer

Here is another example that provides a solution to our Producer-Consumer problem using Events. The solution uses Mutexes rather than CSs. A combination of auto-reset and SetEvent() is used in the producer to ensure only one thread is released. Mutexes in the program make access to the message data structure mutually exclusive while an event is used to signal the availability of new messages.

```
#include "Everything.h"

#include <time.h>

#define DATA_SIZE 256

typedef struct MSG_BLOCK_TAG { /* Message block */

        HANDLE      mGuard;             /* Mutex to uard the message block structure*/

        HANDLE      mReady;             /* "Message ready" auto-reset event   */

        DWORD       fReady, fStop;      /* ready state flag, stop flag   */

        volatile DWORD mSequence;       /* Message block mSequence number*/

volatile DWORD nCons, nLost;

        time_t mTimestamp;

        DWORD       mChecksum;          /* Message contents mChecksum*/

        DWORD       mData[DATA_SIZE]; /* Message Contents*/

} MSG_BLOCK;

/*One of the following conditions holds for the message block */

/*1)    !fReady || fStop*/

/*nothing is assured about the mData          OR */

/*2)    fReady && mData is valid*/

/* && mChecksum and mTimestamp are valid*/

/*  Also, at all times, 0 <= nLost + nCons <= mSequence*/

/* Single message block, ready to fill with a new message    */
```

```
MSG_BLOCK mBlock = { 0, 0, 0, 0, 0 };
DWORD WINAPI Produce (void *);
DWORD WINAPI Consume (void *);
void MessageFill (MSG_BLOCK *);
void MessageDisplay (MSG_BLOCK *);
int _tmain (int argc, LPTSTR argv[])
{
        DWORD status;
        HANDLE hProduce, hConsume;

        /* Initialize the message block mutex and event (auto-reset) */
        mBlock.mGuard = CreateMutex (NULL, FALSE, NULL);
        mBlock.mReady = CreateEvent (NULL, FALSE, FALSE, NULL);
        /* Create the two threads */
        hProduce = (HANDLE)_beginthreadex (NULL, 0, Produce, NULL, 0, NULL);
if (hProduce == NULL)
                ReportError (_T("Cannot create producer thread"), 1, TRUE);
        hConsume = (HANDLE)_beginthreadex (NULL, 0, Consume, NULL, 0, NULL);
        if (hConsume == NULL)
                ReportError (_T("Cannot create consumer thread"), 2, TRUE);
        /* Wait for the producer and consumer to complete */
        status = WaitForSingleObject (hConsume, INFINITE);
if (status != WAIT_OBJECT_0)
```

```
            ReportError (_T("Failed waiting for consumer thread"), 3, TRUE);
        status = WaitForSingleObject (hProduce, INFINITE);
        if (status != WAIT_OBJECT_0)
            ReportError (__T("Failed waiting for producer thread"), 4, TRUE);
        CloseHandle (mBlock.mGuard);
        CloseHandle (mBlock.mReady);
        _tprintf (_T("Producer and consumer threads have terminated\n"));
_tprintf (_T("Messages produced: %d, Consumed: %d, Known Lost: %d\n"),
            mBlock.mSequence, mBlock.nCons, mBlock.mSequence - mBlock.nCons);
        return 0;
}
DWORD WINAPI Produce (void *arg)
/* Producer thread - Create new messages at random intervals */
{
        srand ((DWORD)time(NULL));        /* Seed the random # generator        */
while (!mBlock.fStop) {
            /* Random Delay */
            Sleep(rand()/5); /* wait a long period for the next message */
                /* Adjust the divisor to change message generation rate */
            /* Get the buffer, fill it */

            WaitForSingleObject (mBlock.mGuard, INFINITE);
            __try {
```

```
                        if (!mBlock.fStop) {
                                mBlock.fReady = 0;
                                MessageFill (&mBlock);
                                mBlock.fReady = 1;
                        InterlockedIncrement(&mBlock.mSequence);
                                SetEvent(mBlock.mReady); /* Signal that a message is ready. */
                        }
                }
                __finally { ReleaseMutex (mBlock.mGuard); }
        }
        return 0;
}
DWORD WINAPI Consume (void *arg)
{
DWORD ShutDown = 0;
        CHAR command[10];
        /* Consume the NEXT message when prompted by the user */
        while (!ShutDown) { /* This is the only thread accessing stdin, stdout */
                _tprintf (_T ("\n**Enter 'c' for Consume; 's' to stop: "));
                _tscanf_s (_T("%9s"), command, sizeof(command)-1);
                if (command[0] == _T('s')) {
                        WaitForSingleObject (mBlock.mGuard, INFINITE);
                        ShutDown = mBlock.fStop = 1;
```

```
                        ReleaseMutex (mBlock.mGuard);
} else if (command[0] == _T('c')) { /* Get a new buffer to consume */
                        WaitForSingleObject (mBlock.mReady, INFINITE);
                        WaitForSingleObject (mBlock.mGuard, INFINITE);
                        __try {
                                if (!mBlock.fReady) _leave; /* Don't process a message twice */
                                /* Wait for the event indicating a message is ready */
                                MessageDisplay (&mBlock);
                                InterlockedIncrement(&mBlock.nCons);
                                mBlock.nLost = mBlock.mSequence - mBlock.nCons;
                        mBlock.fReady = 0; /* No new messages are ready */
                        }
                        __finally { ReleaseMutex (mBlock.mGuard); }
                } else {
                        _tprintf (_T("Illegal command. Try again.\n"));
                }
        }
        return 0;
}
void MessageFill (MSG_BLOCK *msgBlock)
{
/* Fill the message buffer, and include mChecksum and mTimestamp        */
        /* This function is called from the producer thread while it   */
```

```
        /* owns the message block mutex*/

        DWORD i;

        msgBlock->mChecksum = 0;
        for (i = 0; i < DATA_SIZE; i++) {
                msgBlock->mData[i] = rand();
                msgBlock->mChecksum ^= msgBlock->mData[i];
}
        msgBlock->mTimestamp = time(NULL);
        return;
}
void MessageDisplay (MSG_BLOCK *msgBlock)
{
        /* Display message buffer, mTimestamp, and validate mChecksum  */
        /* This function is called from the consumer thread while it */
        /* owns the message block mutex                                */
        DWORD i, tcheck = 0;
        TCHAR timeValue[26];
for (i = 0; i < DATA_SIZE; i++)
                tcheck ^= msgBlock->mData[i];
        _tctime_s (timeValue, 26, &(msgBlock->mTimestamp));
        _tprintf (_T("\nMessage number %d generated at: %s"),
```

```
            msgBlock->mSequence, timeValue);
        _tprintf (_T("First and last entries: %x %x\n"),
            msgBlock->mData[0], msgBlock->mData[DATA_SIZE-1]);
        if (tcheck == msgBlock->mChecksum)
            _tprintf (_T("GOOD ->mChecksum was validated.\n"));
        else
            _tprintf (_T("BAD  ->mChecksum failed. message was corrupted\n"));


        return;
}
```