CS609-System Programming

Final term Current Papers updated (Fall 2022)

Made by RAjpoOt

Contact us: Click here

1. **When any key is pressed from the keyboard, the scan code of that key will be sent at port _____.**

   A. Port 40
   B. Port 20
   C. Port 80
   D. Port 60

2. **It is _____ to refer to a mapping file multiple times while using pointers as data structure.**

   A. Useful
   B. Meaningless
   C. Necessary
   D. Optional

3. **If the function inside the DLL are updated, then older program attached to the DLL will not work properly. The statement refers to _____ versioning.**

   A. Strength
   B. Problem
   C. Advantage
   D. Caution

4. **A process have _____ ID(s) and _____ handle(s).**

   A. unique, unique
   B. unique, multiple
   C. multiple, multiple
   D. multiple, unique

5. **The executable image name of a process in CreateProcess()API is specified by _____ parameter(s).**

   A. lpApplicationName only
   B. lpCommandLine only
   C. Either lpApplicationName or lpCommandLine
   D. Neither lpApplicationName nor lpCommandLine

6. **Using ExitProcess() function, a process can be made exit or terminate. Which of the following parameter's is required to pass to the following execution successful?**

   A. Process's real handle
   B. Process's pseudo handle
   C. Process's Id
   D. Process ExitCode

7. **Which statement is incorrect when processes are made synchronized?**

   A. It can decrease the overhead of a system
   B. It can increase the overhead of system
   C. Same processes can be blocked as synchronization
   D. Mutual exclusion is achieved

8. **When all the threads of a process are made exit or terminate, then the process _____.**

   A. Will still survive
   B. Will terminate
   C. Will terminate for specified period of time
   D. Will temporarily suspends

9. **If you want to create 4 threads i.e., Thread 0, Thread 1, Thread 2 and Thread 3, they all must be created at _____ state.**

   A. Ready
   B. Suspended
   C. Running
   D. Blocked

10. **In Thread Local Storage (TLS) arrangement, row represents _____ while column represent _____.**

    A. Thread Number, TLS Index
    B. TLS Index, Thread Number
    C. Thread number, Process Number
    D. Process Number, TLS Index

11. **The use of global variable flag in the solution of critical section problem suffers or fails due to the _____ of threads.**

    A. Context switch
    B. Hold and wait condition
    C. Sleeping condition
    D. Deadlock condition

12. **The use of _____ ensures that the main memory is updated and cache of all processors is coherent.**

    A. Volatile qualifier
    B. Global flag
    C. Volatile qualifier and Global flag
    D. Memory barrier

13. **A thread can enter _____ time(s) into its critical section at different instances is thread synchronization.**

    A. Only one
    B. Several
    C. Exactly ten
    D. At suspended

14. **For mutual exclusion to work, _____ variable(s) must be protected by a single synchronization object across all threads.**

    A. Important
    B. local
    C. Shared
    D. Local integer

15. **Producer consumer problem is a classical problem in _____.**

    A. multi exclusion
    B. mutual process
    C. mutual exclusion
    D. object exclusion

16. **For the efficient use of critical section and mutexes, the thread priority should be decided by the _____.**

    A. Program
    B. OS Scheduler
    C. Mutex
    D. Critical Region

17. Which of the following synchronization construct is used for semaphore?

A. Enter
B. Leave
C. Wait
D. Set

18. Optimal spin-count ensures _____.

A. Less system calls
B. More system calls
C. More overheads
D. Critical Section overheads

19. A(n) _____ is a collection of worker thread that execute asynchronous callbacks on behalf of the application.

A. Address Space
B. Thread pool
C. Callback Functions
D. Environment Variables

20. By default, the _____ makes scheduling decisions and allocates processors to each thread.

A. System programmer
B. Kernel
C. BIOS
D. Loader

21. What is the purpose of a critical section in a program with a mutex?

A. To define a section of code that can only be executed by one thread at a time.
B. To allocate memory dynamically
C. To create a user interface
D. To calculate mathematical expressions

22. How can we determine the number of processors from the system affinity mark?

A. By counting the number of bits set
B. By calling the windows version
C. By using a simple program
D. By counting the number of word set

23. What does the system affinity mask indicate?

A. The active processor masks
B. The Windows version
C. The number of threads
D. The number of processors on the system

24. How much virtual address space is consumed by 1000 threads?

A. 100MB
B. 10GB
C. 1GB
D. 1TB

Note:

The amount of virtual address space consumed by 1000 threads depends on several factors, such as the platform, the operating system, and the application. Each thread in a process requires some amount of virtual address space to store its stack, registers, and other data structures. However, the amount of virtual address space used by a thread can vary depending on its stack size, the number of variables it uses, and the operating system's memory management policies.

Assuming a typical stack size of 1 MB per thread, the total virtual address space consumed by 1000 threads would be approximately 1 GB. However, this is just an estimate, and the actual amount of virtual address space used by the threads may be more or less depending on the factors mentioned above. Therefore, the correct answer would be option C. 1GB.

25. What is the default stack size for each thread in memory?

A. 100MB

B. 1GB
C. 1MB
D. 10MB

Note:

The default stack size for each thread in memory can vary depending on the operating system and other factors. In some systems, the default stack size may be 1MB, while in others it may be 8MB or more. Therefore, the correct answer would depend on the specific operating system and configuration being used.

26. What is the purpose of spin count in SRW locks?

A. To support recursion
B. To determine the size of the lock
C. To reduce the overhead of context switches
D. To allow configuration or adjustment

27. What is an interlocked function in C++ ?

A. A function that performs non-atomic operations
B. A function that performs atomic operations
C. A function that performs complex operations
D. A function that performs basic operations

28. How can you specify a manual-reset event?

A. By setting bMannualReset to FALSE
B. By setting bMannualReset to TRUE
C. By setting bInitialState to FALSE
D. BY setting OpenEvent to TRUE

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

A. VOID
B. LPVOID
C. HANDLE
D. BOOL

30. We provide _____ parameters to the WaitForSingleObject() API.

A. 0
B. 1
C. 2
D. 3

31. Each process must be have at least _____ thread.

A. 0
B. 1
C. 2
D. 3

32. _____ function is used to return the address of the currently running fiber.

A. GetCurrentFiber()
B. GetFiberData()
C. GetFiberId()
D. GetFiberIdentity()

33. _____ API opens an existing thread's handle from a thread ID.

A. GetThreadId()
B. GetCurrentThreadId()
C. GetCurrentThread()
D. OpenThread()

34. Thread affinity mask is a subset of _____.

A. Process Affinity Mask
B. Processor Affinity Mask
C. Thread Affinity Mask

D. Kernel Affinity Mask

35. If invalid file handle is passed as parameter to the CloseFile() API, then it will return _____.

A. True value
B. False value
C. A new handle
D. Close a handle

36. The exception handler is actually a code portion associated with _____ block.

A. _try
B. _catch
C. _except
D. _finally

37. If the file is not opened in concurrent mode, then ReadFile() API starts reading from the _____.

A. start of file
B. Current position
C. End of File
D. Backup file

38. Which function is used to obtain the actual handle of a process?

A. GetCurrentProcess()
B. OpenProcess()
C. CreateProcess()
D. GetCurrentProcessId()

39. The filter function _____ the type of an exception.

A. Identifies
B. Evaluates
C. Restricts
D. Excludes

40. Which option is incorrect when the TraverseDirectory() API is required to be use?

A. It allows non-recursive traversal
B. It allows recursive traversal
C. It allows both the non-recursive and recursive traversal
D. Recursive and non-recursive traversal option is irrelevant

41. The _____ function terminates the process if the program indicates the error is fatal

A. ReportError()
B. ReportHandle()
C. TerminateHandle()
D. TerminateProcess()

42. If binary functions inside DLL are accessed at load time, then called as _____.

A. Explicit linking
B. Implicit linking
C. Both Explicit and Implicit linking
D. Thread linking

43. In a GetProcessTime() API, the elapsed time of a process can be computed by subtracting the _____.

A. ExitTime from CreationTime parameter
B. CreationTime from ExitTime parameter

C. UserTime From ExitTime parameter
D. ExitTime from UserTime parameter

44. Which statement is incorrect when processes are made synchronized?

A. It can decrease the overhead of a system
B. It can increase the overhead of system
C. Some processes can be blocked in synchronization
D. Mutual exclusion is achieved

45. Which parameter is required to be pass to the ExitThread() API?

A. Thread ID
B. Thread Object name
C. Both thread ID and its object name
D. Thread ExitCode

46. How many number of time's parameters are provided to a GetProcessTime() API to compute the elapsed time of a process

A. 0
B. 1
C. 2
D. 4 correct

47. _____ API create thread handle from a thread ID

A. GetThreadId()
B. GetCurrentThreadId()
C. GetCurrentThread()
D. OpenThread()

48. When all the threads of a process are made exit or terminate, then the process _____

A. Will still survive

B. Will terminate Correct

C. Will terminate for specified period of time

D. Will temporarily suspends

49. A fiber can obtain its identify by calling _____ API.

A. GetCurrentFiber()

B. GetFiberData()

C. GetFiberId()

D. GetFiberIdentity()

50. Which of the following is not a thread model?

A. Boss worker

B. Client-server

C. Pipeline

D. Peer to peer

51. The producer also computes a simple _____ of the message in the Producer-Consumer problem.

A. Checkmul

B. checkdiv

C. checksub

D. checksum

52. Windows provides an object called a _____. Using this object, we can enforce mutual exclusion.

A. multex

B. multask

C. mutex

D. mutext

53. Mutexes have some advantages beyond CRITICAL_SECTION. Mutexes can be named and have _____.

A. sections
B. timers
C. handles
D. self-loops

54. To minimize the concurrency related issues, _____

A. Mutex is unlocked on entry and locked on exit
B. Size of the critical region should be minimized
C. Premature exits from the critical region are necessary
D. Critical Region must have more than one entry

55. Which of the following synchronization object is not a named object?

A. Critical Section
B. Mutex
C. Semaphore
D. Event

56. According to the MSDN the optimal spin count for heap related functions is

A. 3000
B. 4000
C. 5000
D. 6000

57. When CreateThreadpoolWorker() is called then _____ is (are) executed using the process thread pool.

A. Callback functions
B. Kernel functions
C. Kernel Routines
D. Work object code

58. By default, the _____ make scheduling decisions and allocates processors to each thread.

A. System Programmer
B. Kernel
C. BIOS
D. Loader

59. _____ can be used to assign a thread to a specific processor.

A. Memory Management Unit
B. Device Driver
C. Processor Affinity
D. Loader

60. What is range of clock rates for most system today?

A. 0-1GHz
B. 1-2GHz
C. 2-3GHz Correct (TOPIC 175)
D. 3-4GHZ

61. What is the size of an SRW Lock?

A. 32 bits & 16 bits
B. 64 bits & 8 bits
C. 8 bits & 16 bits
D. 32 bits & 64 bits Correct (TOPIC 163)

3 Marks Questions

1. **What are Based Pointers in Windows? Also write its syntax.**

Answer:

Based Pointers in Windows are a mechanism for accessing memory in segmented memory architectures. In this mechanism, a segment register is used to point to a segment of memory and an offset register is used to indicate the location within the segment.

The syntax for based pointers in Windows is as follows:

type_name __based(segment_name) *pointer_name;

Here, "type_name" is the data type of the variable being declared, "segment_name" is the name of the segment register to be used, and "pointer_name" is the name of the pointer variable being declared. The "__based" keyword is used to indicate that the pointer is a based pointer.

2. **How a DLL file is loaded inside a program during explicit linking? Mention the API that is used for this purpose?**

Answer:

During explicit linking, a DLL file is loaded into a program using the LoadLibrary() API. The steps to load a DLL file using explicit linking are:

1. Obtain a handle to the DLL file using the LoadLibrary() API.
2. Get the function address of the exported function(s) using the GetProcAddress() API.
3. Call the function(s) through the function pointer(s) obtained from GetProcAddress().

The syntax for loading a DLL file using LoadLibrary() is as follows:

```
HMODULE hModule = LoadLibrary("example.dll");
if (hModule != NULL) {
   // Get the function address using GetProcAddress()
   // Call the function through the function pointer obtained from GetProcAddress()
}
```

Here, "example.dll" is the name of the DLL file to be loaded. If the LoadLibrary() call is successful, it returns a handle to the loaded DLL file, which is stored in the variable "hModule". Once the handle is obtained, the functions exported by the DLL can be accessed using GetProcAddress() and called through the function pointers obtained from it.

3. **Write the names of functions that are used to determine the amount of time consumed by:**

    1. Process

    2. Thread

Answer:

1. GetProcessTimes() function is used to determine the amount of time consumed by a process.
2. GetThreadTimes() function is used to determine the amount of time consumed by a thread.

4. **Which Windows function is used to convert a thread into fiber? Write the general syntax of this function also.**

Answer:

The **"ConvertThreadToFiber"** function is used to convert a thread into a fiber in Windows.

Syntax:

```
LPVOID ConvertThreadToFiber(
  LPVOID lpParameter
);
```

This function takes a single parameter lpParameter which is an optional parameter used to pass data to the fiber. The function returns a fiber data object that represents the newly created fiber.

5. **Mention three principal reasons of slim reader/writer (SRW's) to provide improved locking performance compared to mutexes and Critical sections (CSs)?**

Answer:

The three principal reasons why Slim Reader/Writer (SRW) locks provide improved locking performance compared to mutexes and critical sections (CSs) are:

1. **Reduced context switching**, as SRWs allow for lock state changes without the need for system calls.

2. **Priority boosting**, which improves performance in high-contention scenarios by boosting the priority of the thread holding the lock.

3. **Reader bias**, where SRWs prioritize readers over writers, improving performance in scenarios with many readers.

6. **Mention the three values or flags of dwOpenMode parameter in CreateNamedPipe() API function?**

   Answer:

   The "CreateNamedPipe()" API function is used to create a named pipe and takes several parameters. One of these parameters is the "dwOpenMode" parameter, which specifies the pipe's access mode. The three values or flags of "dwOpenMode" parameter are:

   1. "PIPE_ACCESS_INBOUND:" This flag specifies that the pipe can be used for inbound communication only, meaning that the pipe server can only read from the pipe and the pipe client can only write to the pipe.

   2. "PIPE_ACCESS_OUTBOUND:" This flag specifies that the pipe can be used for outbound communication only, meaning that the pipe server can only write to the pipe and the pipe client can only read from the pipe.

   3. "PIPE_ACCESS_DUPLEX:" This flag specifies that the pipe can be used for bidirectional communication, meaning that both the pipe server and the pipe client can read from and write to the pipe.

7. **What is the name of WndowsAPI that is used to get and report error to the user?**

   Answer:

   The Windows API function that is commonly used to retrieve and report errors to the user is called "FormatMessage". This function is part of the Windows API error-handling mechanism, and it is used to retrieve the text description of an error message associated with a given error code.

8. **Write the general syntax of APIs that are used for:**
   1. **Allocating index in the Thread Local storage (TLS)**
   2. **Deallocating index from the Thread Local Storage (TLS)**

Answer:

1. **Allocating index in the Thread Local Storage (TLS):**

   The API used for allocating an index in the Thread Local Storage is "TlsAlloc". The general syntax of this API is as follows:

   ```
   DWORD TlsAlloc( void );
   ```

   The TlsAlloc function reserves a new index in the Thread Local Storage. If successful, it returns the newly allocated index. If unsuccessful, it returns the value TLS_OUT_OF_INDEXES.

2. **Deallocating index from the Thread Local Storage (TLS):**

   The API used for deallocating an index from the Thread Local Storage is "TlsFree". The general syntax of this API is as follows:

   ```
   BOOL TlsFree(
     DWORD dwTlsIndex
   );
   ```

   The TlsFree function releases an index in the Thread Local Storage that was previously allocated using TlsAlloc. The function takes one parameter, the index to be released (dwTlsIndex), which must be a valid index value obtained from a previous call to TlsAlloc. If the function succeeds, it returns a nonzero value. If it fails, it returns zero.

9. **Write names of three windows functions that are used to manage Mutexes?**

Answer:

1.CreateMutex
The CreateMutex function is used to create a new mutex object.
2.WaitForSingleObject
The WaitForSingleObject function is used to wait for ownership of a mutex object.
3.ReleaseMutex
The ReleaseMutex function is used to release ownership of a mutex object.

10. **In the CreateNamedPipe() API function the dwOpenMode specifies many flags. Name any three.**

Answer:

The CreateNamedPipe function is used to create a named pipe and it takes several parameters, including the dwOpenMode parameter which specifies the mode in which the pipe is opened. Here are three possible flags that can be used with the dwOpenMode parameter:

1. PIPE_ACCESS_DUPLEX

This flag specifies that the pipe can be used for both reading and writing operations. It allows two-way communication between the client and server applications.

2. PIPE_ACCESS_INBOUND

This flag specifies that the pipe can only be used for inbound operations, such as reading data from the pipe. It allows the server application to receive data from the client application.

3. PIPE_ACCESS_OUTBOUND

This flag specifies that the pipe can only be used for outbound operations, such as writing data to the pipe. It allows the client application to send data to the server application.

11. What is the functionalities of following API's

GetLastError()

FormatMessage()

LocalFree()

Answer:

The following APIs are related to error handling in Windows operating system:

GetLastError():

The GetLastError() function retrieves the last error that occurred during the calling thread's execution. This function is typically used when a Windows API function fails, and the caller wants to determine the cause of the failure. The error code returned by GetLastError() is a 32-bit value that is specific to the current thread.

FormatMessage():

The FormatMessage() function retrieves the message string for a given error code. It takes the error code and a set of flags as input, and returns a formatted message string that describes the error. The function can also replace any placeholders in the message string with corresponding values, such as the name of the calling process.

LocalFree():

The LocalFree() function frees memory that was previously allocated by the LocalAlloc() or LocalReAlloc() function. The LocalAlloc() function is used to allocate memory from the local heap, which is a per-process heap that is used for small memory allocations. When the memory is no longer needed, the LocalFree() function can be used to release it, thereby reducing the process's memory usage.

Together, these APIs can be used to effectively handle errors in Windows programs by retrieving error codes, formatting error messages, and freeing up memory that is no longer needed.

5 Marks Questions

1. **Answer the following Questions.**

   1) Mention the API that is used to destroy or close the job object.

   2) Can a process be a member of more than one job object?

   3) Is it incorrect that "a child process is a member of its parent's job object as a default"?

   Answer:

   1. The **"CloseHandle()"** API function is used to destroy or close a job object in Windows. This function can be used to close any kernel object, including a job object.

   2. Yes, a process can be a member of more than one job object. This is because a process can be created in a suspended state and then assigned to a job object before it starts running. In addition, a process can also be transferred from one job object to another during its lifetime.

   3. Yes, it is incorrect to say that "a child process is a member of its parent's job object as a default." By default, a child process is not a member of its parent's job object. However, a child process can be added to its parent's job object using the **"AssignProcessToJobObject()"** API function. This function allows the parent process to assign the child process to a specific job object, which can then impose limits and restrictions on the child process's resource usage.

2. **Write the return type and parameters of the function Create Semaphore ().**

   Answer:

The "CreateSemaphore()" function is used to create or open a semaphore object in Windows. The function returns a handle to the newly created semaphore object or to an existing one that has been opened.

The function prototype for "CreateSemaphore()" is as follows:

```
HANDLE CreateSemaphore(
  LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
  LONG                  lInitialCount,
  LONG                  lMaximumCount,
  LPCTSTR               lpName
);
```

The parameters of the "CreateSemaphore()" function are:

1. "lpSemaphoreAttributes" (Optional): A pointer to a **"SECURITY_ATTRIBUTES"** structure that determines whether the returned handle can be inherited by child processes. If **"lpSemaphoreAttributes"** is "NULL", the handle cannot be inherited.

2. "lInitialCount": The initial count of the semaphore object. This specifies the number of available resources that the semaphore represents when it is created. If this parameter is zero, the semaphore is created in a non-signaled state.

3. "lMaximumCount": The maximum count of the semaphore object. This specifies the maximum number of concurrent users of the semaphore. If this parameter is zero, the semaphore has a maximum count of 64K - 1.

4. "lpName" (Optional): The name of the semaphore object. This parameter can be "NULL" to create an unnamed semaphore object, or a string to create a named semaphore object. If a named semaphore object already exists with the same name, **"CreateSemaphore**()" opens the existing object instead of creating a new one.

3. Briefly describe the following parameters that are used with CreateProcess() API.

1. LPCSTR IpApplicatonName;

2. BOOL bInheriHandles

3. LPVOID lpEnvironment

4. LPCSTR lpCurrentDirectoty

Answer:

1. LPCSTR lpApplicationName: This parameter specifies the name of the application to be executed. It can be a path to an executable file or a pointer to a string that contains the name of the executable file. If this parameter is NULL, the function uses the first whitespace-separated token of the command line as the name of the application.

2. BOOL bInheritHandles: This parameter specifies whether the new process should inherit the handles of the calling process. If this parameter is TRUE, the new process inherits the handles; if it's FALSE, the new process does not inherit the handles.

3. LPVOID lpEnvironment: This parameter specifies the environment block for the new process. It can be a pointer to a block of environment variables or NULL. If it's NULL, the new process uses the environment of the calling process.

4. LPCSTR lpCurrentDirectory: This parameter specifies the current directory for the new process. It can be a pointer to a string that contains the name of the directory or NULL. If it's NULL, the new process uses the current directory of the calling process.

4. **What is the purpose of *Addend and Value parameter used in "InterlockedExchangeAdd()" API?**

Answer:

The **"InterlockedExchangeAdd()"** API is used for performing an atomic addition operation on a variable. The purpose of the two parameters, *Addend and Value, is as follows:

1. *Addend: This parameter is a pointer to the variable to which the value will be added. It is both an input and an output parameter. It represents the current value of the variable before the addition and also receives the new value after the addition.

2. Value: This parameter is the value to be added to the *Addend parameter. It can be a positive or negative value.

The **"InterlockedExchangeAdd()"** API performs the addition operation atomically, meaning that it guarantees that no other thread can access or modify the variable during the operation. The API returns the original value of the variable before the addition.

This API is typically used in multi-threaded applications where several threads may access the same variable simultaneously. By using the **"InterlockedExchangeAdd()"** API, you can ensure that the addition operation is performed atomically and that the value of the variable is consistent across all threads.

5. **Keeping in view the following syntax of DuplicatHandle() API, answer the questions BOOL duplicateHandle (HANDLE hSourceProcessHandle,HANDLEhSourceHandle,HANDLE hTargetProcessHandle,LPHANDLE IpTargetHandle,---)**

1.Mention the name of parameter that shows the original or real handle.

2.Mention the name of parameter that receives a copy of the original handle.

Answer:

1.The parameter that shows the original or real handle is "hSourceHandle". This is the handle to the object being duplicated, which can be a file, pipe, mutex, semaphore, thread, process, or other kernel object.

2. The parameter that receives a copy of the original handle is "IpTargetHandle". This is a pointer to the handle variable that will receive the copy of the handle. If the function succeeds, this parameter will contain a handle to the duplicated object. If the function fails, the value of this parameter will be NULL. Note that the handle returned in this parameter is a copy of the original handle and can be used independently of the original handle.

6. **The general syntax for API that is used to move a thread from its running state to waiting state is**

VOID Sleep (DWORD dwMilliseconds);

1. In above API, if the parameter dwMilliseconds is specified as INFINITE, then what will happen to a thread?

2.If the parameter dwMilliseconds is specified as 0, then what will happen to a thread?

Answer:

1. If the parameter dwMilliseconds is specified as INFINITE, the thread will remain in a waiting state indefinitely until it is signaled by another thread or process. This is useful in situations where the thread needs to wait for an event or resource to become available before continuing execution, but the exact amount of time it needs to wait is unknown.
2. If the parameter dwMilliseconds is specified as 0, the thread will enter a waiting state for a very short period of time, typically a few milliseconds. This is often referred to as a "yield" or "spin-wait" operation because the thread will quickly relinquish its time slice and allow other threads to run. If there are no other threads waiting to run, the operating system will immediately return control to the waiting thread. This can be useful in situations where the thread needs to perform some brief processing, but wants to avoid wasting CPU cycles by constantly polling for a resource that may not be available.

7. **Write the names of 4 functions used to create and manage semaphores?**

Answer:

Here are the names of four functions used to create and manage semaphores in Windows:

1. **CreateSemaphore**

This function is used to create a new semaphore object or open an existing one. It takes several parameters, including the initial count and maximum count of the semaphore, and returns a handle to the semaphore object.

2. **OpenSemaphore**

This function is used to open an existing semaphore object. It takes the name of the semaphore and the desired access rights as parameters, and returns a handle to the semaphore object.

3. **ReleaseSemaphore**

This function is used to release a semaphore, which increments its count by a specified amount. It takes the handle to the semaphore object, the release count, and a pointer to a variable that will receive the previous count as parameters.

4. WaitForSingleObjectEx

This function is used to wait for a semaphore to be signaled. It takes the handle to the semaphore object, the maximum wait time, and a flag that specifies whether to alert the thread when the wait is satisfied as parameters. It returns a status code indicating whether the wait was successful.

8. Consider the following code given below for using the Thread Pools:

int_tmain(DWORD argc.LPTSTR argv[]){

INT nThread:

HANDLE *pWork;bjects:

THARG *pThreadArg;

TP CALLBACK ENVIRON cbe;

nThread=_ttoi(argv[1]);

You need to write code to first allocate memory to pWorkObjects using malloc() then use CreateThreadpoolWork() api function to create thread pool worker object.

Considering the name of work call back is "Worker".

Answer:

There is an example code snippet that demonstrates how to allocate memory to "pWorkObjects" using "malloc()" and use the "CreateThreadpoolWork()" function to create thread pool worker objects:

```
int_tmain(DWORD argc, LPTSTR argv[]) {
```

```
INT nThread;
PTP_WORK *pWorkObjects;
THARG *pThreadArg;
TP_CALLBACK_ENVIRON cbe;
nThread = _ttoi(argv[1]);

// Allocate memory for the work objects
pWorkObjects = (PTP_WORK*)malloc(nThread * sizeof(PTP_WORK));
if (pWorkObjects == NULL) {
   // handle error
}

// Initialize the thread pool environment
InitializeThreadpoolEnvironment(&cbe);

// Set the work callback function
SetThreadpoolCallbackPool(&cbe, NULL);

// Create the thread pool worker objects
for (int i = 0; i < nThread; i++) {
   pWorkObjects[i] = CreateThreadpoolWork(Worker, NULL, &cbe);
   if (pWorkObjects[i] == NULL) {
      // handle error
   }
}

// Do some work with the thread pool worker objects...

// Clean up the thread pool environment
```

```
    DestroyThreadpoolEnvironment(&cbe);

    // Free the memory allocated for the work objects
    free(pWorkObjects);

    return 0;
}

// Worker callback function
VOID CALLBACK Worker(PTP_CALLBACK_INSTANCE instance, PVOID context, PTP_WORK work) {
    // Do some work...
}
```

Note that the Worker callback function is defined outside of the int_tmain function and is passed as a parameter to the CreateThreadpoolWork function when creating the thread pool worker objects. Also, the InitializeThreadpoolEnvironment and DestroyThreadpoolEnvironment functions are used to initialize and clean up the thread pool environment, respectively.