Virtual University	**CS609** **SYSTEM PROGRAMMING** **Update (Topics)** **Current Final Term Subjective 2023**	01/03/2023
100% Correct	**Solved by Faisal (J.c FIA)**	**VU**

Question# 1: What is Explicit Linking and which Windows API is used to load a DLL explicitly inside a program?

Answer: Explicit linking is a method of loading dynamic-link libraries (DLLs) in Windows programs. When a program uses explicit linking, it loads the DLL manually at runtime and explicitly calls the functions within the DLL that it needs.

The Windows API used to load a DLL explicitly inside a program is called "LoadLibrary". The LoadLibrary function loads the specified DLL into the address space of the calling process and returns a handle to the loaded module. Once a module is loaded, the program can use the GetProcAddress function to obtain the address of a function within the DLL, which can then be called like any other function.

Explicit linking is useful when a program only needs to use certain functions within a DLL and doesn't want to incur the overhead of loading the entire DLL into memory.

Question#2: Name the two Windows APIs that are used to return the pseudo and real handle of a process.

Answer: The two Windows APIs used to return the pseudo and real handle of a process are:

1.**GetCurrentProcess**(): This function returns a pseudo handle to the current process. A pseudo handle is a special constant that represents the current process and does not need to be closed.

2. OpenProcess(): This function opens a real handle to a specified process. It takes two parameters: the first parameter is the desired access to the process (such as read, write, or execute), and the second parameter is a Boolean value indicating whether the handle should be inheritable by child processes. The function returns a real handle to the specified process if successful, or NULL if an error occurred.

Question#3: Briefly describe the two parameters associated with the following Windows API. FARPROC GetProcAddress(HMODULE hModule, LPCSTR lpProcName).

Answer: The Windows API "GetProcAddress" is used to retrieve the address of an exported function or variable from a specified dynamic-link library (DLL). It takes two parameters:

HMODULE hModule: This parameter is a handle to the loaded module (DLL) that contains the function or variable to be retrieved. If this parameter is NULL, the function retrieves the address of the function or variable from the current process's executable file.

LPCSTR lpProcName: This parameter is a pointer to a null-terminated string that specifies the name of the function or variable to be retrieved. If this parameter is an ordinal value, rather than a string, it represents the numeric value that corresponds to the function's position in the DLL's export table.

Question#4: Briefly explain system mask and process mask.

Answer:

System mask and process mask refer to sets of processor affinity masks that determine which processors a system or a process is allowed to use.

System mask: The system mask is a bit mask that specifies which processors are available for use by the entire system. By default, all processors are included in the system mask. However, system administrators can modify the system mask to exclude certain processors from use.

Process mask: The process mask is a bit mask that specifies which processors a specific process is allowed to use. By default, a process is allowed to run on any processor in the system. However, the process mask can be set using the SetProcessAffinityMask function to restrict a process to a subset of the available processors.

Restricting the processor affinity of a process can have performance implications. On one hand, restricting a process to a subset of available processors can reduce contention for shared system resources such as memory and I/O. On the other hand, it can also result in decreased performance if the process is CPU-bound and the restricted set of processors is insufficient to handle its workload.

Question#5: Briefly describe each parameter in the following Widows API.

```
BOOL GetProcessTime(HANDLE hProcess,
                    LPFILETIME lpCreationTime,
                    LPFILETIME lpExitTime,
                    LPFILETIME lpKernelTime,
                    LPFILETIME lpUserTime );
```

Answer: The Windows API "GetProcessTimes" is used to retrieve timing information about a specified process. It takes five parameters:

HANDLE hProcess: This parameter is a handle to the process for which timing information is being retrieved. This handle must have the PROCESS_QUERY_INFORMATION or PROCESS_QUERY_LIMITED_INFORMATION access right.

LPFILETIME lpCreationTime: This parameter is a pointer to a FILETIME structure that receives the creation time of the process.

LPFILETIME lpExitTime: This parameter is a pointer to a FILETIME structure that receives the exit time of the process. If the process has not exited, this parameter receives a NULL value.

LPFILETIME lpKernelTime: This parameter is a pointer to a FILETIME structure that receives the amount of time that the process has spent executing in kernel mode.

LPFILETIME lpUserTime: This parameter is a pointer to a FILETIME structure that receives the amount of time that the process has spent executing in user mode.

Question#6: What is the purpose of dwPipeMode parameter in CreateNamedPipe() API and mention the names of its two commonly used values.

Answer: The dwPipeMode parameter in the CreateNamedPipe() API function specifies the mode of the named pipe being created. It determines how data can be transferred through the pipe and how the pipe behaves in different situations.

The dwPipeMode parameter can be set to one or more of the following flags:

PIPE_WAIT: This flag specifies that the pipe is in blocking mode, which means that ReadFile and WriteFile operations will block until data is available or can be written.

PIPE_NOWAIT: This flag specifies that the pipe is in non-blocking mode, which means that ReadFile and WriteFile operations will return immediately, even if there is no data available or the pipe is full.

PIPE_READMODE_BYTE: This flag specifies that the pipe operates in byte mode, which means that data is read and written in byte-sized blocks.
PIPE_READMODE_MESSAGE: This flag specifies that the pipe operates in message mode, which means that data is read and written in message-sized blocks.
PIPE_TYPE_BYTE: This flag specifies that the pipe is a byte-type pipe, which means that the data is transferred as a stream of bytes.
PIPE_TYPE_MESSAGE: This flag specifies that the pipe is a message-type pipe, which means that the data is transferred as a series of messages.

Question#7: Which Windows API is used to release the ownership of a mutex? Also write its signature.

Answer: The Windows API used to release the ownership of a mutex is called ReleaseMutex().
The signature of the ReleaseMutex() API is as follows:

```
BOOL ReleaseMutex(
HANDLE hMutex
);
```

HANDLE hMutex: This parameter is a handle to the mutex object. The calling thread must be the owner of the mutex. If the calling thread is not the owner of the mutex, the function will fail and return zero.

Question#8: Write the four function names of semaphores?

Answer:

1. CreateSemaphore()
2. CreateSemaphoreEx()
3. OpenSemaphore()
4. ReleaseSemaphore()

Question#9: **Write the types of Semaphores?**

Answer: **There are four types of Semaphores:**

1. Creating a semaphore
2. Deleting a semaphore
3. Waiting a semaphore
4. Releasing a semaphore

Question#10: This Answer is important for mcqs and short question?

Answer:

GetCurrentThreadId()
Obtains thread Id rather than the handle.

GetThreadId()
Obtains thread Id from thread handle.

GetCurrentThread()
Non-inheritable pseudohandle to the calling thread.

OpenThread()
Creates Thread handle from the thread Id.

Thread Identity
Thread Ids and handles can be obtained using functions quite similar to one used with processes.

Question#11: Name any thread party frameworks that are used for thread optimization.
Answer: OpenMP and cilk++

Question#12: **Synchronization mutex thread synchronization mutex thread model?**
Answer: Synchronization refers to the coordination of multiple threads or processes to ensure that they access shared resources in a mutually exclusive and coordinated manner. The **mutex model** is widely used in multithreaded programming to prevent race conditions and ensure data consistency in shared resources.

Question#12: Syntax allocate and deallocate function?
Answer: allocate and deallocate are functions used in programming languages to dynamically allocate and deallocate memory during runtime.

syntax for allocate and deallocate functions in C++
Allocate Function
pointer_variable = new data_type [size];
Deallocate Function
delete [] pointer_variable;

Question#13: Interlocked Functions?
Answer: The Interlocked functions are a set of functions provided by the Windows operating system to provide atomic operations on shared variables that are accessed concurrently by multiple threads.
Interlocked functions include
InterlockedIncreament()
InterlockedDecreament().
InterlockedExchange
InterlockedCompareExchange

Question#14: Types of callback functions?
Answer:
1. CreateThreadpoolTimer()
2. CreateThreadpoolIO
3. CreateThreadpoolWait()

Question#15: **Critical section?**
Answer: Critical section is the part of the code that can only executed by one thread at a time. Windows provides the CRITICAL_SECTION objects as a simple lock mechanism for solving critical section problem.

Question#15: When thread Object is create, the space stack memory?
Answer: When a thread is created, the operating system allocates memory to hold the thread's execution

context, which includes the thread's stack space. The stack is a portion of memory that is used to store local variables, function parameters, and return addresses for the functions called by the thread.

Question#16: **ASCII encoded or Unicode format.**

Answer: ASCII and Unicode are two different character encoding schemes used to represent text in computers. ASCII is the most common character-encoding format for text data in computers and on the internet. In standard ASCII-encoded data, there are unique values for 128 alphabetic, numeric or special additional characters.

Question#17: InterlockedExchangeAdd write the parameter or return type.

Answer:

```
Long InterlockedExchangeAdd(
LONG volatile *Addend
LONG Increment )
```

Question#18: Write the GetProcessAffinityMask parameters, API etc.

Answer: most important question.

```
BOOL GetProcessAffinityMask (
    HANDLE hProcess,
    LPDWORD lpProcessAffinityMask,
    LPDWORD lpSystemAffinityMask)
```

Question#19: SetThreadAffinityMask?

Answer: very important

```
DWORD_PTR SetThreadAffinityMask (
    HANDLE hThread,
    DWORD dwThreadAffinityMask)
```

Question#20: CreatePipe parameters Api.

Answer:

```
BOOL CreatePipe (
    PHANDLE phRead,
    PHANDLE phWrite,
    LPSECURITY_ATTRIBUTES lpsa,
    DWORD cbPipe)
```

Question#21: General syntax of Interlockedincrement and InterlockedDecrement.

Answer:

Long InterlockedIncrement(LONG volatile *Addend)
Long InterlockedDecrement(LONG volatile *Addend)

Question#22: Write the name of API that is used to remove the lock from the locked file area?

Answer: The API used to remove a lock from a locked file area depends on the operating system.
In C/C++, the **flock** function can be used to lock and unlock a file region on Unix-based systems.

Question#23: In what condition, the termination Handler is ignored by a process?

Answer: Some conditions under which the termination handler may be ignored by a process.
The process has installed a signal handler that overrides the default termination handler for the SIGTERM signal. **The process** is unable to execute the termination handler due to resource constraints, such as a lack of available memory or CPU time.