Operating Systems

Lesson 4

A lot of material and assignment #2

Please stay focused

Plan

- Windows Process Synchronization
  - Signaled State of Windows object
  - Process signaled State
  - Event object
  - Mutex object
  - Semaphore object
- Modified Beeper Sample
- HW Assignment #2
  - Description
  - Proposed structure
  - Hints

Process Synchronization

- Preemptive multitasking: OS decides when process will get its CPU time slot
- How do we synchronized between processes?
  - Process will run only after other process ended
  - Only single process will have an access to a system resource (e.g. file)

Windows Synchronization Objects

- Some Windows objects can be in **signaled state** (e.g. process object).
- Process can use a System Call to wait until object become signaled or timeout elapses.
- Windows provides special objects to more complex synchronization scenarios
- Blocking **WaitForSingleObject** (HANDLE, TIMEOUT) system call to wait until object will be in signaled state

Process object

- Become signaled when process has ended
- To wait for process to finish use: WaitForSingleObject(hProcess,..)
- Every process that waits on a handle to a signaled process will be alerted.

Event Object (manual reset)

- HANDLE hEvent=CreateEvent(NAME,..)
- Will open event if event with this name already exists
- To wait: WaitForSingleObject(hEvent,…)
- To signal: SetEvent(hEvent)
- Every process that waits on the event's handle will be alerted
- Example: Signal to other process when its input (e.g. file) is ready

Mutex Object

- HANDLE hMutex=CreateMutex(NAME,..)
- Same trick with a name (open if exist)
- Only one process waiting on a Mutex handle will wake up
- Mutex become un-signaled and owned by process
- ReleaseMutex(hMutex) system call to make it signaled again
- Usage: Guard shared resource (e.g. only one process can write to a log file)

Semaphore object

- “A mutex with a counter”
- CreateSemaphore(Name, Counter, MaxValue,…)
- Signaled when counter is >0
- ReleaseSemaphore(hSemaphore,delta…) will increase counter by delta
- Usage: Many processes but limited number of resources (e.g. 2 sound cards but 10 processes)
- Usage: Make sure that no more then “counter”(2) of processes are alive and using resources

Modified Beeper Sample

Assignment #2 (due in 2 weeks)

- Build a fibproc.exe utility
- "Parallel" calculator of Fibonacci number:
  - $A_n=A_{n-1}+A_{n-2}$; $A_0=A_1=1$
- Given n calculate A_n
- Recursion is naïve way of calculating Fibonacci number but we'll do it anyway
- Fibproc.exe will spawn new processes (fibproc.exe) if there are less then **10** fibproc.exe processes already running

HW #2: Concept of operation

- (A) Wait on a semaphore to obtain processing slot for specified timeout (command line parameter)
- (B) If successful then spawn process to calculate A_{n-1}, otherwise calculate recursively in-process
- Repeat A and B for A_{n-2}
- Never spawn a child process for n=1 or n=0

HW #2: Input/Output

- Input
  - Fibproc.exe 50 “c:\log.txt” 100
  - Fibonacci number, path to log file and time to wait before calculating in-process
- Output
  - Fibonacci number as return code
  - Log file entry
    - TIME <TAB>N<TAB> A_n<TAB>NumProc
  - TIME is unsigned result form GetTickCount()
  - NumProc(0-2) is number of child processes

HW #2:Suggested structure

- `DWORD` **`FibByRec`**`(DWORD dwNum)`
  - Calculate Fibonacci by simple recursion
- `DWORD` **`FibByProc`**`(DWORD dwNum, LPCTSTR log_path, DWORD dwWaitTime)`
  - Run child process and wait for it to return. Use its exit code as return value
- `DWORD` **`FibByProcOrRec`**`(DWORD dwNum, LPCTSTR log_path, DWORD dwWaitTime, DWORD* pCount)`
  - Wait on semaphore and run either FibByProc of FibByRec
  - Fall back on FibByRec if FibByProc failed
- `BOOL` **`ProtectedLogWrite`**`(LPCTSTR log_path, LPCSTR log_str)`
  - Write string to a file protected by Mutex

Assignment #2:Main function

- Parse command line
- Call FibByProcOrRec twice
- Prepare output string
- Call ProtectedLogWrite
- Retrun Fibonacci as exit code

- Total: ~130 lines of well-formatted code

Assignment #2: Hints

- Create process with console first and write debug output to a console
- Use GetModuleFileName to get path to current executable
- Use reasonable wait time while waiting for child process to end or mutex to be signaled (e.g. 30 second)

HW#2: System Calls to use

- CreateMutex/CreateSemaphore
- ReleaseMutex/ReleaseSemaphore
- WaitForSingleObject
- CreateProcess/GetExitCodeProcess
- CloseHandle (Mutex, Process, Semaphore)
- GetModuleFileName
- File and string functions