Operating Systems

Lesson 8

Plan

- Threads
- Threads in Windows
- Thread Usage
- Thread Synchronization
- Thread Beeper sample
- HW #4

Processes and Threads

- **Process**

  The virtual address space and control information necessary for the execution of a set of thread objects.

- **Thread**

  An entity within a process that Windows schedules for execution.

Threads

- "light-weight" processes
- Threads in a process share address space
  - Code
  - Heap
- Have private Program Counter (PC) and stack
- Execution Scheduled by OS
  - Preemptive multitasking

Threads in Windows

- CreateThread (…, ThreadFunction,…)
- An object =>has a handle
- A process always has "main thread" associated with it
  - Handle returned by CreateProcess
- Access to shared process resources has to be synchronized among threads

Thread Usage

- Multiple CPUs=>parallel computation
- Asynchronous device communication
  - Wait for slow device operation in one thread while continuing computation in another
- Asynchronous user interaction
  - Perform computation in one thread while reacting on user input in another thread

Thread Synchronization

- Thread had finished=>thread in signaled state.
- Mutex/Semaphore/Event
  - Unnamed objects are allowed (NULL name)
  - Threads access an object through shared handle
- New synchronization object
  - Critical section
  - For synchronization between threads of the same process only (Similar to mutex)

Threads beeper sample

HW #4

- Based on previous reader/writer assignment
- Create 3 DLLs
  - fifo_queue.dll
  - lifo_queue.dll
  - Progress.dll
- Usage
  - Reader.exe fifo_queue.dll
  - Writer.exe fifo_queue.dll
- Check references for HW#4 on the course's homepage

FIFO/LIFO queue implementations in DLLs

- Implement MMF-based fifo and lifo queues (including synchronization)
- Reader/Writer pair load appropriate dll according to command line (run-time binding)
- Both DLL's have same interfaces

```
typedef BOOL (*pfnCreateQueue)(SHARED_QUEUE*);
typedef BOOL (*pfnDeleteQueue)(SHARED_QUEUE*);
typedef BOOL (*pfnPushElement)(SHARED_QUEUE*,DWORD);
typedef BOOL (*pfnPopElement)(SHARED_QUEUE*,DWORD*);
```

SHARED_QUEUE

```
typedef struct
{
    DWORD m_dwCount;
    DWORD m_dwHead;
    DWORD m_dwTail;

} SHARED_QUEUE_HEADER;

typedef struct
{
    SHARED_QUEUE_HEADER*                m_pHeader;
    DWORD*                              m_pQueue;
    HANDLE                              m_hMutex;
    HANDLE                              m_hReadSem;
    HANDLE                              m_hWriteSem;
    HANDLE                              m_hMapFile;
    BYTE*                               m_pBuffer;

} SHARED_QUEUE;
```

Progress DLL

- Progress.dll is used in reader and writer through compile-time binding (using *.lib)
- Implements"progress indicator"
- Interfaces
  - StartProgress() start printing stars "*" on the console window every second
  - StopProgress() stop printing stars and print "\n" if any star was printed
- First star is printed after 1 second

HW#4 concept of operation

- Same input/output as previous assignment
  - Writer accept integers
  - Reader print integers at each new line and sleep for duration of integer value
  - Reader and Writer exit if zero received
- New feature
  - Print progress while waiting on queue operation
    - Reader prints stars while waiting for elements in queue (not while in sleep)
    - Writer prints stars while waiting for free space

Reader Code (see homepage)

```
typedef struct
{
   DWORD m_dwCount;
   DWORD m_dwHead;
   DWORD m_dwTail;
} SHARED_QUEUE_HEADER;

typedef struct
{
   SHARED_QUEUE_HEADER* m_pHeader;
   DWORD*                             m_pQueue;
   HANDLE                                       m_hMutex;
   HANDLE                                       m_hReadSem;
   HANDLE                                       m_hWriteSem;
   HANDLE                                       m_hMapFile;
   BYTE*                               m_pBuffer;
} SHARED_QUEUE;

typedef BOOL (*pfnCreateQueue)(SHARED_QUEUE*);
typedef BOOL (*pfnDeleteQueue)(SHARED_QUEUE*);
typedef BOOL (*pfnPushElement)(SHARED_QUEUE*,DWORD);
typedef BOOL (*pfnPopElement)(SHARED_QUEUE*,DWORD*);
```

Readers Code-Con’t

```
typedef struct
{
   pfnCreateQueue CreateQueue;
   pfnDeleteQueue DeleteQueue;
   pfnPushElement PushElement;
   pfnPopElement  PopElement;
   HMODULE        m_hDLL;
} QUEUE_LIB;

BOOL LoadQueueLibrary(QUEUE_LIB* queue_lib,LPCTSTR dllName)
{
   ...
}

//declarations for compile-time DLL binding
BOOL __declspec(dllimport) StartProgress();
BOOL __declspec(dllimport) StopProgress();
```

Readers Code-Main

```
int _tmain(int argc, _TCHAR* argv[])
{
   assert(argc==2);
   QUEUE_LIB queue_lib;
   LoadQueueLibrary(&queue_lib,argv[1]);
   SHARED_QUEUE queue;
   queue_lib.CreateQueue(&queue);
   do
   {
         DWORD dwElem;
         StartProgress();//start showning "star progress“
         queue_lib.PopElement(&queue,&dwElem);
         StopProgress(); //end "star progress“
         _tprintf(_T("%d\n"),dwElem);
         ::Sleep(dwElem);
         if(!dwElem)
               break;
   }while(1);
   queue_lib.DeleteQueue(&queue);
   FreeLibrary(queue_lib.m_hDLL);
   return 0;
}
```

HW#4 Hints

- FIFO/LIFO DLL’s still have shared code
- Can put in shared mmf_queue.dll (compile-time binding to FIFO/LIFO DLL”s)- **Optional**
- Reader/Writer have shared code/declarations
  - Shared *.h files with DLL’s
  - Move some code to progress.dll (**optional**)