LSN 9
Windows Concurrency Mechanisms
ECT362 Operating Systems
LSN 9 – Thread Synchronization

- Several methods exist including the mutex and semaphore
- The thread synchronizes using the *wait* function
  - Thread is blocked until the internal state of the synchronization object determines that it is okay for the calling thread to proceed
  - Most common wait object
    ```
    DWORD WaitForSingleObject( HANDLE hHandle,
                               DWORD dwMilliseconds );
    ```
  - Result of wait found using
    ```
    GetLastError()
    ```
LSN 9 – Mutex Objects

• Can have an owner thread or be unowned
• Thread can gain ownership at object creation, when a handle to it is opened, or by a wait function

    HANDLE CreateMutex( LPSECURITY_ATTRIBUTE lpMutexAttrib,
                         BOOL bInitialOwner,
                         LPCTSTR lpName );

• Result of mutex creation

    GetLastError()

• Any thread in calling thread’s process can use the mutex
LSN 9 – Mutex Objects

- Mutex is released when thread is finished using it
  
  ```
  ReleaseMutex()
  ```

- Example:
  - Threads X and Y share a resource R
    ```
    int main()
    {
        ...
        open resource R
        ...
        //create the mutex object with no
        // ownership (signaled)
        mutexR = CreateMutex(NULL, FALSE, NULL);
        ...
        CreateThread( ... , workerThrd, ... ); //thread X
        CreateThread( ... , workerThrd, ... ); //thread Y
        ...
    }
    ```
DWORD WINAPI workerThrd( LPVOID )
{
    ...
    //perform work
    ...
    //obtain mutex
    while( WaitForSingleObject(mutexR, 0) != WAIT_OBJECT_O );
    //access the resource R
    releaseMutex( mutexR );
    ...
}
LSN 9 – Semaphore Objects

• Able to maintain a count to represent integer values

```c
HANDLE CreateSemaphore( LPSECURITY ATTRIBUTE lSemAttrib,
                         LONG iInitialCount,
                         LONG lMaximumCount,
                         LPCTSTR lpName );
```

• Semaphore object contains an internal variable that can be set between 0 and \( lMaximumCount \)
  – If the local variable equals 0 and a process calls a wait functions on the semaphore, the process will be blocked
  – If \( 1 \leq \text{local var} \leq lMaximumCount \), a wait function will decrement the value
LSN 9 – Semaphore Objects

- Semaphore is released when thread is finished using it

    ReleaseSemaphore( HANDLE hSemaphore,
                      LONG lReleaseCount,
                      LPLONG lPreviousCount )
LSN 9 – Semaphore Objects

• Example:
  – Threads X and Y are both using units of resource R
    • Either may request K units, use them for a period of time, then return them

```c
int main()
{
    ...
    //this is a controlling thread
    ...
    //create the semaphore object
    semaphoreR = CreateSemaphore( NULL, 0, N, NULL );

    ...
    CreateThread( ... , workerThrd, ... ); //thread X
    CreateThread( ... , workerThrd, ... ); //thread Y
    ...
}
```
DWORD WINAPI workerThrd( LPVOID )
{

    //perform work

    ... //acquire K units of the resource
    for( i=0 ; i<k ; i++ )
    {
        while( WaitForSingleObject(semaphoreR, 0) != WAIT_OBJECT_0 );
    }

    //perform work

    ...

    //release the K units
    ReleaseSemaphore( semaphoreR, K, NULL );

    ...
}
LSN 9 – Critical Section Objects

- Used by all threads of a common process to provides for mutual exclusion synchronization of a critical section
- No guarantee about the order in which threads will obtain ownership of the critical section
- Declaring a critical section

```
CRITICAL_SECTION criticalSectionObject;
```
- Critical section object must be initialized prior to its use

```
InitializeCriticalSection( CRITICAL_SECTION * );
```
LSN 9 – Critical Section Objects

• Each thread must request ownership of the critical section
  
  \texttt{EnterCriticalSection( CRITICAL\_SECTION * );}

• Each thread upon leaving the critical section must relinquish ownership of it
  
  \texttt{LeaveCriticalSection( CRITICAL\_SECTION * );}

• When the program terminates, the critical section must be deleted
  
  \texttt{DeleteCriticalSection( CRITICAL\_SECTION * );}
LSN 9 – Homework

• Reading
  – Chapter 6.10

• Assignment – HW 5
  – Complete HMWRK5.pdf

• References
  – Help within MSVS for specific function syntax descriptions