ECE329 HW #4

Part 1: Implement the following classes:

- A class called `Mutex` with the following interface:
```
Mutex();          // creates mutex
~Mutex();         // destroys mutex
void Lock();      // locks mutex (waits infinite)
void Unlock();    // unlocks mutex
```
- A class called `Semaphore` with the following interface:
```
Semaphore();      // creates semaphore
~Semaphore();     // destroys semaphore
bool Wait( int timeout );  // waits until semaphore is signaled
                           // timeout is in milliseconds,
                           //   ... (negative value means infinite)
                           // returns true if signaled, false if timed out
void Signal();    // signals the semaphore
```
- A base class called `Thread` with the following interface:
```
Thread();         // creates and runs thread
~Thread();        // destroys thread (after terminating it if still running)
Semaphore* GetExitSemaphore();  // returns a pointer to the semaphore
                                //   ... that is signaled when the thread
                                //   ... exits its main routine
```
  The class should have
  - a protected pure virtual method called `MainRoutine` that is called when the thread starts:
    `virtual DWORD MainRoutine() = 0;`
  - a static method called `StaticThreadProc` that calls `MainRoutine`:
    `static DWORD WINAPI StaticThreadProc( void* param );`
- A class called `ByteArray` with the following interface:
```
ByteArray( int n );                  // allocates array of n bytes
~ByteArray();                        // frees array
unsigned char& operator[]( int i );  // returns the ith byte
int size() const;                    // returns the number of bytes
                                     //   ... in the array
```

To implement these classes, you will need some of the following functions from the Win32 API:

- `CreateThread, CreateMutex, CreateSemaphore` -- creates a thread, mutex, or semaphore
- `TerminateThread` -- kills a thread
- `SetThreadPriority` -- changes the priority of a thread
- `WaitForSingleObject` -- locks a mutex; releases a semaphore
- `ReleaseMutex` -- unlocks a mutex

- `ReleaseSemaphore` -- signals a semaphore
- `CloseHandle` -- cleans up an object (e.g., thread, mutex, or semaphore)

Search the MSDN on-line help (http://msdn.microsoft.com/library/) for more specific information. (Be sure to click on the documentation for the function, not for the class method).

Part 2: Create a console-based application that compiles under Visual C++ 6.0 that spawns two threads: a producer and a consumer. The producer should open an input file and repeatedly copy values to a circular buffer. The consumer should open an output file and repeatedly copy values from the same circular buffer to the file. For both the producer and consumer, a random number of bytes between 1 and `n`, inclusive, should be copied each iteration, where `n` is specified by the user. The number of bytes should be randomized each iteration. If the producer is unable to write to the buffer (because it does not contain enough empty elements), or if the consumer is unable to read from the buffer (because it does not contain enough unread items), then it should proceed to the next iteration, choosing a new random number of bytes to copy. (Exception: Once the producer has already read the entire file, then the consumer does not have to continue generating random numbers until it gets an exact hit; instead, it should write the rest of the buffer to complete the copy.) When the program completes, the output file should be an exact copy of the input file.

The syntax of the program should be as follows:

```
copyfile input output n m
```

where

`copyfile` is the name of your executable
`input` is the name of the input file
`output` is the name of the output file
`n` is the maximum number of bytes to copy in any given iteration (described above)
`m` is the size of the circular buffer, in bytes

Use your C++ classes to implement this assignment. You may make slight modifications to the interfaces above (e.g., adding parameters to methods, adding methods, etc.), but such changes should be kept to a minimium. Your code should derive from `Thread` two classes, called `ProducerThread` and `ConsumerThread`, which provide the necessary implementations of `MainRoutine()`.

Separately, answer the following problems in Chapter 2 of the Tanenbaum textbook: 35, 36, 39, 40, 43, 44, 45.