**Comp 4735 Winter 2015**

## Lab Instructor: Mirela Gutica SET : 4D

**Name: Jens Christiansen & Leon Ho Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Lab 3

Solve the following exercises. Work in pairs. Discuss each exercise with your lab instructor.

1. Discuss Figure 4.1. What is the difference between the four models?

**Top left:** Each process has only one thread of execution. MS DOS is an example of such a configuration.

**Top Right:** Each process has multiple threads of execution. This is an example of a multithreaded execution.

**Bottom left:** The system supports more than one process but can only handle one thread per process.

**Bottom Right:** The system supports more than one process and can handle multiple threads of execution per process. Most modern operating systems utilize this configuration.

1. Discuss Figure 4.2. What is the difference between processes and threads?

A process must have a process control block and user address space, whereas in a multithreaded process model, the thread control blocks share the same process control block and user address space.

1. Discuss Figure 4.6.
   1. Trace the execution of Process B.
      * 1. In picture A, process B is currently running on the CPU. The threading library has chosen that Thread 2 should be running at the current time slice, and that thread 1 is in a ready state. The kernel does not know that the threads exist.
        2. In picture B, the process is not running as the kernel has put it in the block state. In the image, thread 2 is still in the state “running” however it is actually not being physically executed by the CPU.
        3. In picture C, the process is recognized to be in the ready state by the dispatcher, however the thread 2 is still marked as “running” as determined by the threading library. In reality, thread 2 is not actually running on the CPU.
        4. In picture d, the dispatcher has stared process b again, and the threading library has decided to switch threads to start running thread 1, and block thread 2. In this picture thread 1 is physically running on the processor as process B is currently running.
   2. Why are Thread 2 in the running state and Process B Blocked??

Thread 2 is in a running state because the OS does not know about user level threads.

1. What is the difference between ULT and KLT?

ULT, or User Level Thread, is a thread created by the user. It is controlled by the Thread Library in the application, and the kernel and OS does not know about it. Sometimes, however, the ULT can be mapped to a kernel-level thread. The KLT, or Kernel Level Thread, is a thread created and controlled by the kernel in the OS.

1. What are the advantages and disadvantages of each model?

The advantages of the User Level Threads are that they are much faster at switching between user level threads. A disadvantage is that the OS doesn’t know about the threads, and therefore the dispatcher cannot coordinate the threads across multiple cores and processors.

The advantages of Kernel Level Threads are that, because the OS knows, and indeed controls the threads, it can coordinate the threads across multiple cores and processors. One disadvantage of KLTs are that they take longer to switch between with the dispatcher.

1. Solve problems: 4.4, 4.7, 4.8

4.4) This model can make multithreaded programs run faster than their single-threaded counterparts because, while one thread issues a system call, like an I/O request, for example, the OS can switch to another thread and continue processing the non-blocked thread concurrently.

4.7) a) The function **count\_positives(list l)** counts the number of positive values in a list. When thread A and B perform their respective actions, the variable global\_positives will become 1. This is because the list passed into the function count\_positives has no positive values, and therefore the variable global\_positives will not be incremented at all.

b) This could create potential problems, because both of them are accessing global\_positives at the same time,