Eddie Gurnee
COSC 592
Operating Systems

Chapter 4 Reading Questions:

4.2 What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

Response:

1. User-level threads are supported above the kernel and and are managed without kernel support, whereas kernel threads are supported and managed directly by the operating system.
2. Regardless of the model (many-to-many, many-to-one, one-to-one), each user thread needs to belong to a kernel process, while kernel threads do not necessesarily need to belong to a process.

User threads are generally much easier to maintain than kernel-level threads; user threads are less expensive to create and use, in many-to-x models multiple user-level threads can be controlled by a fewer number of kernel-level threads, and in a multiprocessor enviroment kernel threads can schedule threads to run on different processors.

4.8 Describe the actions taken by a thread library to context-switch between user-level threads.

Response: Context-switching between user-level threads (like context-switching that occurs anywhere) requires the saving of the state of the registers data when switching away from a thread, and restoring it by loading the other state. The actions taken by the thread library to save such a state is dependant on their specific implementations, however often this is achieved through upcalls, handled by the aptly named upcall handler which run on virtual processors (as lightweight processes), which takes the lightweight process of the currently running thread and replaces it with another thread.

4.10 Which of the following components of program state are shared across threads in a multithreaded process?

a) Register values
b) Heap memory
c) Global variables
d) Stack memory

Response: Across threads in a multithreaded process, the heap memory and global variable components of program state are shared, while each thread has its own register values and stack memory. In Java threads however, there are no global variables of course, so references to different locations must be initialized in various threads.

4.11 Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system? Explain.

Response: A multithreaded solution using multiple user-level threads cannot achieve better performance on a multiprocessor system than on a single-processor system because the multiple threads cannot use the additional processors as the operating system views the program as running as one process, not the multiple threads among the program.