Threads

Practice Exercises

4.1 Provide two programming examples in which multithreading provides better performance than a single-threaded solution.

Answer: (1) A Web server that services each request in a separate thread. 2) (A parallelized application such as matrix multiplication where (different parts of the matrix may be worked on in parallel. (3) An (interactive GUI program such as a debugger where a thread is used (to monitor user input, another thread represents the running (application, and a third thread monitors performance.

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

Answer: (1) User-level threads are unknown by the kernel, whereas the kernel is aware of kernel threads. (2) On systems using either M:1 or M:N mapping, user threads are scheduled by the thread library and the kernel schedules kernel threads. (3) Kernel threads need not be associated with a process whereas every user thread belongs to a process. Kernel threads are generally more expensive to maintain than user threads as they must be represented with a kernel data structure.

4.3 Describe the actions taken by a kernel to context-switch between kernel-level threads.

Answer: Context switching between kernel threads typically requires saving the value of the CPU registers from the thread being switched out and restoring the CPU registers of the new thread being scheduled.

4.4 What resources are used when a thread is created? How do they differ from those used when a process is created?

Answer: Because a thread is smaller than a process, thread creation typically uses fewer resources than process creation. Creating a process requires allocating a process control block (PCB), a rather large data structure. The PCB includes a memory map, list of open files, and environment variables. Allocating and managing the memory map is typically the most time-consuming activity. Creating either a user or

kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

4.5   Assume that an operating system maps user-level threads to the kernel using the many-to-many model and that the mapping is done through LWPs. Furthermore, the system allows developers to create real-time threads for use in real-time systems. Is it necessary to bind a real-time thread to an LWP? Explain.

 Answer: Yes. Timing is crucial to real-time applications. If a thread is marked as real-time but is not bound to an LWP, the thread may have to wait to be attached to an LWP before running. Consider if a real-time thread is running (is attached to an LWP) and then proceeds to block (i.e. must perform I/O, has been preempted by a higher-priority real-time thread, is waiting for a mutual exclusion lock, etc.) While the real-time thread is blocked, the LWP it was attached to has been assigned to another thread. When the real-time thread has been scheduled to run again, it must first wait to be attached to an LWP. By binding an LWP to a real-time thread you are ensuring the thread will be able to run with minimal delay once it is scheduled.

4.6   A Pthread program that performs the summation function was provided in Section 4.3.1. Rewrite this program in Java.

 Answer: Please refer to the supporting Web site for source code solution.