1. Transition from blocked to running can happen if the process is blocked on I/O and the I/O finishes. If the CPU is otherwise idle, the process could go directly from blocked to running. The other missing transition, from ready to blocked, is impossible. A ready process cannot do I/O or anything else that might block it. Only a running process can block.

3. Assembly allows accessing to CPU hardware that is required by Operating system components.

4. The operating system may crash because a poorly written user program does not allow for enough stack space. Also, if the kernel leaves stack data in a user program’s memory space upon return from a system call, a malicious user might be able to use this data to find out information about other processes.

6. When a thread is stopped, it has values in the registers. They must be saved, just as when the process is stopped the registers must be saved. Timesharing threads isn’t different than timesharing processes, so each thread needs its own register save area.

7. No. If a single-threaded process is blocked on the keyboard, it cannot fork.

8. A thread will block when it has to read a web page from the disk. If user-level threads are being used, this action will block the entire process, destroying the value of multithreading. So, it is essential that kernel threads are used to permit some threads to block without affecting the others.

9. Threads in a process cooperate. They are not hostile to one another. If yielding is needed for the good of the application, then a thread will yield. After all, it is usually the same programmer who writes the code for all of them.

10. User-level threads cannot be preempted by the clock unless the whole process quantum has been used up. Kernel-level threads can be preempted individually. In the latter case, if a thread runs too long, the clock will interrupt the current process and thus the current thread. The kernel is free to pick a different thread from the same process to run next if it so desires.

11. In the single-threaded case, the cache hits take 15 msec and cache misses take 90 msec. The weighted average is 2/3 × 15 + 1/3 × 90. Thus the mean request takes 40 msec and the server can do 25 per second. For a multithreaded server, all the waiting for the disk is overlapped, so every request takes 15 msec, and the server can handle 66 2/3 requests per second.

14. The biggest advantage is the efficiency. No traps to the kernel are needed to switch threads. The biggest disadvantage is that if one thread blocks, the entire process blocks.

15. Yes it is possible, but inefficient. A thread wanting to do a system call first sets an alarm timer, then does the call. If the call blocks, the timer returns control to the threads package.

21. Each thread calls procedures on its own, so it must have its own stack for the local variables, return addresses, and so on. This is equally true for user-level threads as for kernel-level threads.