**Chapter 4 Review**

1. Thread Execution
   1. States:
      1. Running
      2. Ready
      3. Blocked
   2. Operations:
      1. Spawn – IP and arguments, own register context and stack
      2. Block – thread needs to wait for an event
      3. Unblock – when thread is moved back to ready queue
      4. Finish – reg context and stacks are dealloc’d
   3. Sync
      1. All threads of a process share the same address space and resources
      2. An alteration of one threads affects the other threads
2. Types of Threads
   1. User Level Thread (ULT)
      1. Basic:
         1. All thread management is done by the application
         2. Kernel don’t know about these
      2. Advantages of UTLs
         1. Thread switching does not require kernel mode priveleges
            1. All in user address space
            2. Doesn’t have to do a mode switch
         2. Scheduling can be application specific
            1. Can be round-robin or priority-based or whatnot
            2. Can be tailored to the application
         3. ULTs can run on any OS, need no change to kernel
            1. Threads library is shared by all apps
      3. Disadvantages
         1. In an OS many system calls are blocking
            1. When a ULT executes a system call not only is that thread blocked, but all of the threads within that process are blocked
         2. In a pure ULT a multithreaded app cannot take advantage of multiprocessing
      4. Overcoming Disadvantages
         1. Jacketing
            1. Converts a blocking system call into a non-blocking system call
         2. Writing an app as multiple processes rather than multiple threads
            1. Obviously this is dumb and kinda misses the whole point, because then it’s a process context switch, not a thread
   2. Kernel-Level Threads (KLTs)
      1. Basic:
         1. Thread management is done by kernel
         2. No thread management by app
      2. Advantages
         1. Kernel can simultaneously schedule multiple threads from the same process on multiple processors
         2. If one thread is blocked, kernel can switch to another thread in that process
         3. Can do multithreading
      3. Disadvantage
         1. Transfer of control from one thread to another within a process requires a mode switch to the kernel, which is expensive
   3. Combined Approaches
      1. Basic:
         1. Thread creation is done in the user space
         2. Bulk of scheduling and synch is done by app
   4. Applications That Benefit
      1. Multithreaded native applications
         1. Small number of highly threaded processes
      2. Multiprocess applications
      3. Java Apps
         1. Java is naturally conducive to multithreading
      4. Multiinstance apps
3. Specific OS stuff?