Chapter 4 Questions

Questions may be asked in class, on quizzes, and on exams

* + Explain how a thread is different than a process.

Threads (of the same process) run in a shared memory space, while processes run in separate memory spaces.

Threads are used for small tasks light-weight, where as processes are used for more ‘heavy-weight’ tasks – basically the execution of applications.

Threads within the same process share the same address space, whereas different processes do not. This allows threads to read from and write to the same data structures and variables, and also facilitates communication between threads.

A process was an executing program with a single thread of control

A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter (PC), a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signal

* + Explain why a programmer might use a thread instead of a process.

Complex application may actually look to the operating system like several different programs, with almost continuous switching and communication among them.

The more processes running, the greater the percentage of time the CPU and operating system will spend doing expensive context switches.

With enough processes to run, a server might eventually spend almost all of its time switching among processes and never do any real work.

A thread is like a child process, except all the threads associated with a given process share the same address space.It takes much less CPU time to switch among threads than between processes, because there's no need to switch address spaces.

* + Identify the four primary benefits of using multiple threads in a program.

Responsiveness,Resource sharing,Economy,Scalability.

* + Explain what task parallelism is, and provide an example that uses task parallelism.

Task parallelism involves distributing not data but tasks (threads) across multiple computing

cores. Each thread is performing a unique operation. Different threads may be operating on the same data,or they may be operating on different data

an example of task parallelism might involve two threads, each performing a unique statistical operation on the array of elements. The threads again are operating in parallel on separate computing cores, but each is performing a unique operation.

* + Explain what data parallelism is, and provide an example that uses data parallelism.

Data parallelism focuses on distributing subsets of the same data across multiple computing cores and performing the same operation on each core

Consider, for example, summing the contents of an array of size N. On a single-core system, one thread would simply sum the elements [0] ...[N −1]. On a dual-core system, however, thread A, running on core 0, could sum the elements[0]...[N∕2 − 1] while thread B, running on core 1, could sum the elements [N∕2]...[N − 1]. The two threads would be running in parallel on separate computing cores.

* + Explain how a many-to-one threading model compares to a one-to-one threading model.

Many user-level threads mapped to single kernel thread.  
One thread blocking causes all to block.  
Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time.  
Few systems currently use this model.

Each user-level thread maps to kernel thread.

Creating a user-level thread creates a kernel thread.  
More concurrency than many-to-one.  
Number of threads per process sometimes restricted due to overhead.

* + Explain how a one-to-one threading model compares to a many-to-many threading model.

Many-to-many:Allows many user level threads to be mapped to many kernel threads.

Allows the operating system to create a sufficient number of kernel threads.

* + Explain the difference between asynchronous and synchronous threading.

**Synchronous** basically means that you can only execute one thing at a time.

**Asynchronous** means that you can execute multiple things at a time and you don't have to finish executing the current thing in order to move on to next one

* + Explain what the difference in resource usage between a thread and a process; provide an example.

Process has its own block hold lot of data handling dynamic data have it internal variables process handle all the program.

Thread just handle the specific part.

* + Explain the difference between a kernel-level thread and a user-level thread; provide an example.

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 where as every user thread belongs to a process

* + End of Chapter 4
    - Exercises: 4.1, 4.3, 4.4, 4.5, 4.6, 4.8, 4.10, 4.12, 4.13, 4.15, 4.17, 4.19
    - Projects: 4.28 (look this problem over and think about how you would solve it)