Quiz 6 Notes

Chapter 29: Threads Intro & Tutorial link

* Treads: mechanisms that permit an application to perform multiple tasks concurrently.
  + An independent stream of instructions that can be scheduled to run as such by the operating system.
* Pthreads = POSIX threads
* Why Pthreads: In the world of high performance computing, the primary motivation for using Pthreads is to realize potential program performance gains.
* pthread\_t = Thread identifier
* In traditional UNIX API, errno is a global int. This doesn’t suffice for threaded programs. If a thread made a function call that returned an error in a global errno variable, then this would confuse other threads that might also be making function calls and checking errno.
* All pthreads functions return 0 on success or a positive value on failure.
* On Linux, programs that use the Pthreads API must be compiled with the cc –pthread option.
* When a program is started, the resulting process consists of a single thread, called the initial or main thread.
  + int pthread\_create(pthread\_t \*thread, consta….) – creates a new thread
  + Returns 0 on success, or positive error number on error.
* pthread\_exit(void \*retval) – terminates the calling thread, and specifies a return value that can be obtained in antoehr thread by calling pthread\_join().
* Each thread within a process is uniquely identified by a thread ID.
  + ID returned to called of pthread\_create()
  + A thread can obtain its own ID using pthread\_self().
* pthread\_equal() checks whether two threads are the same.
  + if (pthread\_equal(tid, pthread\_self()), printf(“tid matches self\n”);
* Joining with a terminated thread: pthread\_join(thread, returnval)
  + Returns 0 on success, positive errno on error
  + Waits for thread identified by thread to terminate. This is “joining.”
* If we want system to automatically clean up and remove thread when it terminates, mark thread as *detached.*
  + pthread\_detach(thread)
* Threads Vs. Processes
  + Sharing data between threads is easy. Not as easy as processes.
  + Thread creation is faster than process creation.
  + Need thread-safe functions while using threads
  + A bug in one thread can ruin other threads.
  + Each thread is competing for use of finite virtual address space of the host process.
* Parallel Programming: On modern, multi-cpu machines, pthreads are ideally suited for parallel programming, and whatever applies to parallel programming in general, applies to parallel pthreads programs.
* All threads have access to the same global, shared memory
* Threads also have their own private data
* Programmers are responsible for synchronizing access globally shared data.
* Thread-safeness: Refers an application’s ability to execute multiple threads simultaneously without “clobbering” shared data or creating “race” conditions.
* A program that runs fine on one platform, may fail or produce wrong results on another platform.
* The POSIX standards for Pthreads is still changing and evolving.
* Compiler/Platform for Intel Linux command C: icc –pthread
* Creating threads: Initially, your main() program comprises a single, default thread. All other threads must be explicitly created by the programmer.
* By default, a thread is created with certain attributes. Some of these attributes can be changed by the programmer via the tread attribute object.
  + Question: After a thread has been created, how do you know: a. When it will be scheduled to run by the operating system b. Which processor/core it will run on.
    - ANSWER: Unless you are using the Pthreads scheduling mechanism, it is up to the implementation and/or operating system to decide where and when threads will execute. Robust programs should not depend upon threads executing in a specific order or on a specific processor/core.
* Mutex: Abbreviation for “mutual exclusion.” Mutex variables are one of the primary means of implementing thread synchronization and for protecting shared data when multiple writes occur.