# CS 511 Final Exam Preparation

## 1 Overview

The final exam for CS 511, Concurrent Programming, will be given 6-10pm in 330 E.A. Stevens on Saturday, Dec. 13.

Your exam score will count form 16% of your overall course grade.

The exam will be open book, open notes. Electronic devices will be prohibited.

There will be several questions, each requiring a short answer. Topics will be taken from lecture material. Expect the exam to last approximately two hours although four hours will be available to you. All exam questions will be taken verbatim, or nearly so, from the list of questions/topics below.

## 2 Possible Questions

Explain the difference between a process and a thread.

Explain when it is more appropriate to use a process instead of a thread. When is it more appropriate to use a thread instead of a process?

Draw the state transition diagram for a process. Label the states and indicate examples of events that cause transitions among states. Show all transitions.

Write C code to create a new UNIX process. Your code should execute function `A()` if the attempt to create the new process fails. If the attempt succeeds, your code should execute function `B()` in the parent process and function `C()` in the child process.

Explain how to wait for input from more than one file descriptor simultaneously.

Explain what it means to "block," "ignore," or "handle" a UNIX signal. Which system call is used to block a signal? To ignore or handle a signal?

Some C library functions should not be called by a signal handler. Explain what aspects of a function make it one that shouldn't be called by a signal handler and name one such function.

What does it mean for a section of code to be "reentrant"?

The non-reentrancy of some functions can be spotted based only on a description of the function's interface. Explain how to do so.

Explain how hardware interrupts are used by the operating system to implement preemptive scheduling.

Write C code to create a new Pthread, wait for it to finish executing, then collect its result. The thread's start function should accept 3 arguments: an integer, a string, and a floating point number. The thread should return an integer.

Explain how to pass an unbounded amount of data as the argument to a Pthread even though its start function accepts only a single "`void *`" argument.

Explain how a "zombie" UNIX process is created. Explain why Pthreads does not include the zombie concept.

Explain how to make a function thread-safe.

Explain how to make a function cancel-safe.

What is the result if an N-thread process forks? That is, how many threads does the child process have and what code is each executing when the child process begins?

What happens to each of a process's threads when a system call in the "`exec`" family is invoked?

If a signal is generated by hardware or software exception, to which thread in a multi-threaded process is it delivered? Give an example of one such signal type.

If a signal is generated by an external process, to which thread in a multi-threaded process is it delivered?

Give an example of a race condition. Explain how the race condition may lead to incorrect results.

Explain how preemptive scheduling may cause race conditions to occur.

Which statements of a high level programming language (e.g., C or Java) may a programmer correctly assume to be atomic on typical computer architectures? Which assembly language statements?

Explain the major advantage and major disadvantage of non-preemptive scheduling.

Explain what mutual exclusion is. What problem does it solve?

List and explain the necessary conditions for a satisfactory solution to mutual exclusion.

"Peterson's Solution" and "Strict Alternation" provide mutual exclusion assuming only atomic single-word read and write operations. Explain how these solutions are inadequate.

Explain why the disabling of interrupts is no longer a viable synchronization technique, even inside operating systems.

Explain each of these terms: liveness, starvation, livelock.

What does it mean to "busy-wait" or "spin" waiting for a lock? What is disadvantageous about this technique?

Explain how to use the "load-linked" and "store-conditional" machine instructions to implement a lock, and explain why this solution is suited to a shared memory multiprocessor.

Explain what a semaphore is and how it is used to provide mutual exclusion.

Explain the primary disadvantage of the semaphore concept.

Explain why it is not a good idea for more than one thread to share a single file descriptor.

Explain what is the "monitor invariant"?

Name the three major operations on a monitor and explain their operation.

Why should an application-specific condition—failure of which to be satisfied would lead a monitor procedure to call `wait`—be tested by a `while` loop rather than tested by an `if` statement?

How is a "recursive" Pthread mutex lock different from a standard Pthread mutex lock?

What is dangerous about cancelling a Pthread? How can these dangers be avoided or mitigated?

What is "fairness" (wrt accessing a resource, such as a lock)?

What is the purpose of having multi-mode locks? Give an example of a moded lock.

Consider the `Runnable` and `Callable` Java interfaces. Explain what they have in common and what are their differences.

Explain how the Java `Future` interface is useful.

Suppose that `int foo(String a, float b)` is a Java method that should be executed in a separate thread. Write Java code to create a new thread, execute `foo` with the proper arguments, and retrieve `foo`'s result.

In Java what is an "object lock"? How is it useful for controlling concurrency?

Contrast the functionality provided by a monitor versus the functionality provided by Java's `synchronized` keyword.

Evaluate the functionality provided by Java's `synchronized` keyword as a solution for mutual exclusion.

Write Java code for a synchronized block (block only, not an entire method). Write "BODY" to indicate where the statements of the block's body would be.

Write Java code that gets a lock using Java's `ReentrantLock` class and is guaranteed to drop the lock even if an exception is thrown between lock-obtaining statement and the lock-dropping statement.

Write a monitor solution for the producer/consumer problem using Pthreads `pthread_mutex_*` and `pthread_cond_*` functions. Do the same using Java's `ReentrantLock` and `Condition` classes.

Suppose that a Java thread is using a read-write lock and already holds a lock in read mode. Explain the steps involved to correctly "upgrade" the lock to write mode.

Java provides a variety of thread pools. Explain how "fixed" and "cached" thread pools operate. What is the relative advantage of each over the other?

Java provides a variety of synchronized Collection classes. What is the major advantage of using such classes? What is the major disadvantage?

Explain how to use a "synchronization wrapper" to convert an unsynchronized Collection class into a synchronized class.

Java's `volatile` keyword is sometimes said to be useful for concurrent programming. Explain why. What is the main danger of using `volatile`? What programming approach can be used instead of using `volatile`?

Write Erlang code that starts a new thread. The thread should initially execute function "`func`" from module "`mod`" with three arguments `a`, `b`, and `c`. There is no need to retrieve the function's result.

Repeat the exercise above except that the new thread should be "monitored" by the initial thread.

What facilities does Erlang provide to control concurrency? Explain how these facilities could be used to implement a semaphore.

When an Erlang thread that is being monitored crashes, how does the monitoring thread learn about the crash?

Erlang does not permit threads to share memory. Explain how this feature aids the construction of reliable concurrent code.

If many Erlang threads each had to read and make updates to a single very large data structure, how would you design the code to do so?

Explain what "model checking" is.

What advantage does model checking have compared to testing?

What disadvantage does model checking have compared to testing? What disadvantage does model checking have compared to program proof?