CS 3013 Operating Systems
Craig E. Wills
Given: Friday, September 15, 2006

WPI, A Term 2006
Midterm Exam (100 pts)

NAME:_____

This is a closed book (and notes) examination. Answer all questions on the exam itself. Take the number of points assigned to each problem and the amount of space provided for your answer as a measure of the length and difficulty of the expected solution. The exam totals 100 points.

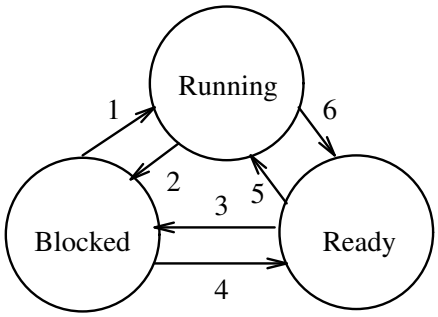
SCORE:_____

1. (16 points) Mutual exclusion problem.

   (a) One of the requirements for a solution to the mutual exclusion problem is to be *environment independent*. What is this requirement? Give an example of a potential solution that is not environment independent.

   (b) Mutual exclusion involves a critical section (region). What is a critical section (region) as it relates to the mutual exclusion problem?

2. (22 points) As shown below, processes can be in one of three states: running, ready and blocked. There are six possible state transitions (labeled 1-6). For each label, indicate whether the transition is *valid* or *invalid*. If valid, indicate when the transition is used for a process. If the transition is not valid then indicate why.



State transitions:

(a) 1: Blocked to Running

(b) 2: Running to Blocked

(c) 3: Ready to Blocked

(d) 4: Blocked to Ready

(e) 5: Ready to Running

(f) 6: Running to Ready

(g) Which transition (1-6) takes place for a *voluntary* context switch? Which transition takes place for an *involuntary* context switch?

3. (8 points) One means of designing an operating system is to organize it using a virtual machine approach. What is this approach and why is it used? Are there any disadvantages?

4. (13 points) In Project 1, you used various system calls for executing a command and gathering statistics about its execution.

   (a) Is it possible that a call to *fork()* can fail? If it cannot fail explain why, if it can fail give an example of when it would fail.

   (b) Is it possible that a call to *execve()* (*execvp()*) can fail? If it cannot fail explain why, if it can fail give an example of when it would fail.

   (c) The following is a "simplified" version of the *doit* program you wrote for Project 1. Explain the outcome(s) of running this code with command line arguments similar to those used in Project 1. Remember to consider all return codes for *fork()* and *execvp()*.

```
main(int argc, char *argv[])
{
    fork();
    execvp(argv[1], &argv[1]);  /* shift arguments */
    cout << "program finished\n";
}
```

5. (22 points) Process scheduling.

   (a) From the standpoint of system performance, name an advantage and a disadvantage of the shortest job first (shortest process next) process scheduling policy.

   (b) From the standpoint of system performance, name an advantage and a disadvantage of the first-come, first-served process scheduling policy.

   (c) Suppose that an operating system scheduling policy favors those processes that have used the least processor time in the recent past. Will this policy favor either short processes or long processes? Why?

6. (7 points) In terms of process coordination, what is a race condition?

7. (12 points) Assume that the integer variable `m` is a single mailbox shared among all processes in a system (this is not Unix/Linux). Processes put a value in the mailbox using the routine `SendMsg()` and remove the value from the mailbox using the routine `RecvMsg()`. If the mailbox already contains a value, then any process calling `SendMsg()` should block until the previous value has been removed. If the mailbox contains no value, then any process calling `RecvMsg()` should block until a value is available.

   Write the code for `SendMsg()` and `RecvMsg()` using semaphores to coordinate their activity. Show code in the `Initialization()` routine for any initialization that is needed.

```
int m;       /* shared mailbox (an integer) */

Initialization()
{


}

/* place the given value in the mailbox */
SendMsg(int value)
{




}

/* remove the value from the mailbox and return in the
   given address */
RecvMsg(int &value)
{




}
```