CMSC 412 Operating Systems

Practice Problems Exam 1

1.) Give brief description of the following terms:

a) Thread

b) Policy vs. mechanism

c) Preemptive multi-tasking

d) Critical section (three conditions)

e) PS register

f) c. MUTEX

g) d. Context Switch

h) e. System Calls

i) Interrupts and Traps

2.) Which of the following instructions must be treated as *privileged* and hence could be executed only in kernel mode? Give reasons for your answer.

a) Halt

b) Read the time-of-day clock

c) Set the time-of-day clock

d) Change process priority

e) Make a system call for I/O

f) Write the memory limit register

g) Floating Point Multiply

h) Modify PS Register.

i) Change value in instruction counter

3.) Deadlock

a) What are the four necessary conditions for deadlock?

b) Why are these four conditions necessary, but not sufficient?

4.) Synchronization

a) If you computer has an atomic swap instruction, but not test-and-set, show the code to simulate test-and-set using swap.

b) In a multi-processor system, why is turning off interrupts on a processor not a sufficient solution to the critical section problem?

5.) Why is synchronization necessary among processes when all of them are executing on a single CPU machine and only one of them can actually execute at any one moment?

6.) What is a system call and why is its use necessary is the design of an operating system.

7.) Why a thread is sometimes referred to as a “light weight process”? Explain.

8.) What are the two models of inter-process communications? What are the strengths and weaknesses of the two approaches?

9.) Why most computers support at least two modes (e.g. Kernel Mode and User Mode) and how are these modes used?

10.) Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single processor system? Explain

11.) Describe the actions taken by a kernel to context-switch between processes.

12.) What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.

a) Synchronous and asynchronous communication

b) Automatic and explicit buffering

c) Fixed-Sized and variable sized messages

13.) In a system that contains only one instance of each resource, circular waiting is a necessary and sufficient condition for deadlock.

a) Why circular waiting is sufficient if there is only one instance of each resource, but only necessary if there is one instance of each resource?

b) Consider a system with one instance of each resource, and the condition that requests for resources must be satisfied in the order in which they are received. Give an algorithm that takes as input requests to allocate and release resources and determines if the system is deadlocked.

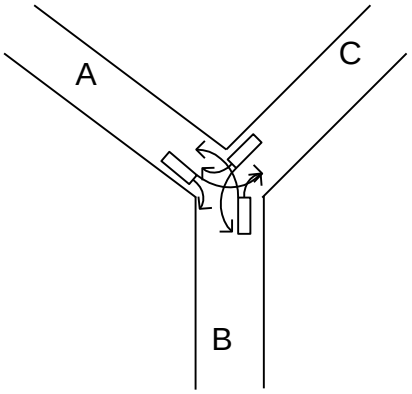
14.) Using test-and-set instructions, provide an algorithm that solves the dining philosophers problem. Recall that in this problem, n philosophers sit at a round table with one chop stick between each philosopher. Philosophers either are thinking or eating (rather dull people :). To eat, they need to acquire the chopsticks

to their left and their right. Your solution must work for any number of philosophers and ensure that none of them starves to death.

15.) A commonly used method for memory protection is to use base and limit registers which are checked for every memory access. Can this checking we done in software? If yes, how, if no, why not?

16.) Explain the difference between busy waiting and blocking.

17.) Consider the intersection shown in the figure below.



Cars arrive at the intersection along the roads marked A, B, and C, and proceed to take a left or a right turn. Thus Car A taking a right turn goes to road B and taking a left turn goes to road C, etc. The intersection has no traffic light or any other traffic control signs. Each car knows the road it is on and the turn it wants to take. Cars along each road come in one at a time and execute a procedure LEFT or RIGHT to go left or right respectively, passing the identity of the road it is coming from as an argument. (Thus car coming on road B and wanting to take a left will execute LEFT(B)).

a) Design the procedures LEFT and RIGHT using semaphores. The correct solution should support maximal parallelism, assure that no deadlocks occur, and there is no starvation. Of course there should be no collisions of the cars.

b) How many semaphore variables does your solution require and why?

c) What is the maximum number of cars that can be in the intersection in your solution?

d) Repeat this problem using a Monitor based solution.

18.) Consider the *mutual inclusion* problem in which N processes have a section of code which k(<N) of them must execute in the same time period, i.e. they must enter it together and leave it together. Therefore the structure of a process is as shown below:

loop

Non inclusion section
Inclusion Entry Code
Inclusion Code
Inclusion Exit Code
end

Write the Entry Code and the Exit Code using a fixed number of semaphores

19.) You have to solve a variation of the readers-writers problem, in which multiple writers can write at the same time. Specifically, there are readers and writers. Up to 5 reads at the same time are allowed, but only one write at the same time are allowed. A read and a write at the same time is not allowed. Provide a solution using semaphores with the following properties:

- no busy waiting.
- starvation-free (i.e. a continuous stream of readers does not starve writers, and vice versa) is desirable but not compulsory (**but you will lose some points**).
- you cannot use process ids and you cannot have a separate semaphore for every process.

Below is a skeleton program for you to build upon by supplying code for the boxes and perhaps introducing more

variables. You are also welcome to disregard this skeleton and come up with something else.

Declare variables and semaphores here. Please indicate initial values.

20.) Suppose processes P0 and P1 share variable V_{01}, Processes P1 and P2 Share variable V_{12}, and processes P2 and P3 share variable V_{23}.

a) Show how the processes can use enable/disable interrupts to coordinate their accesses to V_{01}, V_{12}, and V_{23} without developing race conditions

b) Show how the processes can use semaphores to coordinate their accesses to V_{01}, V_{12}, and V_{23} without developing race conditions

c) How would you implement a monitor to carry out the required synchronization?