CS 372 — Fall 1996 **Midterm #1**
Introduction to Operating Systems **October 23, 1996**

Please write your name on all pages. Read the questions carefully before writing your answer. The exam is worth 70 points.

GOOD LUCK!!!!

Problem 1 (10 pts)

For each of the following programs, list values possible for x if the program terminates. If the program terminates sometimes but not always, state it, and list the values for the cases in which the program terminates.. And, if the program never terminates, state it.

You may assume that variables are initialized as follows:

x: integer init 0

y: integer init 23

m: general semaphore, init 1

ml: general semaphore, init 0

mx: general semaphore, init 1

1. **cobegin** $x := x + 1 \parallel x := y + 1$ **coend**

2. **cobegin**
   $P(m); x := x + 1; V(m)$
   $\parallel$
   $P(m); x := y + 1; V(m)$
   coend

3. **cobegin**
   $P(ml); x := x + 1; V(m)$
   $\parallel$
   $P(m); x := y + 1; V(ml)$
   coend

4. **cobegin**
   $V(ml); x := x + 1; P(m)$
   $\parallel$
   $V(ml); x := y + 1; P(ml)$
   coend

5. **cobegin**
   $P(mx); P(m); x := x + 1; V(mx); V(m)$
   $\parallel$
   $P(m); P(mx); x := y + 1; V(m); V(mx)$
   coend

Problem 2 (12 pts)

Simulate the traffic of two intersecting one-way streets using semaphore operations. In particular, the following rules should be satisfied:

- Only one car can be crossing the intersection at any time.
- When a car reaches the intersection of two one-way streets and there are no cars approaching from the other street, it should be allowed to cross.
- When cars are arriving from both directions, they should alternate to prevent infinite postponement in either direction.

You should give the implementation of two procedures, *EnterIntersection* and *ExitIntersection*, that take the direction as a parameter. These procedures would be called by *car processes* when attempting to enter the intersection and when exiting the intersection. Be sure to state what kind of semaphore you are using (binary or general) and the fairness property you are assuming (i.e. how do the semaphores choose which blocked process gets unblocked as a result of a V operation).

Problem 3 (9 pts)

Consider the following four potential sources of overhead in a context switch.

A. Procedure call and return.

B. Trap to operating system and return.

C. Save and restore general-purpose registers.

D. Save and restore memory-mapping context.

For each of the following context switches, list all of the above four sources of overhead that apply.

1. Switch from one user-level thread to another user-level thread in the same process.

2. Switch from one kernel thread to another kernel-thread in the same process.

3. Switch from one kernel thread to another kernel-thread in a different process.

Problem 4 (15 pts)

1. (5 pts) What is the difference between a binary semaphore and a condition variable (as in a monitor)? Describe a scenario in which the two behave differently.

2. (10 pts) Consider the following implementation of the general semaphore operations using binary semaphores:

   count: **integer** := 0;
   s: **integer** := {initial value of the semaphore};
   mutex: **binary semaphore** := 1;
   block: **binary semaphore** := 0;

   P(s): $\mathbf{P}_b$(mutex);
   if s = 0 **then**
   count := count + 1;
   $\mathbf{V}_b$(mutex);
   $\mathbf{P}_b$(block)
   else /* $s > 0$ */
   s := s - 1;
   $\mathbf{V}_b$(mutex);
   fi

   V(s): $\mathbf{P}_b$(mutex);
   if count = 0 **then**
   s:= s + 1;
   $\mathbf{V}_b$(mutex);
   else /* *count* > 0 */
   count := count - 1;
   $\mathbf{V}_b$(block);
   $\mathbf{V}_b$(mutex);
   fi

   This implementation suffers from a liveness problem. Trace an execution which ends with $s > 0$, *count* $= 0$ and a process waiting on the semaphore *block*. Why is this a liveness problem?

Problem 5 (7 pts)

1. (2 pts) What are the 4 necessary conditions for deadlock?

2. (5 pts) On the island of *Deadlokia*, all streets are one-way and have two lanes. Some streets are one-way from north to south, and all others are one-way from east to west. The only way to reach Deadlokia is via a one-way bridge, which deposits vehicles in the northeast corner of the island on a north-south street. The only way to leave Deadlokia is via a one way bridge that is reached from a north-south street in the southwest corner of the island.

   Deadlokia is scenic but extremely dangerous for tourists. Therefore, tourists never stop and leave their vehicles—they simply drive onto the island, drive around on the island (obeying the directionality of the one-way street and bridges), and drive off the island.

   Traffic on Deadlokia is chaotic, but Deadlokians like it as it is. They even refused in a referendum to adopt the following law:

   > "It is illegal for a vehicle to enter an intersection unless a way exists for that vehicle to exit the intersection."

   claiming that it would take creativity out of their driving experience.

   A poll taken by the Deadlokian Tourist Office shows that visitors to the island live in constant fear of a traffic deadlock (a *gridlock*), and you have been hired to study the situation. Is gridlock possible on Deadlokia? If so, explain how one might arise. If not, indicate which of the 4 necessary conditions for deadlock does not hold, and briefly explain why.

Problem 6 (8 pts)

- True or false. A thread cannot read the stack of another thread, even if that other thread is in the same process.

- True or false. An operating system that does not support multithreading has no use for semaphores.

- True or false. There are problems that can be solved with a monitor, but not with a binary semaphore.

- True or false. It never makes sense to use busy-waiting to implement a mutex on a uniprocessor. Briefly explain why.

Problem 7 (9 pts)

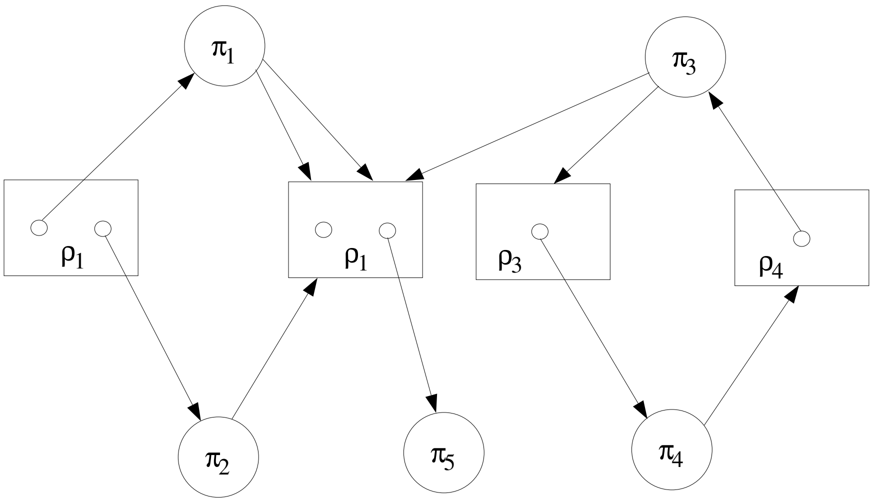

Figure 1: A reusable resource graph.

Consider the resource graph in Figure 1.

1. Which processes are blocked?
2. Which processes are deadlocked?
3. Is the state a deadlock state?

If you are using reductions to determine your answers, show your intermediate steps.