CS 162 Midterm Exam
October 18, 1993

Your Name:

Lecture (A.M. or P.M.):

General Information:

This is a closed book examination. You have 60 minutes to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points given to the question; there are 60 points in all. Write all of your answers directly on this paper. Make your answers as concise as possible (you needn't cover every available nano-acre with writing).

If there is something in a question that you believe is open to interpretation, then please go ahead and interpret *BUT* state your assumptions in your answer.

Problem 1: (6 points)

For each of the following statements, indicate in one sentence whether the statement is true or false, and why.

(a) If a set of cooperating threads can correctly time-share one processor, then they will correctly execute on separate processors of a multiprocessor.

(b) The CPU scheduling policy that minimizes average response time can never lead to starvation.

(c) If a system employing only segmentation wants to enlarge an address space, it is always possible if the total amount of free memory is bigger than the requested enlargement.

Problem 2: (8 points)

For each of the following items, write a one sentence definition:

(a) Interrupt

(b) Fully associative cache

(c) Working set

(d) Throughput

Problem 3: (12 points)

(a) What is the worst possible CPU scheduling algorithm – the one that yields the worst average response time? Assume that the algorithm must run a thread if there is one available, and that a context switch has zero overhead.

Problem 3 (cont'd):

(b) Can round robin ever be the worst possible CPU scheduling algorithm? If so, under what circumstances? If not, explain why not.

(c) Can FIFO ever be the worst possible CPU scheduling algorithm? If so, under what circumstances? If not, explain why not.

Problem 4: (10 points)

Consider the clock algorithm for page replacement, being used in its simplest form ("first-chance" replacement, where the clock is only advanced on a page fault, not in the background). Suppose that there are P pages of physical memory in the system and that over a particular interval of time F page faults have occurred.

Express the minimum and maximum number of times that the clock hand could possibly have been advanced during the time interval, as a function of P and F. (Don't worry about "off by one" errors, I'm just looking for the basic idea.)

Problem 5: (24 points)

You have been hired by your professor to be a grader for CS 162. Below you will find some sample solutions to a concurrency assignment. For each proposed solution, mark it either (i) "correct", if it has no flaws, (ii) "incorrect", if it does not work, or (iii) "dangerous", if it sometimes works but sometimes doesn't. You may assume either Mesa-style or Hoare-style condition variables, but if it matters, you must say which one you are using.

If the solution is incorrect or dangerous, explain everything wrong with the solution, and add a **minimal** amount of code to correct the problem. (NOTE: don't just implement a completely different solution – use the code we provide as a base.)

Here is the synchronization problem: A particular river crossing is shared by both cannibals and missionaries. A boat is used to cross the river, but it only seats three people, and must always carry a full load. In order to guarantee the safety of the missionaries, you cannot put one missionary and two cannibals in the same boat (because the cannibals would gang up and eat the missionary), but all other combinations are legal. Two procedures are needed: *MissionaryArrives* and *CannibalArrives*, called by a missionary or cannibal when it arrives at the river bank. The procedures arrange the arriving missionaries and cannibals into safe boatloads; once the boat is full, one thread calls *RowBoat* and after the call to *RowBoat*, the three procedures then return. There should also be no undue waiting: missionaries and cannibals should not wait if there are enough of them for a safe boatload.

Here is one proposed solution:

```
int numMissionaries = 0, numCannibals = 0;
Lock *mutex;
Condition *missWait, *cannWait;

void RowBoat() {printf("Row, row, row your boat");}

void MissionaryArrives() {
  mutex->Acquire();
  if (numMissionaries == 2) {
    missWait->Signal();
    missWait->Signal();
    RowBoat();
  else if (numMissionaries == 1 && numCannibals == 1) {
    missWait->Signal();
    cannWait->Signal();
    RowBoat();
  } else {
    numMissionaries++;
    missionaryWait->Wait(mutex);
    numMissionaries--;
  }
  mutex->Release();
}

void CannibalArrives() {
  mutex->Acquire();
  if (numCannibals == 2) {
    cannWait->Signal();
    cannWait->Signal();
    RowBoat();
  else if (numMissionaries == 2) {
    missWait->Signal();
    missWait->Signal();
    RowBoat();
  } else {
    numCannibals++;
    cannWait->Wait(mutex);
    numCannibals--;
  }
  mutex->Release();
}
```

Here is another proposed solution (don't worry about the potential starvation due to the loop in PersonArrives – the scheduler causes lots of random timeslices!):

```
Semaphore *boatCount = new Sempahore(3);       // initial value = 3
int numMissionaries = 0, numCannibals = 0;
Lock *boatLock;
Condition *boatWait;
bool success;

void RowBoat() { success = TRUE;}
void MissionaryArrives() { PersonArrives(&numMissionaries); }
void CannibalArrives() { PersonArrives(&numCannibals); }

void PersonArrives(int *numPeople) {
  for (;;) {
    boatCount->P();
    *numPeople++;
    if ((numMissionaries + numCannibals) < 3) {
      boatLock->Acquire();
      boatWait->Wait(boatLock);
    } else {
      if (!(numMissionaries == 1) && (numCannibals == 2))
        RowBoat();
      else
        success = FALSE;
      boatWait->Signal();
      boatWait->Signal();
    }
    numPeople->P();
    boatCount->V();
    if (success)
      return;
  }
}
```

Problem 6: Extra Credit (2 points)

Give an engineering estimate (to within a factor of 2 or so) of the number of barbers and hairdressers in the United States. Explain your method.