CS162, Fall/1993
Midterm 1

General Information

&nbsp &nbsp &nbsp This is a closed bok 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).
&nbsp &nbsp &nbsp If there is something in a question 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) &nbsp If a set of cooperating threads can correctly time-share one processor, then they will correctly execute on separate processors of a multiprocessor.

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

(c) &nbsp If a system employing only segmentation wants to enlarge an addresss 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) &nbsp Interrupt

(b) &nbsp Fully associative cache

(c) &nbsp Working set

(d) &nbsp Throughput

Problem #3: (12 points)

(a) &nbsp 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.

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

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

Problem #4: (10 points)

&nbsp &nbsp &nbsp 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.
&nbsp &nbsp &nbsp Express the minimum and maximum number of times that the clock hand could possibly have been advancing 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)

&nbsp &nbsp &nbsp you have been hired by your professor to be a grader for CS162. Below you will find some sample solutions to a concurrency assignment. For each poroposed 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.
&nbsp &nbsp &nbsp 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.)
&nbsp &nbsp &nbsp Here is the sychronization 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 ro *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, *canWiat;

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

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

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

&nbsp &nbsp 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 Semaphore(3);
int numMissionaries = 0, numCannibals = 0;
Lock *boatLock;
Condition *boatWait;
bool success;

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

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

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.