Midterm

CS 414
12th October 2006

NetID/Email: _____

Read all of the following information before starting the exam:

- Write down your netid/email NOW.
- Write legibly. If I can't read it, I can't give you credit.
- Please keep your written answers brief; be clear and to the point. I will take points off for rambling and for incorrect or irrelevant statements.
- If a question is unclear, please simply answer the question and state your assumptions clearly.
- This test has 5 problems and is worth 40 points, plus a bonus question the end. It is your responsibility to make sure that you have all of the pages.
- Good luck!

1. (*10 points*) Consider the following set of processes, with length of the CPU burst given in milliseconds:

Process	Burst time
P1	7
P2	1
P3	2
P4	5
P5	3

The processes are assumed to arrive in the following order: P1, P2, P3, P4, P5, all at time 0.

a. (*5 pts*) Show the scheduling order for these processes under first-come-first-served (FCFS), shortest-job first (SJF), and round-robin (RR) scheduling (quantum = 1).

b. (*4 pts*) For each process in each schedule above, indicate the wait time and turnaround time.

c. (*1 pt*) Briefly explain why these two values are meaningful criteria for evaluating scheduling algorithms.

2. (*10 points*) In parallel programs (a single process with multiple threads), a common way to design the processing is in stages. All threads work independently during each stage, but must synchronize at the end of each stage. When all threads reach this synchronization point (called a barrier), they are notified and begin execution on the next phase of the computation.
Two potential complications: In many cases, there is no master thread watching over the children threads, waiting for each of them to get to the barrier, and then telling them to re-start. In other words, the children must monitor themselves! Second, it is often not known in advance how many children threads there will be during the lifetime of the parallel program. In other words, a child can spawn another child (realizing that this new child will start in the same program stage as the child which created it)! In all cases, all children must synchronize at the barrier before the processing is allowed to continue to the next phase.

a. (*8 pts*) Your task is to provide the pseudo code for a monitor class called Barrier which enables this style of barrier synchronization. Things to take into consideration are the creation of a new child thread (one more process that needs to synchronize), processing when a thread reaches the barrier, and the releasing of waiting threads when the last thread reach the barrier. Remember that some of the standard monitor methods available to you are wait() and signal() and broadcast().

b. (*2 pts*) Did you put the call(s) to wait() at the end of your method(s)? If so, why? If not, what possible complication might arise after your call to broadcast()?

3. (*5 points*) Consider a system with four processes P1 through P4 and five allocatable resources R1 through R5 . The current allocation and maximum needs are as follows:

Process	Allocated	Maximum	Available
P1	1 0 2 1 1	1 1 2 1 3	0 0 x 1 1
P2	2 0 1 1 0	2 2 2 1 0	
P3	1 1 0 1 0	2 1 3 1 0	
P4	1 1 1 1 0	1 1 2 2 1	

What is the smallest value of x for which this is a safe state? For full credit, you must justify your answer.

4. (*5 points*) What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?

5. (*10 points*) Define both internal and external fragmentation. Draw a pictorial representation of a pure paging system (not paged segmentation, or segmented paging, but just plain paging) which demonstrates internal fragmentation. Draw a pictorial representation of a pure segmentation system which demonstrates external fragmentation.

Bonus Question (For Glory Only!):

In 1966, Communications of the ACM, one of the most distinguished computing journals in the world, published this incorrect solution to the two process mutual exclusion problem:

```
1  Shared: boolean blocked[2];
2                  int turn;
3
4  Init:  blocked[0] = false;
5          blocked[1] = false;
6          turn = 0;
7
8
9  P(i) {
10       while(true) {
11              blocked[i] = true;
12              while(turn != i) {
13                     while(blocked[1-i]) ;
14                      turn = i;
15               }
16               /* critical section */
17               blocked[i] = false;
18       }
19 }
```

Assuming load/stores are atomic operations show a counterexample that demonstrates that this solution is incorrect. Use line numbers in your explanation.