Assignment 3

CS 325

50 points, due Apr. 1, 2009

1. (10 pts.) An alternative to the TAS instruction is the FAI (Fetch And Increment) instruction. Here are its semantics:

   ```
   int FAI(int& val)
   {
      return val++;   // Performed atomically.
   }
   ```

   Devise a solution to the critical section problem for n processes using this instruction. Model your solution on the manner in which a bakery, for instance, serializes customer service by means of a number dispenser.)

   Show that your solution is, in fact, a solution to the critical section problem.

2. (15 pts.) A barbershop consists of a waiting room with `N` chairs, and the barber room containing the barber chair. If there are no customers to be served the barber goes to sleep. If a customer enters the barbershop and all chairs are busy, then the customer leaves the shop. If the barber is busy, then the customer sits in one of the available free chairs. If the barber is asleep, the customer wakes the barber up. Write pseudo-code for the cooperating barber and customer threads. In devising your solution, you may make use of queueing semaphores.

3. (15 pts.) Consider the following set of processes:

Process	**Arrival Time**	**CPU Bursts**	**Priority**
P1	0	2, 3, 2	3
P2	2	1, 1	1
P3	2	1, 2	3
P4	3	2, 1	4
P5	4	1, 2, 2	2

   Draw four Gantt charts illustrating the execution of these processes using FCFS (non-preemptive), preemptive SJF, non-preemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1).

   Also, for each scheduling algorithm, answer the following:

   (a) What is the turnaround time of each process?

   (b) What is the waiting time of each process?

   Which algorithm results in the smallest average waiting time?

4. (10 pts.) The first known correct software solution to the critical section problem for twp processes was proposed by Dekker:

```
#define FALSE 0
#define TRUE 1

int flag[2] = { FALSE, FALSE }   /* flag[i] indicates that Pi wants to */
                                 /*  enter its critical section */
int turn = 0;                    /* turn indicates which process has */
                                 /*  priority in entering its critical */
                                 /*  section


flag[i] = TRUE;
while (flag[1 - i])
   if (turn == 1 - i) {
      flag[i] = FALSE;
      while (turn == 1 - i)
         ;
      flag[i] = TRUE;
   }
/* Critical section for Pi. */
turn = 1 - i;
flag[i] = FALSE;
```

Prove that the algorithm satisfies the three requirements for a solution to the critical section problem.