Systems Programming and Operating Systems	Assignment 1
CS414, Summer 1996	July 11, 1996; Due on July 18, 1996

Problem 1 (30 points)

Imagine a job pool which has received jobs which need to be scheduled. Each job has the following run time and the following priority –

Process	Running Time	Priority
1	10	30
2	20	20
3	15	25
4	17	24
5	18	16

A job with a higher priority number has the higher priority.

Determine the following

1. Average turnaround time
2. Average waiting time
3. Average scheduling time

for the following policies –

i FCFS.

ii Shortest Job First.

iii Priority with **no** preemption.

Since no preemption is allowed in these schedules, the average waiting time is euqal to the average scheduling time.

1. **FCFS** – The processes are serviced in the order they arrived.

Process	Starting Time	Finishing Time
1	0	10
2	10	30
3	30	45
4	45	62
5	62	80

(a) Average Turnaround Time $= (10 + 30 + 45 + 62 + 80)/5 = 45.4$

(b) Average Scheduling Time $= (0 + 10 + 30 + 45 + 62)/5 = 29.4$

2. **Shortest Job First** – The job with the shortest execution time is scheduled first.

Process	Starting Time	Finishing Time
1	0	10
3	10	25
4	25	42
5	42	60
2	60	80

(a) Average Turnaround Time = $(10 + 25 + 42 + 60 + 80)/5 = 43.4$

(b) Average Scheduling Time = $(0 + 10 + 25 + 42 + 60)/5 = 27.4$

3. **Priority Scheduling** – The preocesses are scheduled with the highest priority first.

Process	Starting Time	Finishing Time
1	0	10
3	10	25
4	25	42
2	42	62
5	62	80

(a) Average Turnaround Time = $(10 + 25 + 42 + 62 + 80)/5 = 43.8$

(b) Average Scheduling Time = $(0 + 10 + 25 + 42 + 62)/5 = 27.8$

Problem 2 (20 points)

Given a set of processes $P_0, P_1, \ldots P_n$ arriving in a pool which can run to completion (i.e. no preemption) prove that the shortest job first scheduling has the minimum average waiting time of all possible schedules.

In the case where there is no premeption and the processes are run to completion the average waiting time is equal to the average scheduling time.

Assume that the schedule S_1 with the minimum average waiting time is not SJF. Let the schedule be $P_1, P_2, \ldots, P_i, \ldots P_j, \ldots P_n$. Let the running times of the processes be $T_1, T_2 \ldots T_i, \ldots T_j, T_n$. There must exist i and j such that $j > i$ and $T_i > T_j$ (because this is not SJF). The average waiting time for this schedule is given by $T_{S_1} = ((n-1) \times T_1 + (n-2) \times T_2 + \ldots + (n-i) * T_i + \ldots + (n-j) \times T_j + \ldots 1 + \times T_{n-1})/n$

In the schedule with P_i and P_j interchanged (S_2), the waiting time is $T_{S_2} = ((n-1) \times T_0 + (n-2) \times T_1 + \ldots + (n-i) * T_j + \ldots (n-j) \times T_i + \ldots + 1 \times T_{n-1})/n$

The quantity $T_{S_1} - T_{S_2}$ is equal to $(j-i) \times (T_i - T_j)$ which is greater than 0. Hence S_1 cannot be the minimising schedule which is a contradiction.

Problem 3 (10 points)

In the third (correct) solution of the ciritcal section problem, reproduced below –

```
flag[self] = TRUE
turn    = other
```

```
while (flag[other] && turn != self) no-op

CRITICAL SECTION

flag[self] = FALSE
```

why does *turn* have to be set to *other* and not to *self*?
If *turn* is set to *self* the solution will lead to

1. **Violation of Mutual Exclusion** – Consider the case where P_1 is in the critical section and P_2 tries to enter. It will set *turn* to *self* and the condition $turn != self$ will be false and hence it will not wait in the while loop.

2. **Starvation** To see this, consider the case where P_1 executes the code script in a loop. Initially P_1 is in the critical section while P_2 is waiting in the while loop and has been put to sleep by the scheduler. Meanwhile, P_1 zips around and enters the critical section again setting *turn* to *self*. If P_2 is woken up here it will not get to enter the critical section. In fact it may never get to enter the critical section if this scenario continues.