Student Name /ID: _______________
Student Signature: ________________ Date: _______________

NETS3304/3604 Operating System Internals

Tutorial 5

1. Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.

a. Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?
c. What is the waiting time of each process for each of the scheduling algorithms in part a?
d. Which of the schedules in part a results in the minimal average waiting time (over all processes)?

Answer:

a. The four Gantt charts are: (omitted)

b. Turnaround time:

	FCFS	RR	SJF	Priority
P_1	10	19	19	16
P_2	11	2	1	1
P_3	13	7	4	18
P_4	14	4	2	19
P_5	19	14	9	6

c. Waiting time (turnaround time minus burst time):

	FCFS	RR	SJF	Priority
P_1	0	9	9	6
P_2	10	1	0	0
P_3	11	5	2	16
P_4	13	3	1	18
P_5	14	9	4	1

d. Shortest Job First.

Tutor signature: __________________ Date: _______________

Student Name /ID: _______________
Student Signature: ________________ Date: _______________

2. What is the meaning of the term *busy waiting*? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

Answer: *Busy waiting* means that a process is waiting for a condition to be satisfied in a tight loop without relinquish the processor. Alternatively, a process could wait by relinquishing the processor, and block on a condition and wait to be awakened at some appropriate time in the future. Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached.

3. The **sleeping barber problem** is a classic inter-process communication and synchronization problem. The problem is analogous to that of keeping a barber working when there are customers, resting when there are none and doing so in an orderly manner. The barber and his customers represent aforementioned processes.
   The analogy is based upon a hypothetical barber shop with one barber, one barber chair, and a number of chairs for waiting customers. When there are no customers, the barber sits in his chair and sleeps. As soon as a customer arrives, he either awakens the barber or, if the barber is cutting someone else's hair, sits down in one of the vacant chairs. If all of the chairs are occupied, the newly arrived customer simply leaves. Construct a software skeleton using semaphores for solving the problem.

Answer: (suggested)

The most common solution involves using three semaphores: one for any waiting customers, one for the barber (to see if he is idle), and a mutex. When a customer arrives, he attempts to acquire the mutex, and waits until he has succeeded. The customer then checks to see if there is an empty chair for him (either one in the waiting room or the barber chair), and if none of these are empty, leaves. Otherwise the customer takes a seat – thus reducing the number available (a critical section). The customer then signals the barber to awaken through his semaphore, and the mutex is released to allow other customers (or the barber) the ability to acquire it. If the barber is not free, the customer then waits. The barber sits in a perpetual waiting loop, being awakened by any waiting customers. Once he is awoken, he signals the waiting customers through their semaphore, allowing them to get their hair cut one at a time.

```
int NumberOfFreeSeats = N;
Semaphore Customers = 0;
Semaphore Barber = 0;
Semaphore accessSeats = 1; //(mutex)
The Barber (Thread):
while(true) //runs in an infinite loop
{
// tries to acquire a customer - if none is available
// he goes to sleep
P(Customers);
//at this time he has been awaken -> want to modify
//the number of available seats
P(accessSeats);
NumberOfFreeSeats++ //one chair gets free
//we don't need the lock on the chairs anymore
V(accessSeats);
```

Tutor signature: __________________ Date: _______________

Student Name /ID: _______________
Student Signature: ________________ Date: _______________

```
V(Barber); // the barber is ready to cut
//here the barber is cutting hair
}

The Customer (Thread):
P(accessSeats); //tries to get access to the chairs
if (NumberOfFreeSeats>0){ //if there are any free seats
NumberOfFreeSeats--; //sitting down on a chair
// don't need to lock the chairs anymore
V(accessSeats);
// notify the barber, who's waiting that there is
// a customer
V(Customers);
// now it's this customers turn, but wait if the
// barber is busy
P(Barber);
//here the customer is getting the hair cut
}
else
// there are no free seats
// tough luck, but don't forget to release the lock on
// the seats
V(accessSeats);
```

Tutor signature: __________________ Date: _______________