Utku Noyan
60717

COMP 304- Operating Systems: Assignment 2

Due: April 12, midnight

Notes: This is an individual assignment. No late assignment will be accepted. Please submit your answers through blackboard. This assignment is worth 10% of your total grade.

Corresponding TA: Aditya Sasongko (msasongko17@ku.edu.tr)

Problem 1

(20 points) Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P1	8	3
P2	5	1
P3	3	3
P4	1	4
P5	10	2

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

a) Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a non- preemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 4 ms) scheduling.

b) Calculate the waiting time of each process for each of the scheduling algorithms in part (a). Which of the schedules results in the minimal average waiting time?

c) Calculate average turnaround time for each of the scheduling algorithms in part (a).

Assume no context-switch overhead.

Problem 2

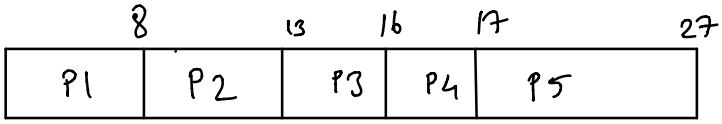
(10 points) Now assume that the context-switching overhead is equivalent to 0.5 ms. Calculate the CPU utilization for all four scheduling algorithms in Problem 1.

Problem 1

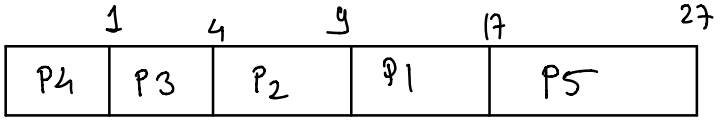
14 - 5 = 9

1.a

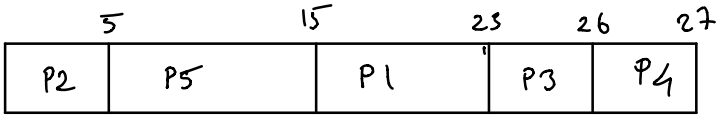
Process execution Gantt Chart with using FCFS



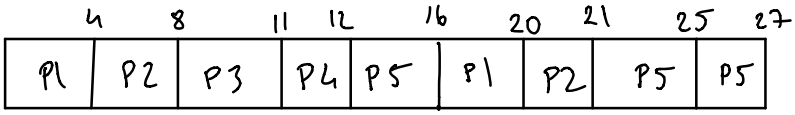
Process execution Gantt Chart with using SJF



Process execution Gantt Chart with using Non-preemptive Priority



Process execution Gantt Chart with using RR



1.b

FCFS

Waiting time of P1: 0

Waiting time of P2: 8

Waiting time of P3: 13

Waiting time of P4: 16

Waiting time of P5: 17

Average waiting time: (0+8+13+16+17)/5 = 12.8

SJF

Waiting time of P1: 9

Waiting time of P2: 4

Waiting time of P3: 1

Waiting time of P4: 0

Waiting time of P5: 17

Average waiting time: (9+4+1+17)/5 = 6.2

Utku NOYAN / 60717

Non-preemptive Priority

Waiting time of P1: 15

Waiting time of P2: 0

Waiting time of P3: 23

Waiting time of P4: 26

Waiting time of P5: 5

Average waiting time: (15+23+26+5)/5 = 13,8

RR

Waiting time of P1: 16-4=12

Waiting time of P2: 20-4=16

Waiting time of P3: 8

Waiting time of P4: 11

Waiting time of P5: 21-4=17

Average waiting time: (12+16+8+11+17)/5 = 12,8

+ SJF results the minimum average waiting time.

1.c

Average turnaround time for FCFS:

(8+13+16+17+27)/5 = 16.2

Average turnaround time for SJF:

(1+4+9+17+27)/5 = 11.6

Average turnaround time for Non-preemptive Priority:

(23+5+26+27+15)/5 = 19.2

Average turnaround time for RR:

(20+21+11+12+27)/5 = 18.2

Problem 2

$$\text{CPU utilization} = \frac{\text{Total time spend on process}}{\text{Total time spend on execution}} \times 100$$

CPU utilization of FCFS:

$(27/(27+4*0.5))*100 = \frac{27}{29} \times 100 = 93,10\ \%$

CPU utilization of SJF:

$(27/(27+4*0.5))*100 = \frac{27}{29} * 100 = 93,10\ \%$

CPU utilization of Non-preemptive Priority:

$(27/(27+4*0.5))*100 = \frac{27}{29} * 100 = 93,10\ \%$

CPU utilization of RR:

$(27/(27+8*0.5))*100 = \frac{27}{31} * 100 = 87,09\ \%$

1) Hu Noyan
60717

```c
//PROBLEM 3

#define MAX_CONNECTIONS 5000
int available_connections = MAX_CONNECTIONS;

/* When a thread wishes to establish a connection with the server,
it invokes the connect() function:*/

int connect() {
   if (available_connections < 1)
        return -1;
   else
       available_connections --;
   return 0;
}

/* When a thread wishes to drop a connection with the server,
it invokes disconnect() */
int disconnect() {
   available_connections ++;
   return 0;
}
```

Problem 3

(15 points) Consider the code example above that creates threads for opening connections at a server.
a) Identify the race condition(s) in the provided program.
b) If you have identified any race conditions, use pthread locks to prevent the race condition(s). Provide the code snippet.
c) Could we replace the integer variable available_connections with atomic integer, which allows atomic update of a variable of this type? Explain your answer.
atomic_t available_connections = MAX_CONNECTIONS;

Problem 4

(15 points) A hotel management would like to use online reservation system for their rooms. The hotel has M rooms available for online reservation. If all the rooms are occupied, the hotel will not accept any more customers until a room becomes available again (a party leaves the hotel).

Give an algorithm for the hotel using semaphore primitives to limit the number of reserved rooms so that it does not exceed M. Note that a room can accommodate up to 4 customers. If a party more than 4 customers arrives, multiple rooms may be needed. If the required number of rooms is not available, the room request for the entire party will be declined. Provide a pseudo-code and briefly explain your algorithm in a short paragraph.

Problem 3

3.a

Output of code is non-deterministic and relies on order of execution of procesi since there is race condition.

Integer variable available_connections is shared variable and possibly it is written by multiple threads concurrently inside of connect(), disconnect(). Because of write includes multiple instructions, race condition may happen when more threads access the shared data and try to change concurrently.

3.b

Code snippet with pthread locks is below.

```
1  //PROBLEM 3
2
3  #define MAX_CONNECTIONS 5000
4  int available_connections = MAX_CONNECTIONS;
5
6  /* When a thread wishes to establish a connection with the server,
7  it invokes the connect() function:*/
8
9  int connect() {
10     mutex_lock(&lock);
11     if(available_connections<1){
12        mutex_unlock(&lock);
13        return -1;
14     } else {
15        available_connections--;
16     }
17     mutex_unlock(&lock);
18     return 0;
19 }
20 /* When a thread wishes to drop a connection with the server,
21 it invokes disconnect() */
22 int disconnect() {
23     mutex_lock(&lock);
24     available_connections++;
25     mutex_unlock(&lock);
26     return 0;
27 }
```

3.c

The problem of change of shared variable by multiple threads concurrently can be prevented by using atomic integer. The only usage of atomic integer is not sufficient. When one of threads inside of the if statement of connect(), which is in line 11 before return, other thread can increase the available_connections that effect of the result of if statement. For instance, connect() can return 1 although available_connections is 1.

↳ actually 0 when line 11 checked, then it increased to 1 before return by other thread inside of disconnect().

Problem 4

The semaphore can be used with initial value M. If party with more than 4 numcustomers arrive, we have to check that we have enough rooms. Therefore, semaphore can be used with initial value 1, and global variable to check availability of room

```
1  //PROBLEM 4
2  int numOfRoom = M;
3  semaphore lockRoom = 1;
4  int checkout(int num){
5    wait(lockRoom);
6    numOfRoom = num + numOfRoom;
7    signal(lockRoom);
8  }
9  int Reserve(int numCustomer){
10   int needed = (numCustomer +3)/4;
11
12   wait(lockRoom);
13   if (numOfRoom >= needed){
14         //accept
15         numOfRoom = numOfRoom - needed;
16   } else{
17         //decline
18   }
19   signal(lockRoom);
20   return 0;
21 }
22
23
24
25
26
27
```

Wtku MOYAA
60717

Problem 5

(25 points) Consider a hospital that offers a free check-up service to senior people (65 years old or older) on Sundays. M many senior citizens $(n_1, n_2, ..., n_m)$ are expected to visit the hospital for this Sunday. Write a monitor (in pseudocode) that allocates 4 equally professional doctors to these people, using the priority numbers of patients for deciding the order of their doctor visit. The older the person, the higher her/his priority is.

When a patient arrives to the hospital, s/he should call *void request_doctor(int priority)* function. If all doctors are busy, the patient is made to wait. When a patient is done with her/his visit, the patient should call *release_doctor()* function. You can assume that there is a *sort()* function, which takes an integer array and sorts it.

Problem 6

(15 points) Consider the following snapshot of a system, where A, B, C and D are resources and P1-P5 are processes.

	Allocation	Max	Available
	A B C D	A B C D	A B C D
P1	0 0 1 4	0 6 5 6	1 5 2 0
P2	0 6 3 2	0 6 5 2	
P3	1 3 5 4	2 3 5 6	
P4	0 0 1 2	0 0 1 2	
P5	1 0 0 0	1 7 5 0	

Using the bankers algorithm for deadlock avoidance:

a) What is the content of the matrix Need?

b) Is the system in a safe state? Explain your answer.

c) If a request from process P5 requests (0, 3, 3, 0), can the request be granted immediately? Explain the reason.

Problem 5

```
condition cont;
int doctors[4] = [0,0,0,0]; // doctors assigned no patients
int requests[M] = 0; // initializing all requests as queue, max M patients
int ptr = 0; // points to last patient that requests
mutex mut;
void release-doctor(int index){
//index= doctor index
    mutex_lock(mut); // getting lock
    doctors[index] = 0; //make doctor free
    if( ptr > 0){  // If there is any patient that waits doctor, send next have highest
        doctors[index] = 1; //priority
         ptr--; // reduce the requests queue
    }
mutex_unlock(mut);
}
void request_doctor(int priority){
   mutex_lock(mut);
   for(int j=0; j<4; j++){
      if(doctors[j]==0){// Assign patient, if there is availible doctor.
          doctors[j] = 1;
          mutex_unlock(mut)
          return;
      }
   }
   requests[ptr++] = priority; // If there is not any availble doctor, push patient
                               // to requests queue
   sort(*requests[0], *requests[ptr]; // Since new patient added, sort queue from
   mutex_unlock(mut);                 // 0 to ptr.
}
```

Problem b:

6.a Need for process:

	A	B	C	D
P1 =	0	6	4	2
P2 =	0	0	2	0
P3 =	1	0	0	2
P4 =	0	0	0	0
P5 =	0	7	5	0

6.b Yes, with Available being equal to (1,5,2,0) either process P2, P4

Once process P2 runs, it releases its resources, which allow other existing process to run.

Satisfy need P2 = [1,5,2,0] + [0,6,3,2]

new availability = (1, 11, 5, 2)

Satisfy need P4 = (1,11,5,2) + (0,0,1,2) = (1,11,6,4) (new availability)

" " P1 = (1,11,6,4) + (0,0,1,4) = (1, 11, 7, 8)

" " P3 = (1,11,7,8) + (1,3,5,4) = (2, 14, 12, 12)

" " P5 = (2,14,12,12) + (1,0,0,0) = (3, 14, 12, 12)

We are able to satisfy need of all process, therefore in safe state.

6.c

Although we trying to allocate (0,3,3,0), available resources (1,5,2,0). We are trying to allocate 3 for C to P5 but we have only 2 resources. So it is wrong. Request is not granted.