Part 1: Process Management & Synchronization

Question A:

- How does a parent process create child processes using fork()?
- What is the behavior of child processes in terms of execution time and termination?
- How does the parent process use waitpid() to wait for its children?

First, we see that N is set to 30, a matrix is defined with N elements, an integer i and another integer child_status. Within the first for the table pid is filled, which will now contain the ids of the children of the parent process.This is done via the fork().If the id is 0 the called thread sleeps for 60-2*I secsecondd then exits in 100+i seconds. In the next for waitpid is called so the parent waits for the successful completion of the child. It checks via and wifexited , wxitstatus the way they did terminate and updates accordingly via an if(if the process child terminated normally, it prints it along with the id and status otherwise it prints that it terminated abnormally).

Question B:

- How can a user specify the number of child processes to create?
- How does the bakery algorithm ensure mutual exclusion among processes?
- How does the parent process wait for its children and handle their termination?

The user is asked to provide the number of affiliates that want to create and through a for and the fork the ids are stored in the pid table. SA is then asked to have zeros so through a for we initialize it with 0. Then we run the bakery algorithm just like in Mr. Makri's transparencies. To be able to we made a max function which finds the the largest number in a table. It is called waitpid ,the father waits for successful completion of the affiliate and is informed of the way they terminate. each one.

Part 2: Thread Synchronization & Scheduling

Question A:

- Given two threads modifying a shared variable, what are the possible final values due to different execution interleavings?

The orders according to the pronunciation are non-individual and each executed in three steps.

We have:

THREAD 1 -> X:X+1 THREAD

2 -> X:Y+1

1.TX=X 4.TY=Y

2.TX=X+1

5.TY=Y+1

3.X=XX 6.X=TY

The possible mixes are many, but they all result in a of the 3 values of the following mixtures.

(1)(2)(3)(4)(5)(6) -> X is 11

(4)(5)(6)(1)(2)(3) -> To X is 12

(4)(5)(1)(2)(6)(3) ->To X is 1

So, the possible values of X are 1,11,12.

Question B:

- Implement a synchronization mechanism using semaphores to control process execution order.
- Ensure that processes print "PIZZA" in a controlled manner.

a)

```
var s1,s2,s3: semaphore;
count:int;
begin
s1:=1;s2:=0;s3:=0;count=0;
cobegin
Process1 Process2 Process3
while(TRUE){ while(TRUE){ while(TRUE){
down(s1); down(s2); down(s3);
print("P"); print("Z"); print("A");
print("I"); count=count+1; }
up(s2); if(count==2){
} up(s3);
}
else{
up(s2);
}
endif;
}
coend;
end;
```

In the above code I initially defined my flags and an int variable count. First, I giveflagg s1 the value 1 and s2,s3 the value 0 so that the process Process1 and print the letters P,I , the value of flag s1 is now 0 and process1 is suspended while value of s2 is 1 therefore Process2 is awakened. In Process2 I print Z then through an if I check the value of count , if the value of count is 2 -which means that the Process2 has been executed 2 times therefore it has printed: Z,Z- the flag s3 takes the value 1 and Process3 is woken up and prints A, otherwise Process2 is awakened so that it prints A to give the second Z. After

Process3 is executed all the flags have the value 0 so it cannot be executed any other process, and the final print result will will be the word PIZZA.

b)

```
var s1,s2,s3: semaphore;
count:int;
begin
s1:=1;s2:=0;s3:=0;count=0;
cobegin
Process1 Process2 Process3
while(TRUE){ while(TRUE){ while(TRUE){
down(s1); down(s2); down(s3);
print("P"); print("Z"); print("A");
print("I"); count=count+1; up(s1);
up(s2); if(count==2){ }
} count=0;
up(s3);
}
else{
up(s2);
}
endif;
}
coend;
end;
```

To print the word PIZZA repeatedly I have the code of a) query with the first difference in Process3 where it becomes up(s1) so Process1 is woken up to repeat the word PIZZA

and the second difference in Process2 inside if before it is up(s3) I reset the count to zero to print Z only 2 times in each execution of Process2.

Question C:

- Given a set of processes and semaphore conditions, determine whether a specific execution order is possible.
- Analyze how semaphores control execution flow.

a)

The completion of the processes in the order given in the is possible and its execution scenario is as follows:

3 processes of type A enter the system simultaneously (A1,A2,A3) and 2 processes of type B (B1,B2). The value of the flag value s2 is initially 0 therefore it cannot be executed towards currently a process of type B. The value of flag s1 is initially 2 therefore a process of type A (A1 or A2 or A3) and A1 which we want in this case. After the execution of the A1 process of type A the values of the flags will be as follows: s1=1, s2=1 Process type B cannot be executed yet since the flag s2 will have to take the value 2 to be able to type B process to be executed. Since s1=1 we can execute a type A process (A2 or A3), namely we execute A2. After executing process A2 (type A) the flags s1,s2 have the following values: s1=0, s2=2 The process of type A can no longer be executed. However, the process of type B (B1 or B2), if s2=2, can be executed. Therefore, B1 is now executed. After the execution of B1 the values of the flags are as follows: s1=1, s2=1 The process of type B cannot be executed, but it can but a process of type A can be executed. So now A3 is executed. After execution of A3 the values of the flags are as follows:

s1=0, s2=2

Process type A can no longer be executed. But the process type B can be executed as long as s2=2. Therefore, B3 is now running. The values of the flags will now be as follows:

s1=1, s2=0

b)

The completion of the processes in the order given in the pronunciation is not possible. The execution scenario of A1, A2, B1 is the same as in question a). After executing B1 I

notice that the values of the flags s1 and s2 are: s1=1, s2=1 I notice that based on the order of execution of the pronunciation after B1 we want to execute B2 , but the s2 flag to any process of type B must have a value of s2>=2. So, since s2=1 B2 cannot be executed after B1. Therefore, the processes cannot be completed with either order.

Question D:

- Why is process synchronization necessary when multiple threads access shared data?
- How do semaphores prevent race conditions?

(a)

If there are no markers to control the flow of the code, process instructions are likely not to be executed in the desired order. How can this be done? A process to execute its instructions must enter the critical area. When there are markers, the mutually exclusive other processes are mutually exclusive and cannot enter the critical area while another process is in the critical area. But now, a process can enter the critical area and fly out of the critical area another processes out of it, while that process has not completed all its instructions.

One possible scenario :

Process_i :

L1 : while (TRUE) {

L2 : L:=K;

L3 : K:K+11;

L4 : print_num(L, L+10);

L5 : }

P1 : L1,L2,L3 -> P2 : L1,L2,L3,L4,L5 ->

-> P3 : L1,L2,L3,L4,L5

In this order, the result will be printed :

1st time : 12 to 22

2nd time : 23 to 33

3rd time : 34 to 44

(b)

The code after importing flags will become :

Var s : semaphore;

S :=1;

Process_i

Begin

Wait(s);

L1

L2

L3

L4

L5

Signal(s);

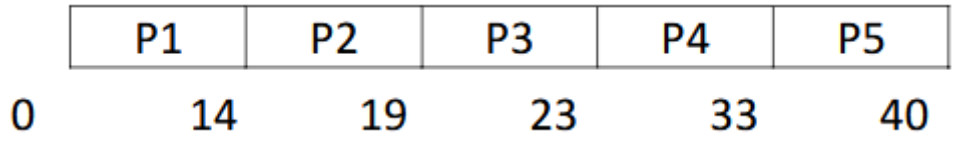
End

Question E:

Implement and analyze various CPU scheduling algorithms:

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time First (SRTF)
- Priority Scheduling
- Round Robin (RR)

First Come First Served Algorithm



Where the times are added to the numbers below the table of each process. In this algorithm, processes are executed according to the order of their arrival. If t1 is the entry time and t2 is the exit time, then the completion time is given by the formula:

XO=t2-t1.

XO1=14-0=14

XO2=19-2=17

XO3=23-4=19

XO4=33-7=26

XO5=40-12=28

So, the average completion time is: MXO=(14+17+19+26+28)/5=20.8

If t1 is the entry time and t2 is the start time of the process, then the waiting time is given by formula: HA=t2-t1.

XA1=0-0=0

XA2=14-2=12

XA3=19-4=15

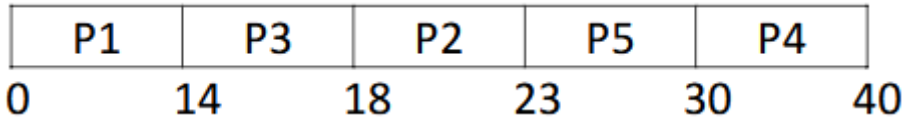
XA4=23-7=16

XA5=33-12=21

So, the average waiting time is: MXA=(0+12+15+16+21)/5=12.8

Shortest Job First Algorithm

In this algorithm, processes are executed according to which has the shortest execution time. The exception is first, which is the one with the shortest arrival time.



At time 0, there is only P1 in the system, so it is executed. P1 completes in 14 instants, so the rest will have arrived, since the arrival times of all all processes are 0, 2, 4, 4, 7, 12. So, now that all the processes in the system, we only look at the runtime and we follow the algorithm.

XO1=14-0=14

XO2=23-2=21

XO3=18-4=14

XO4=40-7=33

XO5=30-12=18

So, MXO=(14+21+14+33+18)/5=20

XA1=0-0=0

XA2=18-2=16

XA3=14-4=10

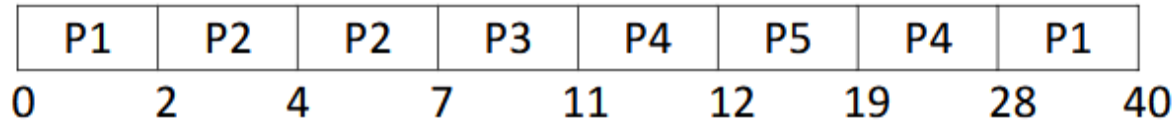
XA4=30-7=23

XA5=23-12=11

So, MXA=(0+16+10+23+11)/5=12

Shortest Remaining Time First

In this algorithm, priority is given to processes with the shortest remaining time.



Pre-shifting is used, i.e. if a process arrives and there is less running time, the running process will be interrupted and the process that just arrived will be executed. Once the new process arrives, a check is made to see if it has less remaining than the process currently running.

XO1=40-0=40

XO2=7-2=5

XO3=11-4=7

XO4=28-7=21

XO5=19-12=7

MXO=(40+5+7+21+7)/5=16

XA1=0-0=0

XA2=2-2=0

XA3=7-4=3

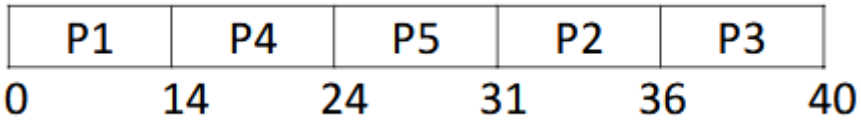
XA4=11-7=4

XA5=12-12=0

MXA=(0+0+3+4+0)=1.4

Priority Scheduling

In this algorithm, higher priority is given to the processes with the highest priority value.



It is a non-procedural algorithm, so the processes are executed in their entirety once they enter the CPU, without interruption.

XO1=14-0=14

XO2=36-2=34

XO3=40-4=36

XO4=24-7=17

XO5=31-12=19

MXO=(14+34+36+17+19)/5=24

XA1=0-0=0

XA2=31-2=29

XA3=36-4=32

XA4=14-7=7

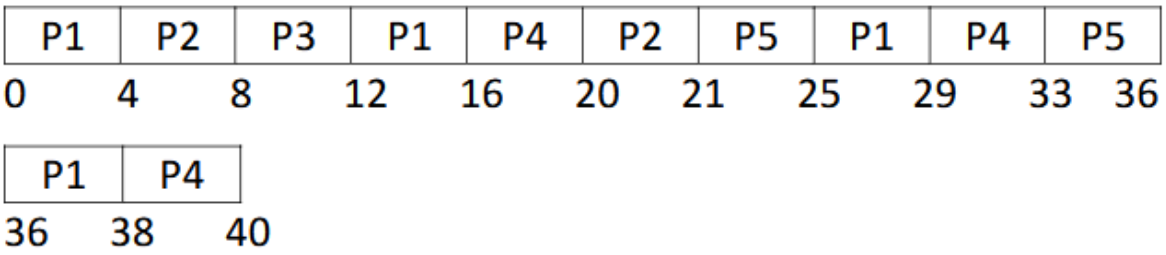
XA5=24-12=12

MXA=(0+29+32+7+12)/5=16

Round Robin Algorithm

In this algorithm, the processes are put in an order of execution. Each process is executed for a specific quantum of time. Once that is complete, they go to the end of the queue and the next process is executed. When a new process enters the system, it enters the execution queue first and then the process that was running and was interrupted.

Gantt :



Execution Queue :

P1 P2 P3 P1 P4 P2 P5 P1 P4 P5 P1 P4

XO1=38-0=38

XO2=21-2=19

XO3=12-4=8

XO4=40-7=33

XO5=36-12=24

MXO=(38+19+8+33+24)/5=24.4

XA1=0-0=0

XA2=4-2=2

XA3=8-4=4

XA4=16-7=9

XA5=21-12=9

MXA=(0+2+4+9+9)/5=4.8