Operating Systems - Midterm exam –2015
Computer Science Degree

* The exam duration is 1:50 **hours.**
***** It is not allowed to use any book or notes. Electronic devices are also not allowed.

Name and surname: **Group:**

Exercise 1 (20 points) .

Solution not available here. You can find the solutions doing quizzes online.

Exercise 2 (40 points) .
Develop a small program in C using system calls (e.g. fork(), waitpid(), exit(), kill(), ..) that does the following:

- A parent process creates two child processes
- The first child creates a grandchild and sleeps 120 seconds and exists returning value 1
- The grandchild waits 30 seconds and then executes the system call ls.
- The second child process sleeps 150 seconds and exists returning value 2
- The parent sleeps for 60 seconds and then kills the first child process and receives and prints the returning value of the second child.

Answer the following additional questions based on the program that you have written:

1. What happens to the **first child** if it is terminated (killed by a SIGKILL signal) at second 30?
2. What happens to the **second child** if the **first child** is terminated (killed by a SIGKILL signal) at second 30?
3. What happens to the **parent process** if the **first child** is terminated (killed by a SIGKILL signal) at second 30?
4. What happens to the **first child process** if the **parent process** is terminated (killed by a SIGKILL signal) at second 20?
5. What happens to the **grandchild process** if the **parent process** is terminated (killed by a SIGKILL signal) at second 20?

```
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
#include <signal.h>

main()
{
  int first_child_id, second_child_id, grandson_id,myID;
  int r1;
  first_child_id = fork();
  if(first_child_id == 0)  /* first child process*/
  {
        myID=getpid();
        printf("the child process ID is: %d\n", myID);
        grandson_id =fork();
        if(grandson_id ==0) /*grandchild process*/
        {
           printf("grandchild process \n");
           sleep(30);
           execlp("ls", "ls", NULL);
        }
        sleep(120); // First child
    }

  second_child_id = fork();
  if(second_child_id == 0)  /* second child process*/
  {
    sleep(150);
    exit(2);
  }
  else // Parent process
  {
    sleep(60);
    kill(first_child_id,SIGKILL);  /* Kill the first child process */
    wait(&r1);
    printf("\n All child processes terminated. Returning value: %d \n",r1)
    exit(1);
   }
}
```

Questions:

1. It becomes a zombie process.
2. Nothing.
3. Nothing but the kill system call will return an error.
4. It becomes orphan and it becomes child of the init process.
5. Nothing.

Exercise 3 (40 points) .

A certain single processor system with round-robin preemptive scheduling policy uses a time slice of 200 milliseconds. Average time to perform a context switch is 5 microseconds.

In this system 3 processes arrive:

- Process A arrives at instant t=0. This process performs computation during 300 milliseconds, then input/output during 150 milliseconds, and finally, computation for 100 milliseconds.
- Process B arrives at instant t=100 milliseconds. This process performs computations during 300 milliseconds. Then it repeats a task twice. Each iteration of this tasks performs computations during 150 milliseconds followed by input/output during 50 milliseconds.
- Process C also arrives at instant t=100 milliseconds. It only performs computations for 500 milliseconds.

In this system all I/O calls are blocking calls. Consequently when a process starts an I/O operation it transitions to the blocked state (waiting for I/O completion) and the scheduler may select another process for execution.

a) Given a general case with multiple processes where none of them performs input/output, what is the maximum percentage of time that the system would be using for context switching?

b) What would be the maximum time that can be afforded for the context switching time, in order to keep the context switching overhead below 0.1%?

c) Is it possible to get a normalized return time exactly equal to one? What does this situation mean?

d) Determine how the execution of processes A, B, and C would be in the case of non-preemptive SJF scheduling. Fill in a table as the one below. Please, ignore any overhead due to context switching, process startup and termination. Assume that once a process starts running it is no rescheduled even in the case of I/O operations. For the SJF duration, use the total amount of time used by CPU and I/O.

Process	Arrival	Service	Initiation	End	Return	Wait	Normalized Return
A	0						
B	100						
C	100						

e) Determine how execution of processes A, B, and C would be in the cases of preemptive Round-Robin (cyclic) scheduling. Fill in a table as the one below. Please, ignore any overhead due to context switching, process startup and termination.

Process	Arrival	Service	Initiation	End	Return	Wait	Normalized Return

A	0						
B	100						
C	100						

f) Compare average wait time and average normalized return time. What can you conclude from the results?

SOLUTION:

a) Let ts be the time for a time slice and tc the time needed for a context switch. Then the overhead ratio can be expressed as:

o = tc/ (ts+tc)

In this particular case:

o = 5/ (200000+5) = 0.000024

That is 0.0024%

b) Using the same equation:

0.001 = tc / (200000 + tc)
200 + 0.001 * tc = tc
0.999 * tc = 200

tc = 200.2 microseconds

c) Normalized return time is the ratio between the return time and the service time. By definition the return time is always greater or equal that the service time and consequently the normalized result can be one when both are equal.

This situation means that the process has not been any time waiting in queues.

d) With SJF, scheduling depends on service times. Computing service times for this particular case, we have:

- TS(A) = 300 + 150 + 100 = 550
- TS(B) = 300 + 2 * (150 + 50) = 700
- TS(C) = 500

At t=0 the only available process is A. When process A has finished, C is selected is it has a shorter time than B.

Process	Arrival	Service	Init	End	Return	Wait	Normalized Return

A	0	550	0	550	550	0	550/550=1.0
B	100	700	1050	1750	1650	950	1650/700=2.35
C	100	500	550	1050	950	400	950/500=1.90

e)

Instant	In Execution	Ready	Blocked	Pending computation	Pending Input/Output	Final
0	A			A(300)		
100	A	B,C		A(200), B(450), C(500)		
200	B	C, A		A(100), B(450), C(500)		
400	C	A, B		A(100), B(250), C(500)		
600	A	B, C		A(100), B(250), C(300)		
700	B	C	A	B(250), C(300)	A(150)	
850	B	C, A		A(100), B(100), C(300)		
900	C	A, B		A(100), B(50), C(300)		
1100	A	B, C		A(100), B(50), C(100)		
1200	B	C		B(50), C(100)		A
1250	C		B	C(100)	B(50)	
1300	C	B		B(150), C(50)		
1350	B			B(150)		C
1500	<nulo>		B		B(50)	
1550	<nulo>					B

Thus:

Process	Arrival	Service	Init	End	Return	Wait	Normalized Return
A	0	550	0	1200	1200	650	1200/550=2.18
B	100	700	200	1550	1450	750	1450/700=2.07
C	100	500	400	1350	1250	750	1250/500=2.5

f) In case of SJF, we have:

- Average wait time: (0+950+450)/3 = 466.67
- Average normalized return time: 1.75

In case of round-robin:

- Average wait time: (650+750+750)/3 = 716.67
- Average normalized return time: (2.18+2.07+2.5)/3 = 2.25

In this case cyclic scheduling increases both wait time and normalized return time. However the total completion time for the batch of jobs is lower. This effect is due to the fact that round-robin balances cpu sharing to increase fairness among processes.