Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Consolidation

Exercises 1

Exercise 1

An operating system uses scheduling algorithm based on "**priority classes**" with two levels, and **the scheduling policy for each level is FCFS**. **Scheduling between the two levels is preemptive priorities**, being the highest priority class 1. **When a process is preempted, it goes to the head of its queue**. Processes share a single disk with FCFS scheduling. The operating system assigns processes to one of two classes depending on their processor and disk consumption. **If a process has consumed more CPU time than disk, it is assigned to class 2, otherwise it is assigned to class 1**. Initially the processes are in class 1. The system determines the class for each process every unit of time. Three processes arrive simultaneously to the system, but in the order A, B and C. The execution profiles are shown in the following table.

Obtain the **processing time line**, the **average turnaround time** and the **average waiting time**.

Process	Execution profile
A	2 CPU + 1 DISK + 7CPU
B	1CPU + 3 DISK + 1 CPU + 2 DISK + 1 CPU + 2 DISK + 1 CPU
C	2CPU + 1 DISK + 1CPU + 2 DISK + 1 CPU + 2 DISK + 1 CPU

Ejercicio 1: Solución

t	READY		CPU	Queue Disk	Disk	Comments
	Class 1	Class 2				
0	CBA		A(1)			Arrive A, B & C
1	CB	A	B(0)			
2		A	C(1)	B	B(2)	
3		CA	A(0)		B(1)	
4			C(0)	A	B(0)	
5	B		B(0)	CA	A(0)	
6		A	A(6)	BC	C(0)	
7		C	A(5)		B(1)	
8		C	A(4)		B(0)	
9	B	CA	B(0)			
10		C	A(3)		B(1)	
11						
12						
13						
....						
...						

```
If time(CPU) > time(IO)
  class = class2
else
    class = class1
```

- Average Turnaround time

-Average waiting time

Process	Execution profile
A	2 CPU + 1 DISK + 7CPU
B	1CPU + 3 DISK + 1 CPU + 2 DISK + 1 CPU + 2 DISK + 1 CPU
C	2CPU + 1 DISK + 1CPU + 2 DISK + 1 CPU + 2 DISK + 1 CPU

Exercise 1

	FCFS	FCFS				
t	READY		CPU	Queue Disk	Disk	Comments
	Class 1	Class 2				
0	CBA		A(1)			Arrive A, B & C
1	CB	A	B(0)			CPU => B
2	C		C(1)		B(2)	
3		C	A(0)		B(1)	CPU => A
4			C(0)	A	B(0)	IO => B
5	B		B(0)	C	A(0)	CPU => B; IO => A
6		A	A(6)	B	C(0)	IO=>C
7		C	A(5)		B(1)	
8		C	A(4)		B(0)	IO => B
9	B	CA	B(0)			CPU => B
10		CA	A(3)		B(1)	
11			A(2)		B(0)	IO => B
12	B	CA	B(0)			end B
13		C	A(1)			
14		C	A(0)			end A
15			C(0)			
16					C(1)	
17					C(0)	
18	C		C(0)			
19					C(1)	
20					C(0)	
21	C		C(0)			end C
22						

If time(CPU) > time(IO)
class = class2
else
class = class1

	A	B	C
Finishing time	15	13	22
Waiting time	4	1	10
Tournaround	15	13	22

- Average Turnaround time
-Average waiting time

Process	Execution profile
A	2 CPU + 1 DISK + 7CPU
B	1CPU + 3 DISK + 1 CPU + 2 DISK + 1 CPU + 2 DISK + 1 CPU
C	2CPU + 1 DISK + 1CPU + 2 DISK + 1 CPU + 2 DISK + 1 CPU

Exercise 2

- This OS uses **four priority classes** numbered from 0 to 3. The scheduling algorithm is **Round Robin for classes 0, 1 and 2**, and is **FCFS for class 3**. The **highest priority class is 0**. Time quanta, **qi for classes 0, 1 and 2** are given by the following formula: **qi = i + 1**. The scheduling algorithm is **multilevel queue preemptive**.
- **Processes that arrive to the system are initially queued in the highest priority class (0)**. There is a mechanism of priority degradation following the rule: "**a process remains in its class until it has consumed 2 quanta, thereafter it is degraded to the next lower priority class**“. Processes that reach Class 3 remain there until they finish the execution.
- Obtain the **turnaround time** and the **ending class** of every of the following three processes:

Process	Arrival time	Execution Profile
P1	0	4 CPU
P2	0	8 CPU
P3	0	12 CPU

Exercise 2:

Multilevel queue preemptive algorithm

	READY QUEUEs					
T	Q3:FCFS	Q2:RR q=3	Q1:RR q=2	Q0:RR q=1	CPU	Events
0						Arrive A, B & C
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						

	Tiempo Retorno	Clase
P1	8	Clase 1
P2	18	Clase 2
P3	24	Clase 2

- Average Turnaround time

- Ending class

Process	Arrival time	Execution Profile
P1	0	4 CPU
P2	0	8 CPU
P3	0	12 CPU

Exercise 2:

Multilevel queue preemptive algorithm

	READY QUEUEs					
T	Q3:FCFS	Q2:RR q=3	Q1:RR q=2	Q0:RR q=1	CPU	Events
0				C(12) - B(8)	A(3)	Arrive A, B & C
1				A(3) - C(12)	B(7)	
2				B(7) - A(3)	C(11)	
3				C(11) - B(7)	A(2)	
4			A(2)	C(11)	B(6)	
5			B(6) - A(2)		C(10)	
6			C(10) - B(6)		A(1)	
7			C(10) - B(6)		A(0)	Finish A
8			C(10)		B(5)	
9			C(10)		B(4)	
10			B(4)		C(9)	
11			B(4)		C(8)	
12			C(8)		B(3)	
13			C(8)		B(2)	
14		B(2)			C(7)	
15		B(2)			C(6)	
16		C(6)			B(1)	
17		C(6)			B(0)	Finish B
18					C(5)	
19					C(4)	
20					C(3)	
21					C(2)	
22					C(1)	
23					C(0)	Finish C
24						
25						
26						
27						
28						

	Tiempo Retorno	Clase
P1	8	Clase 1
P2	18	Clase 2
P3	24	Clase 2

- Average Turnaround time
- Ending class

Process	Arrival time	Execution Profile
P1	0	4 CPU
P2	0	8 CPU
P3	0	12 CPU

Exercise

- Write the strings that prints the program in the terminal after its execution. Explain your answer.

Note: delay (n) performs a delay of n milliseconds; rand() return a random value range 0.0 .. 1.0

```
void * f1(void * arg) {
    printf("f1\n");
    delay(4000+rand()%1000);
    printf("Thread 1 text\n");
    return 0;
}
```

```
void * f2(void * arg) {
    printf("f2\n");
    delay(4000+rand()%1000);
    printf("Thread 2 text\n");
    return 0;
}
```

```
int main (void) {
  pthread_t th1,th2, th3, th4;
  pthread_attr_t attrib;

  pthread_attr_init( &attrib );

  printf(“Start Process ...\n”);
  pthread_create( &th1, &attrib, f1, NULL);
  pthread_create( &th2, &attrib, f2, NULL);
  pthread_create( &th3, &attrib, f1, NULL);
  pthread_create( &th4, &attrib, f2, NULL);
  //delay(5000+rand()%1000);
  exit(0);
}
```

Exercise

- Let’s consider two executable files F1 and F2, obtained after compiling the following C programs:

Info. The command "date +%S" prints on the screen the seconds portion of the current time. For example if the current time is 20:30:12, then this command will print "12".

/** F1.c **/	/** F2.c **/

#include <…>
 	
int pid;
 	
main(){
  pid=fork();
  if (pid==0)
    {sleep(3);}
  execl("./F2","F2",NULL);
}
	#include <…>
	
	int pid;
	
	main(){
	  pid=fork();
	  if (pid==0)
	    { sleep(1);}
	  execl("/bin/date","date","+%S",NULL);
	}

Suppose that the executables F1 and F2 are in the current working directory and that they run without errors. Answer the following questions:

Exercise

- Let’s consider two executable files F1 and F2, obtained after compiling the following C programs:

Info. The command "date +%S" prints on the screen the seconds portion of the current time. For example if the current time is 20:30:12, then this command will print "12".

Suppose that the executables F1 and F2 are in the current working directory and that they run without errors. Answer the following questions:

```
/** F1.c **/
#include <…>

int pid;

main(){
  pid=fork();
  if (pid==0)
     {sleep(3);}
  execl("./F2","F2",NULL);
}
```

```
/** F2.c **/
#include <…>

int pid;

main(){
  pid=fork();
  if (pid==0)
     { sleep(1);}
  execl("/bin/date","date","+%S",NULL);
}
```

a) How many processes are created when running the command ". / F2" and what is their parent ship.

b) What will be the output on the screen when running the command “./F2” at 09:10:25.

c) How many processes are created when running the command ". / F1" and what is their parent ship.

d) What will be the output on the screen when running the command “./F1” at 09:10:25.

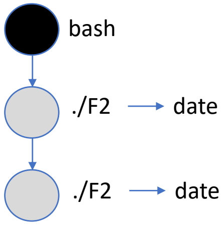
Exercise

Suppose that the executables F1 and F2 are in the current working directory and that they run without errors. Answer the following questions:

```
/** F1.c **/
#include <…>

int pid;

main(){
  pid=fork();
  if (pid==0)
    {sleep(3);}
  execl("./F2","F2",NULL);
}
```

```
/** F2.c **/
#include <…>

int pid;

main(){
  pid=fork();
  if (pid==0)
    { sleep(1);}
  execl("/bin/date","date","+%S",NULL);
}
```

a) How many processes are created when running the command ". / F2" and what is their parent ship.

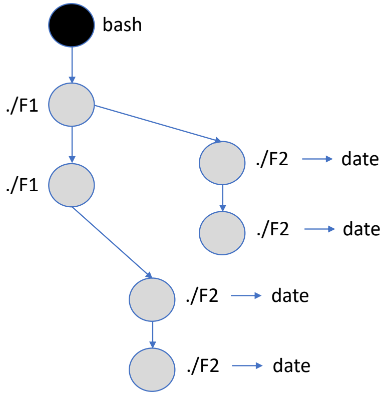


b) What will be the output on the screen when running the command "./F2" at 09:10:25.

25

26

c) How many processes are created when running the command ". / F1" and what is their parent ship.



d) What will be the output on the screen when running the command "./F1" at 09:10:25.

25

25

26

28

29

Exercise

- ¿Which of the following elements belong to the operating system kernel and which don’t?

	kernel	application	device
bash			
file manager			
disk controller			
file browser			
internet browser			
ps command			
process manager			
system call interface			
clock interrupt handler			
usb driver			

Exercise

- Consider that there is a file "hello.txt" in the current working directory, containing the text "hello \n", so that command "cat hello.txt" prints a line with the word **hello**.

Assume that the following program is compiled and executed. (executable **a.out**) Write the output when it is invoked:

$ ./a.out
$ ./a.out 1
$ ./a.out 2
$ ./a.out 3
$ ./a.out 4
$ ./a.out 5
$ ./a.out 12

```
#include <...all headers...>

int main(int argc, char *argv[]) {
    int X;
    int val = 0;
    int parent_pid = getpid();

    if (argc >1) X = atoi(argv[1]);
    else  X = 0;

    printf("X= %d\n", X);
    if (X >= 3) val = fork();
    if (val == 0) {
        if (X%2 == 1) {
            execl("/bin/cat", "cat", "hello.txt", NULL);
            printf("Err: /bin/cat generates error\n");
        }
        else {
            execl("/cat/bin", "cat", "hello.txt", NULL);
            printf("Err: /cat/bin/cat generates error\n");
        }
    }
    if (getpid() == parent_pid)
        printf("parent\n");
    else
        printf("child\n");
    sleep(2);
    return 0;
}
```

Exercise

- Consider that there is a file "hello.txt" in the current working directory, containing the text "hello \n", so that command "cat hello.txt" prints a line with the word **hello**.

Assume that the
program is comp
executed. (execu
Write the outpu
invoked:

```
#include <...all headers...>

int main(int argc, char *argv[]) {
    int X;
    int val = 0;
    int parent_pid = getpid();

    if (argc >1) X = atoi(argv[1]);
    else  X = 0;

    printf("X= %d\n", X);
    if (X >= 3) val = fork();
    if (val == 0) {
        if (X%2 == 1) {
            execl("/bin/cat", "cat", "hello.txt", NULL);
            printf("Err: /bin/cat generates error\n");
        }
        else {
            execl("/cat/bin", "cat", "hello.txt", NULL);
            printf("Err: /cat/bin/cat generates error\n");
        }
    }
    if (getpid() == parent_pid)
        printf("parent\n");
    else
        printf("child\n");
    sleep(2);
    return 0;
}
```

```
$ ./a.out
$ ./a.out 1
$ ./a.out 2
$ ./a.out 3
$ ./a.out 4
$ ./a.out 5
$ ./a.out 12
```

```
$ ./a.out
X= 0
Err: /cat/bin/cat generates error
parent
$ ./a.out 1
X= 1
hello
$ ./a.out 2
X= 2
Err: /cat/bin/cat generates error
parent
$ ./a.out 3
X= 3
parent
hello
$ ./a.out 4
X= 4
parent
Err: /cat/bin/cat generates error
child
$ ./a.out 5
X= 5
parent
hello
$ ./a.out 12
X= 12
parent
Err: /cat/bin/cat generates error
child
```