UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Departamento de Informática de Sistemas y Computadoras (DISCA)

fSO

Assessment
November, 7th, 2011

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

SURNAMES		**NAME**		Grup
DNI		**Signature**		

- **Do not unclip the sheets.**
- **Put your answer inside the answer reserved space.**
- **Write clearly and understandable. Make your answers brief and precise.**
- **The exam has 9 questions, the <u>6th question</u> scores 2 (1.2+0.8) points and the remaining questions 1 point every one.**

Explain what is multiprogramming and which its advantages are. Analyze if it has sense in case of a workload made of jobs with only one CPU burst and no I/O bursts. Suppose that there is no user machine interaction in the system.

1

Multiprogramming is the operating system ability to run multiple processes concurrently with a single CPU. So process CPU bursts of one process are mixed with I/O bursts from another process. This implies that such processes must be at the same time in memory. The main advantage is an increase in system efficiency and CPU utilization

If processes have only CPU bursts, there can be no concurrency between CPU and I/O, therefore it would not increase CPU utilization. As there must be context switches for processes to run concurrently, such kind of process mix only would produce a decrease in throughput.

Multiprogramming does not imply user-machine interaction, therefore, the concurrent execution of multiple processes is not perceived by the user and only makes sense if it increases CPU utilization and throughput.

Given a computer with an operating system:

a) Justify the requirement of having at least two execution modes available in the CPU.
b) Select for every one of the following actions which ones will be performed in kernel mode and which ones in user mode.

2

a) The main objective of having several execution modes is protecting hardware access. User programs will require the intervention of the operating system, to access the hardware they must request services via system calls.
This prevents users programs to access freely to main memory, to monopolize the CPU, to access certain records in the system and to access directly to system devices.
In the processor instruction set, a subset of instructions can only be executed in privileged mode

ACTIONS	**Kernel**	**User**
To program the disk controller	X	
To select a process from the ready queue	X	
To make a system call		X
To read 256 bytes in a file	X	
To execute instructions that compute complex math expressions		**X**

Given the following process state transitions justify if they are possible or not, in case of being possible describe a particular situation when it happens:

3	**From execution to ready** It is possible. It can happen in cases when the running process has not finished its CPU burst and the actual assigned CPU quatum has finished. It may also occur on systems with preemptive schedulers, when a new process arrives to the ready queue with higher priority than the one that has the CPU assigned.
	From ready to suspended It is not possible. If the process does not request I/O or to be suspended until an event trigering, calling system calls or executing instructions, and that means to be running, it cannot pass to suspended.
	From suspended to execution It is not possible. From suspended a process must pass to ready and be selected by the scheduler in order to go to execution. The only case to consider would be the one in which there is only one process in the system and in this case it would be the OS

Given the following code which executable file is named "Ejemplo1".

```
/*** Ejemplo1***/
#include "all_required_headers.h"

main()
{ int pid, i=0;

  while (i<2)
  {
     switch((pid=fork()))
      { case (-1): {printf("Child creation error\n");
                    break;}
        case (0): {printf("Child %i created\n",i);
                   break;}
        default: {printf("Parent\n");}
      }
    i++;
  }
  exit(0);
}
```

Give an explained answer to:

a) The number of processes generated by its execution and the relationship between them.
b) What messages will appear on the screen if fork() call always succeeds.

4	*a)* A total of 4 processes are created when you run this code a parent that creates two children, child 0 (created with i = 0) and child 1 (created with i = 1). Furthermore child 0 becomes parent and creates a child process with i = 1.

	b) If fork()call suceeds the following messages will be printed: Child 0 created Child 1 created Parent Child 1 created Parent Parent

---

Given the following code which executable file is named "Ejemplo1".

```
   /*** Ejemplo1***/
 1 #include "all_required_headers.h"
 2
 3 int main()
 4 {
 5   int status;
 6   printf("Message 1: before exec()\n");
 7   if (execl("/bin/ps","ps","-la", NULL)<0)
 8      { printf("Message 2: after exec()\n");
 9        exit(1);}
10
11   printf("Message 3: before exit()\n);
12   exit(0);
   }
```

Give an explained answer to:

a) How many processes can be created along its execution, in what line number this creation is done and what message prints every one of them.
b) When will appear on the standard output "*Message 3: before exit()*".

5	*a)* A single process is created when running this code. This process will print the message on screen "message 1: before execl()" Then the call to exec() is done that replaces the inherited code above by the one contained in the executable file /bin/ps. If the file /bin/ps exists it will appear on screen the result of executing ps.If the file /bin/ps doesn't existst then it will be displayed on screen "Message 2: after exec()"
	b) This message is not displayed never, since there are only two possibilities: * Execl() is successful, in this case this code has been replaced by the one contained in the /bin/ps file * Execl () is not successful, then it will execute exit(1) in line 9 and it will end the process

---

A time sharing OS accepts jobs sent from local terminals (I), from a network connection (R) and batch jobs (B). The job scheduling is done by a multilevel queue without feedback scheduler. The inter-queue scheduling policy is preemptive priority. The queue with greatest priority is the one associated to interactive jobs (I) and the queue with least priority is the one for batch jobs (B). Every jog toes to its corresponding queue type and stays there along its whole lifetime. The scheduling policies are:

- Interative: R-R (q=1)
- Network: R-R (q=2)
- Batch: FCFS

The job arrival order to the queues is the following: first the new ones, next the ones coming from I/O

and finally the ones coming from CPU. Supponse that all the I/O operations are done in a single I/O device the FCFS policy. It is asked to execute the following job group:

Job	Arrival time	Type	
T1	0	I	2CPU+2E/S+1CPU+2E/S+2CPU
T2	2	R	4CPU+1E/S+ 4CPU+1 E/S+1 CPU
T3	4	B	1CPU+1E/S+ 5CPU+1 E/S+1CPU
T4	5	I	2CPU+4E/S+1CPU
T5	21	B	2CPU

a) Obtain the workload execution diagram, filling the following table for every time instant.
b) Compute the mean waiting time in ready queue, the mean turnaround time and the CPU utilization.

6 a

T	Cola I	Cola R	Cola B	CPU	Cola E/S	E/S	Evento
0	(T1)			T1			Llega T1
1				T1			
2		(T2)		T2		T1	Llega T2 y T1 a E/S
3				T2		T1	
4	(T1)	T2	T3	T1			Llega T3
5		(T2)	T3	T2		T1	T1 a E/S
6	(T4)	T2	T3	T4		T1	Llega T4
7	T4,(T1)	T2	T3	T1			
8	T1 (T4)	T2	T3	T4			
9	(T1)	T2	T3	T1		T4	
10		(T2)	T3	T2		T4	Fin de T1
11			(T3)	T3	T2	T4	
12				-	T3,T2	T4	
13	(T4)			T4	T3	T2	
14		(T2)	T3	T2		T3	Fin de T4
15			T3	T2			
16		(T2)	T3	T2			
17			T3	T2			
18			T3	T3	(T2)	T2	
19		(T2)	T3	T2			
20			(T3)	T3			Fin de T2
21			T5	T3			Llega T5
22			T5	T3			
23			T5	T3			
24			(T5)	T5		T3	
25			T3	T5			
26			(T3)	T3			Fin T5
27							Fin T3
28							

6 b

Mean waiting time = 0+5+14+1+3/5 = 23/5 = 4.6
Mean turnaround time =
((10-0)+(20-2)+(27-4)+(14-6)+(26-21))/5=10+18+23+8+5/5 = 12.8
CPU utilización = 25/26

Given the critical section problem and its possible solutions, indicate if the following sentences are true(T) or false(F):

7	T/F	
	T	The set of instructions of a process that access data shared, and possibly written, by at least another process in the system, is called critical section.
	F	A correct solution to the critical section problem has to guarantee mutual exclusion of the whole code of processes being executed concurrently.
	T	A proposed solution to the critical section problem uses instruction test_and_set that has to execute atomically.
	F	Atomic execution means that it cannot be executed with active waiting instead it requires a mutex intervention.
	F	The condition that any valid solution to the critical section has to meet, and that is enunciated as: "*if no process is executing its critical section and there are processes that want to enter their critical sections, then the decision about which process will enter has to be taken in a finite time and only depends on the processes that want to enter*" it is called **limited waiting**.
	T	The critical section problem solutions based on active waiting are featured by processor underutilization.
	T	POSIX *mutex* are a solution to the critical section problem without active waiting

Given the producer consumer problem studied in class, where there are both a critical section and precedence relation. Remember that consumers should not consume before producers have produced. Complete the following code doing operations over semaphores *mutex*, *empty* and *full*, that are already declared and initialized. Explain your solution pointing to the utility of the proposed operations.

```
#include <semaphore.h>
#define N  20
int  buffer[N];
int  input, output, counter
sem_t mutex, full, empty;
```

```
void *func_prod(void *p) {
  int item;
  while(1) {
    item = produce()
//Complete (1)
……..
    buffer[input] = item;
    input = (input + 1) % N;
    counter = counter + 1;
……
//Complete (2)
  }
}
```

```
void *func_cons(void *p) {
  int item;
  while(1) {
//Complete (3)
    ……
    item = buffer[output];
    output = (output + 1) % N;
    counter = counter - 1;
    ……
//Complete(4)
    consume(item);
  }
}
```

```
void main ()
{
……
sem_init(&mutex,0,1);
sem_init(&vacio,0,N);
sem_init(&lleno,0,0);
…
}
```

8

Proposed solution

```
(1) sem_wait(&empty);sem_wait(&mutex)
(2) sem_post(&mutex);sem_post(&full)
(3) sem_wait(&full);sem_wait(&mutex)
(4) sem_post(&mutex);sem_post(&empty)
```

a) Utility of every proposed instruction

```
(1) sem_wait(&empty);sem_wait(&mutex) producer input protocol
(2) sem_post(&mutex);sem_post(&full) producer output protocol
(3) sem_wait(&full);sem_wait(&mutex) consumer input protocol
(4) sem_post(&mutex);sem_post(&empty) consumer output protocol
```

Full and empty semaphores control sequencing between producers and consumers so that producers do not generate data if there are no vacants and consumers do not consume if there are no items. The semaphore **mutex** controls access to the critical section.

Write the strings printed in the screen by the following program, explaining your answer.

```
void *function_thread1(void * arg)
{
  sleep(40);
  printf("I am child 1\n");
  return null;
}
```

```
void *function_thread2(void * arg) {
  sleep(50);
  printf("I am child 0\n");
  return null;
}
```

```
int main (void) {
  pthread_t th1,th2;
  pthread_attr_t atrib;

  pthread_attr_init( &atrib );

  printf("Once upon a time there were two threads...\n");
  pthread_create( &th1, &atrib, function_thread1, null);
  pthread_create( &th2, &atrib, function_thread2, null);
  exit(0);
}
```

9

```
Only prints:
"Once upon a time there were two threads..."
```

As the main thread does not wait for the other threads with pthread_join, it is very likely that the main thread be completed before the other threads be no longer suspended by sleep