UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Informática de Sistemas y Computadoras (DISCA)

fSO

EEE1: Ejercicio de Evaluación

November, 3rd 2014

etsinf Escola Tècnica Superior d'Enginyeria Informàtica

SURNAME		**NAME**		Group
IDN		**Signature**		E

- **Keep the exam sheets stapled.**
- **Write your answer inside the reserved space.**
- **Use clear and understandable writing. Answer in a brief and precise way.**
- **The exam has 9 questions, everyone has its score specified.**

1. Considering the concept of system calls and the Unix shell that works with internal and external commands, answer to the following sections as brief and precise as possible:
   a) Describe what are internal and external command in the Unix shell, and how they are implemented.
   b) Explain what a system call is, how they are used by applications and if they are necessary to implement internal and external shell commands.
   c) Suppose you are working with the Unix Shell (bash), indicate what are the name of the file/s that contain executable code for every one of the following commands:

```
cd /usr/bin
ls –la | more
./MyCopy  fic1  fic2
```

NOTE: **cd** is an internal command; **`cp, ls, more`** are external commands; and ***`MyCopy`*** is an scripts with execution permissions that contains one line “`cp $1 $2`”.

(0.4+0.3+0.3=1.0 points)

1	**a)**
	b)
	c)

2. Given the following C code, that produces the executable file named “prog” and considering that **COND** can be defined as “= =” or “>”, answer the following items:

```
/***** Source code for prog.c ***/
#include <all required headers>

int main() {
  pid_t pid;
  int i;

  for (i=0; i<3; i++){
    pid = fork();
    if (pid COND 0){
      printf("PID_IF = %d, PPID_IF = %d \n", getpid(), getppid());
      sleep(5);
      break;
    }
    printf("PID_FOR = %d, PPID_FOR = %d \n", getpid(), getppid());
  }
  sleep(5);
  return(0);
}
```

a) Suppose that "prog" is executed in such a way that its parent PID is 4000, and its own PID is 4001. Along its execution the system assigns to the processes that are created the sequence of PIDs: 4002, 4003, 4004, etc. Fill the following tables with the values that are shown in the screen with the strings PID_IF, PPID_IF, PID_FOR and PPID_FOR, considering that **COND** is defined as:

a1) `#define COND ==`

a2) `#define COND >`

b) Considering "`#define COND ==`" explain if there can be zombie or orphan processes, and if so how many of each type can appear.

(0.4+0.4+0.4=1,2 points)

2

a1) `#define COND ==`

PID_IF	PPID_IF	PID_FOR	PPID_FOR

a2) `#define COND >`

PID_IF	PPID_IF	PID_FOR	PPID_FOR

b) Zombie and orphan processes with “`#define` **`COND`** `==`”

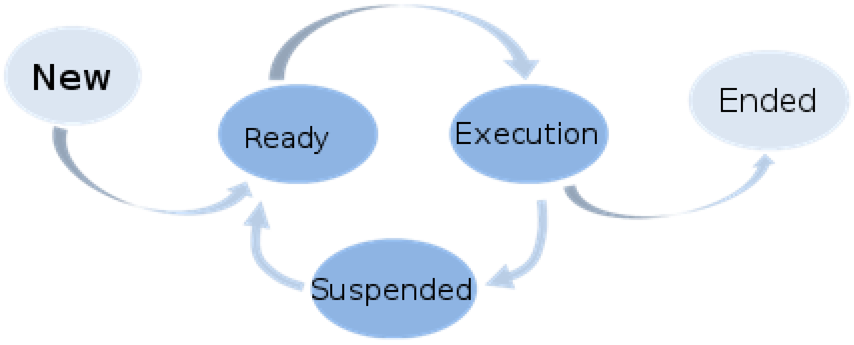
3. Indicate for each of the following items if it is a component of the operating system or an operating system utility

NOTE. Two errors void a correct answer.

(0.7 points)

3	**OS component**	**OS utility**	**Element**
			Process scheduler
			Command Shell
			Text editor
			Compiler
			Virtual memory manager
			System monitorization tools
			I/O device driver

4. A given operating system has the following diagram of possible states and transitions for processes:



Explain if this operating system could be working with a preemptive scheduler.

(0.5 points)

4

5. Consider two executable files named “compare” and “same” located in the current working directory and whose source codes are shown below:

```
// Program compare.c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char * argv[]){
  int pid;int status;
  if (argc != 3){
    printf("compare: wrong arguments\n");
    exit(-1);
  }
  pid=fork();
  if (pid==0){
    execl("./same","same",argv[1],argv[2],NULL);
    exit(-2);
  }
  wait(&status);
  if(status==0){
    printf("Text A\n");
    exit(0);
  }
  printf("Text B\n");
  exit(0);
}
```

```
// Program same.c

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char * argv[]){
  int a,b;
  if (argc != 3){
    printf("same: wrong \n");
    exit(255);
  }
  a=atoi(argv[1]);
  b=atoi(argv[2]);

  if (a==b) exit(0);

  exit(1);
}
```

The *atoi* function returns a numerical value of type “*int*” from a string (*char* *), so *atoi("215")* returns the integer value 215.

Indicate:

- the total number of created processes, and
- the text lines shown in the terminal after every one of the following executions:

a) `./compare 3 3`
b) `./compare 6 3`

(0.5+0.5=1.0 point)

5	a) `./compare 3 3`
	b) `./compare 3 6`

6. A CPU scheduler has 3 queues (S, U1, U2), managed with an inter queue policy of preemptive priorities, being S the one with the highest priority and U2 the one with the lowest priority. Queue S receives only server processes that remain in this queue until ending, but user processes are assigned to queues U1 and U2. When a user process enters the system it is located on queue U2 and it remains in this queue until its first I/O operation is completed, after that it promotes to queue U1 where it will remain until ending its execution. The system has a single I/O device managed with a FCFS policy. Every CPU queue is managed with the following policies:

S:FCFS **U1: Round Robin with q=1** **U2: Round Robin with q=2**

Five processes arrive to the system with the following features:

Process type	Process	Execution profile	Arrival instant
Server	Sa	1 CPU + 4 I/O + 1 CPU+1 I/O + 1 CPU	5
Server	Sb	1 CPU + 2 I/O + 1 CPU	7
User	Uc	4 CPU + 2 I/O + 2 CPU	0
User	Ud	4 CPU + 2 I/O + 3 CPU	1
User	Ue	3 CPU + 1 I/O + 1 CPU	3

a) Obtain the time line of CPU, I/O and queues process allocation (1.3 points)

T	FCFS Queue S	RR (1) Queue U1	RR (2) Queue U2	CPU	Queue I/O	I/O	Comments
0							Uc arrival
1							Ud arrival
2							
3							Ue arrival
4							
5							Sa arrival
6							
7							Sb arrival
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							

b) Compute the mean waiting time, the mean turnaround time and the CPU utilization.

(0.75 points)

6 **b)**

Mean waiting time =

Mean turnaround time =

CPU utilization =

7. Given the following code, use semaphores with the notation proposed by Dijkstra (P and V operations) to ensure that following running threads execute functions whose name begins with "sequence-" according to the order that suggest the numbers employed in such names. If there are multiple functions with the same name (including the number at the end), none should start before the end of the functions with the previous name that must be completed (all of them) before the start of any function with the following name. Furthermore, calls to function "sc()" must be done in mutual exclusion. Declare and initialize the semaphores required to accomplish the described behavior.

1.0 point

7 *Declare and initialize the required semaphores*

THREAD 1	THREAD 2	THREAD 3

`cs();`
`sequence1();`
`cs();`
`sequence3();`
	`cs();`
	`sequence1();`
	`cs();`
	`sequence3();`
		`cs();`
		`sequence2();`
		`sequence4();`

8. The following program, whose executable code has been generated with name "threads", processes the strings passed as arguments, showing the result on the standard output.

```
/** Program threads.c **/
#include <all required headers>

#define MAX_THREADS 100
pthread_t thread_p[MAX_THREADS];
pthread_t thread_m[MAX_THREADS];

pthread_attr_t atr;

void *Print(void* ptr){
  char *str= (char*) ptr;
  int longx;
  longx= strlen(str); //String length
  sleep(longx);
  printf("%s\n",str);
}

void *UpperC(void* ptr){
  char *str= (char*) ptr;
  int longx;
  int i;
  longx= strlen(str); //String length
  for(i=0;i<longx;i++)
    str[i]= str[i]-32; //Uppercase conv.
  sleep(2);
}
```

```

int main(int argc, char *argv[]){
  int i, h=0;

  pthread_attr_init(&atr);

  for (i=1; i<argc; i++){
    pthread_create(&thread_p[i],&atr,Print,argv[i]);
  }
  for (i=1; i<argc; i++){
    pthread_create(&thread_m[i],&atr,UpperC,argv[i]);
    pthread_join(thread_m[i], NULL);
  }
  pthread_exit(NULL);
}
```

Considering that the execution of “threads” is done with the following command line:

```
$./threads dont unstaple the sheets
```

Answer the following:

a) What is the maximum number of threads that will run concurrently?

b) Why is it necessary to call *pthread_exit(NULL)* at the end of `main` ? What can happen if it is not called?

c) Suppose that the time consumption of “threads” is determined exclusively by the waiting introduced by sleep() calls, that is, consider that the rest of the instructions do not consume a significant amount of time. Considering the command below, indicate at what time instants printing is completed and at what time instants is completed the conversion to uppercase of each one of the 4 strings that are passed as arguments, naming them as "arg1" to "arg4". Suppose that “threads” execution begins at t = 0.

(0.4+0.4+0.5=1.3 points)

8	*a)*
	b)

c) **`$./threads dont unstaple the sheets`** // Execution starts at t = 0

9. Answer as much briefly and precisely as possible to the following questions:

(1.25 points)

9

a) How do you define a concurrent program?

b) What is a race condition?

c) What is a critical section?

d) With what requirements must comply a protocol that protects critical sections?

e) What is busy waiting synchronization?