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

fSO

EEE1: Ejercicio de Evaluación
3 de Noviembre de 2014

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

APELLIDOS		**NOMBRE**		Grupo
DNI		**Firma**		

- **No desgrape las hojas.**
- **Conteste exclusivamente en el espacio reservado para ello.**
- **Utilice letra clara y legible. Responda de forma breve y precisa.**
- **El examen consta de 9 cuestiones, cuya valoración se indica en cada una de ellas.**

---

1. Teniendo en cuenta el concepto de llamadas al sistema y que el shell de Unix trabaja con órdenes, internas y externas, conteste de forma razonada y concisa a los siguientes apartados:
   a) Describa brevemente que son las órdenes internas y externas del Shell de Unix, como se implementan.
   b) Indique brevemente que es una llamada al sistema, en qué forma se ofrecen a las aplicaciones que quieren hacer uso de ellas y si son necesarias para implementar las órdenes internas y externas del shell.
   c) Suponga que trabaja con el Shell de Unix (programa bash) e indique cuáles son los nombre de archivos/s que contiene/n el código ejecutable de cada una de las siguientes órdenes:

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

   NOTA: **cd** es una orden interna; **cp, ls, more** son órdenes externas; y ***MiCopia*** es un script con permisos de ejecución cuyo contenido es "`cp $1 $2`".

(0.4+0.3+0.3=1.0 puntos)

1	**a)**
	b)
	c)

---

2. Dado el siguiente código fuente en C, cuyo ejecutable se denomina "prog" y considerando que **COND** puede ser definido como "= =" o ">", conteste a las siguientes propuestas

EEE1: Ejercicio de Evaluación
3 de Noviembre de 2014

```
/***** Codigo fuente de prog.c ***/
#include <todos los .h necesario>

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) Suponga que se ejecuta "prog" y su padre tiene PID 4000, mientras que el PID de "prog" es 4001 y que durante su ejecución el sistema asigna a los procesos que se crean los PIDs de forma consecutiva (4002, 4003, etc.). Indique en las tablas adjuntas los valores que se visualizarán durante su ejecución junto a las cadenas PID_IF, PPID_IF, PID_FOR and PPID_FOR, considerando que **COND** está definido como:
   a1) `#define COND ==`
   a2) `#define COND >`

b) Indique si podrían aparecer procesos zombies y/o huérfanos y en su caso cuántos procesos zombies y/o huérfanos podrían aparecer si "prog" se ejecuta con "`#define COND ==`".

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

2

a1) `#define COND ==`

PID_IF	PPID_IF	PID_FOR	PPID_FOR

a2) `#define COND >`

PID_IF	PPID_IF	PID_FOR	PPID_FOR

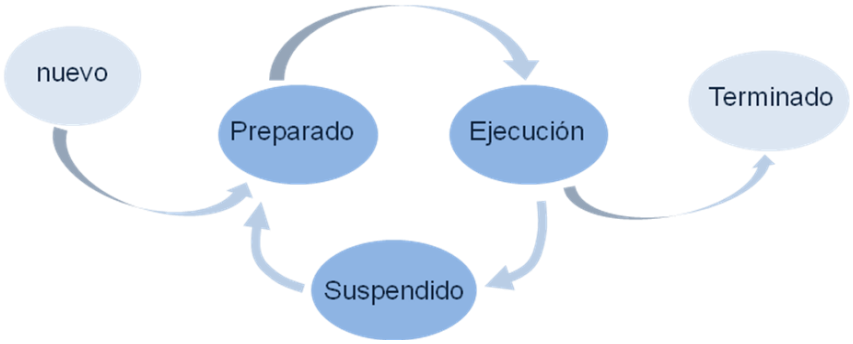
b)Proceso zombies y huérfanos con "`#define COND ==`"

3. Indique para cada uno de los siguientes elementos si es un componente del sistema operativo o una utilidad del sistema operativo.
Nota: Dos errores descuenta un acierto

(0.7 puntos)

3	Componente del S.O.	Utilidad del S.O.	Elemento
			Planificador de procesos
			Interprete de órdenes
			Editor
			Compilador
			Gestor de memoria virtual
			Herramienta de monitorización del sistema
			Manejador o driver de dispositivo de E/S

4. Los procesos de cierto sistema operativo tienen el siguiente diagrama de posibles estados y transiciones.



Indique de forma justificada si el sistema operativo podría estar trabajando con un planificador expulsivo.

(0.5 puntos)

4	

5. Considere dos archivos ejecutables de nombre *compare* y *same* que se encuentran en el directorio actual de trabajo y cuyos códigos fuente se muestran a continuación:

// Programa 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);
 }
```

//Programa 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);
}
```

Recuerde que la función *atoi* devuelve el valor numérico (*int*) leído en un argumento de tipo (*char* *). Es decir, *atoi("215")* devuelve el valor numérico 215.

Indique:

- el número total de procesos creados,
- los mensajes o líneas que se muestran en el terminal después de cada ejecución

si se ejecuta *compare* con los siguientes parámetros:

a. ./compare 3 3

b. ./compare 3 6

(0.5+0.5=1.0 puntos)

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

6. Un sistema dispone de un planificador a corto plazo con 3 colas (S, U1, U2), gestionadas con prioridades expulsivas siendo S la cola más prioritaria y U2 la menos prioritaria. Los procesos servidores siempre llegan a cola S y permanecen en ella hasta finalizar su ejecución, mientas que los procesos normales de usuario utilizan las colas U1 y U2. Los procesos nuevos de usuario llegan a la cola U2 y permanecen en esta cola hasta que terminan su primera operación de E/S, momento en el que promocionan a la cola U1, en la que permanecerán hasta finalizar su ejecución. El sistema está dotado de un único dispositivo de E/S gestionado con FCFS. Los algoritmos de planificación que rigen cada una de las colas son los siguientes:

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

A dicho sistema llegan 5 procesos cuyo perfil de ejecución e instante de llegada son:

Perfil proceso	Proceso	Perfil de ejecución	Instante de llegada
Servidor	Sa	1 CPU + 4 E/S + 1 CPU+1 E/S + 1 CPU	5
Servidor	Sb	1 CPU + 2 E/S + 1 CPU	7
Usuario	Uc	4 CPU + 2 E/S + 2 CPU	0
Usuario	Ud	4 CPU + 2 E/S + 3 CPU	1
Usuario	Ue	3 CPU + 1 E/S + 1 CPU	3

a) Represente mediante el diagrama temporal la ocupación de la CPU, del periférico de E/S y de las diferentes colas en cada unidad de tiempo. (1.3 puntos)

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

b) Indique cuál es el tiempo medio de espera, el tiempo medio de retorno y la Utilización de la CPU.

(0.75 puntos)

6	**b)** Tiempo medio de espera= Tiempo medio de retorno = Utilización de la CPU=

---

7. **D**ado el siguiente código, utilice semáforos con la notación propuesta por Dijkstra (P y V para las operaciones) para garantizar que los diferentes hilos ejecuten las funciones cuyo nombre empieza por "secuencia-" según el orden que sugieren los números empleados en tales nombres. Si hay varias funciones con el mismo nombre, ninguna de ellas debe empezar antes de que termine la función con el nombre anterior y todas ellas deben haber terminado antes de que empiece la función con el nombre siguiente. Además, ha de asegurarse que las funciones "sc()" se ejecuten en exclusión mutua. Indique qué valor inicial deberán tener los semáforos que haya utilizado.

1.0 puntos

7 *Declare e inicialice los semáforos*

HILO 1	HILO 2	HILO 3
`sc();`	`sc();`	`sc();`
`secuencia1();`	`secuencia1();`	`secuencia2();`
`sc();`	`sc();`	`secuencia4();`
`secuencia3();`	`secuencia3();`	

8. El siguiente programa, cuyo código ejecutable ha sido generado con el nombre "hilos", procesa las cadenas de caracteres que se le pasan como argumentos, mostrando el resultado por la salida estándar.

```
/**Programa hilos.c **/
#include <todos los .h necesarios>

#define MAX_HILOS 100
pthread_t hilo_p[MAX_HILOS];
pthread_t hilo_m[MAX_HILOS];

pthread_attr_t atr;

void *Print(void* ptr)
{ char *str= (char*) ptr;
  int longx;
 longx= strlen(str); //Longitud cadena

  sleep(longx);
  printf("%s\n",str);
}

void *Mayus(void* ptr)
{ char *str= (char*) ptr;
  int longx;
  int i;
  longx= strlen(str); //Longitud  cadena
 for(i=0;i<longx;i++)
  str[i]= str[i]-32; //Pasa a mayúsculas
  sleep(2);
}
```

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

  pthread_attr_init(&atr);

  for (i=1; i<argc; i++, h++){

    pthread_create(&hilo_p[h],&atr,Print,argv[i]);
  }
  for (i=1; i<argc; i++, h++){

    pthread_create(&hilo_m[h],&atr,Mayus,argv[i]);
    pthread_join(hilo_m[h], NULL);
  }
  pthread_exit(NULL);
}
```

Teniendo en cuenta que se invoca la ejecución de hilos con la siguiente línea de órdenes y argumentos:

`$./hilos no desgrape las hojas`

Indique de forma justificada

a) ¿Cuál es el número máximo de hilos que se ejecutaran concurrentemente?

b) ¿Por qué es necesario incluir la llamada *ptread_exit(NULL)* al final de la función `main` y que podría suceder si se omite?

c) Suponga que el consumo de tiempo de *hilos* viene determinado exclusivamente por las esperas introducidas mediante las llamadas *sleep()* , es decir, considere que el resto de las instrucciones no consumen tiempo. Indique en qué **instantes de tiempo** se completa **la impresión** y en que instantes **la conversión a mayúsculas** de cada una de las 4 cadenas de caracteres que se pasan como argumentos, nómbrelas como "arg1" a "arg4", si el instante en que se inicia la ejecución corresponde a t=0.

(0.4+0.4+0.5=1.3 puntos)

8	*a)*
	b)

c) `$./hilos no desgrape las hojas` //La ejecución comienza en t=0

9. **C**onteste de forma concreta y concisa a las siguientes preguntas:

(1.25 puntos)

9

a) ¿Cómo definiría un programa concurrente?

b) ¿Qué es una condición de carrera?

c) ¿Qué es una sección critica?

d) ¿Qué condiciones deben cumplir los protocolos de las secciones críticas?

e) ¿En qué consiste la espera activa?