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

fSO

etsinf Escola Tècnica Superior d'Enginyeria Informàtica

**EEE1: Ejercicio de Evaluación**

3 de Noviembre de 2014

| **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)**

**El shell trata dos tipos de ordenes:**

- **Ordenes internas: El shell incluye internamente el código que hace el trabajo de la orden**
- **Ordenes externas: el shell crea un proceso para ejecutar la orden. Se trata de un programa cuyo archivo se encuentra en el disco, como archivos con código binario (ejecutable), hechos en C y compilados. O bien son archivos Shell scripts, que contienen varias órdenes del Shell.**

**b)**

**Las llamadas al sistema son el mecanismo para solicitar servicios al Sistema Operativo. El S.O proporciona esta interfaz para poder acceder a los recursos hardware de la máquina.**
**Estos servicios se ofrecen en forma de Funciones de Biblioteca y permiten acceder a los recursos gestionados por el sistema operativo.**
**Dado que él Shell es una utilidad del sistema para hacer más fácil al usuario la interfaz con el sistema operativo, todas las órdenes del Shell utilizan una o más llamadas al sistema y esto no depende de si son internas o externas, sino del servicio que han de proporcionar. Por tanto ambos tipos de instrucciones utilizan llamadas al sistema.**

**c)**

**cd /usr/bin: es un orden interna, el código de cd se encuentra el archivo del Shell, es decir el archivo bash**
**ls –la | more : el archivo ls y el archivo more, suelen estar por defecto en /bin**
**MiCopia: el archivo MiCopia está en el directorio actual de trabajo, el archivo del Shell, es decir el del bash (para interpretar el script ) y el archivo /bin/cp**

---

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 ==`".

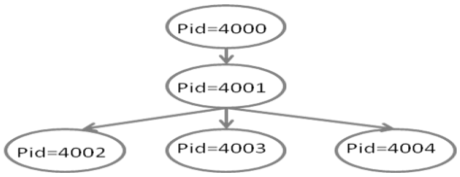
(0.4+0.4+0.4=1,2 puntos)

**2**

**a1)** `#define COND ==`

Se genera un abanico de procesos

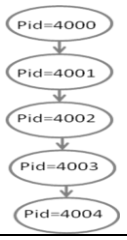
| PID_IF | PPID_IF | PID_FOR | PPID_FOR |
|---|---|---|---|
| 4002 | 4001 | 4001 | 4000 |
| 4003 | 4001 | 4001 | 4000 |
| 4004 | 4001 | 4001 | 4000 |




**a2)** `#define COND >`

Se genera una cadena de procesos

| PID_IF | PPID_IF | PID_FOR | PPID_FOR |
|---|---|---|---|
| 4001 | 4000 | 4002 | 4001 |
| 4002 | 4001 | 4003 | 4002 |
| 4003 | 4002 | 4004 | 4003 |




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

Se genera un abanico de procesos de tal manera que las sentencias dentro de "if (pid == 0)" sólo las ejecutan los procesos hijos, los cuales entran en suspensión durante 5 segundos y después salen del bucle para entrar nuevamente en suspensión durante 5 segundos más, en total 10 segundos.

El proceso inicial con PID 4001 ejecuta todas las iteraciones del bucle for, y antes de finalizar sleep(5). Además el padre no tiene la llamada wait() por lo tanto existe la posibilidad de que finalice antes que sus hijos generándose **3 huerfanos.**

Dado que el tiempo de ejecución de los hijos es mayor que el del padre es bastante improbable que se generen **procesos zombi**.

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 |
|---|---|---|---|
| | X | | Planificador de procesos |
| | | X | Interprete de órdenes |
| | | X | Editor |
| | | X | Compilador |
| | X | | Gestor de memoria virtual |
| | | X | Herramienta de monitorización del sistema |
| | X | | 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** | Planificador expulsivo es aquel que permite que un proceso sea expulsado de la CPU antes de terminar su ráfaga de CPU para que otro más prioritario la ocupe y pueda ejecutar sus instrucciones.<br>En el caso de la figura no puede pertenecer a un planificador expulsivo ya que sería necesario que la transición de en ejecución a preparado fuese posible. |
|---|---|

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:

**EEE1: Ejercicio de Evaluación**

3 de Noviembre de 2014

**// 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 6 3

(0.5+0.5=1.0 puntos)

**5**

a) ./compare 3 3

```
$ ./compare 3 3
Text A
Se crean dos procesos.
El primer proceso con la imagen de ./compare y ejecuta el código de compare que
crea un proceso hijo con la llamada fork().
El proceso hijo ejecuta la llamada
    execl("./same","same",argv[1],argv[2],NULL);
y por tanto ejecutara el código ./same.
EL código de terminación del hijo es 0 (exit(0)) el padre le espera en
wait(&status) e imprime "Text A".
```

b) ./compare 6 3

```
$ ./compare 6 3
Text B

Se crean dos procesos.
El primer proceso con la imagen de ./compare y ejecuta el código de compare que
crea un proceso hijo con la llamada fork().
El proceso hijo ejecuta la llamada
    execl("./same","same",argv[1],argv[2],NULL);
y por tanto ejecutara el código ./same.
EL código de terminación del hijo es 1 (exit(1)) el padre le espera en
wait(status) e imprime "Text B".
```

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** | | | | **Uc(3)** | | | **Llega Uc** |
| **1** | | | **Ud(4)** | **Uc(2)** | | | **Llega Ud** |
| **2** | | | **Uc(2)** | **Ud(3)** | | | |
| **3** | | | **Ue(3) Uc(2)** | **Ud(2)** | | | **Llega Ue** |
| **4** | | | **Ud(2) Ue(3)** | **Uc(1)** | | | |
| **5** | | | **Uc(1) Ud(2) Ue(3)** | **Sa(0)** | | | **Llega Sa** |
| **6** | | | **Uc(1) Ud(2)** | **Ue(2)** | | **Sa** | |
| **7** | | | **Ue(2) Uc(1) Ud(2)** | **Sb(0)** | | **Sa** | **Llega Sb** |
| **8** | | | **Ue(2) Uc(1)** | **Ud(1)** | **Sb** | **Sa** | |
| **9** | | | **Ue(2) Uc(1)** | **Ud(0)** | **Sb** | **Sa** | |
| **10** | | | **Ue(2) Uc(1)** | **Sa(0)** | **Sa Ud** | **Sb** | |
| **11** | | | **Ue(2)** | **Uc(0)** | **Sa Ud** | **Sb** | |
| **12** | | | **Ue(2)** | **Sb(0)** | **Uc Sa** | **Ud** | |
| **13** | | | | **Ue(1)** | **Uc Sa** | **Ud** | **Finaliza Sb** |
| **14** | | | **Ue(1)** | **Ud(2)** | **Uc** | **Sa** | |
| **15** | | **Ud(2)** | **Ue(1)** | **Sa(0)** | | **Uc** | |
| **16** | | | **Ue(1)** | **Ud(1)** | | **Uc** | **Finaliza Sa** |
| **17** | | **Ud(1)** | **Ue(1)** | **Uc(1)** | | | |
| **18** | | **Uc(1)** | **Ue(1)** | **Ud(0)** | | | |
| **19** | | | **Ue(1)** | **Uc(0)** | | | **Finaliza Ud** |
| **20** | | | | **Ue(0)** | | | **Finaliza Uc** |
| **21** | | | | | | **Ue** | |
| **22** | | | | **Ue(0)** | | | |
| **23** | | | | | | | **Finaliza Ue** |

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)**<br>Tiempo medio de espera= (0+0+9+7+15)/5=31/5=6,2<br>Tiempo medio de retorno = ((16-5)+(13-7)+(20-0)+(19-1)+(23-3))/5=(11+6+20+18+20)/5=75/5=15<br>Utilización de la CPU= 22/23=0, 956 |

---

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*
*mutex=1, S2 =0, S3=0, S4=0*

| HILO 1 | HILO 2 | HILO 3 |
|---|---|---|
| *P(mutex)*<br>**sc();**<br>*V(mutex)*<br>**secuencia1();**<br>*V(s2);*<br>*P(mutex);*<br>**sc();**<br>*V(mutex);*<br>*P(s3);*<br>**secuencia3();**<br>*V(S4);* | *P(mutex)*<br>**sc();**<br>*V(mutex)*<br>**secuencia1();**<br>*V(S2);*<br>*P(mutex);*<br>**sc();**<br>*V(mutex);*<br>*P(S3);*<br>**secuencia3();**<br>*V(S4);* | *P(mutex)*<br>**sc();**<br>*V(mutex)*<br>*P(s2); P(S2);*<br>**secuencia2();**<br>*V(S3); v(S3);*<br>*P(S4); P(S4);*<br>**secuencia4();** |

---

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 qué se ejecutarán concurrentemente?

b) ¿Por qué es necesario incluir la llamada *ptread_exit(NULL)* al final de la función `main` y qué 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)*

*Se crean un total de 4+4+1 (main) hilos.*
*Los 4 primeros corresponden a uno por cada argumento ("no" "desgrape" "las" "hojas") que ejecutan la función Print, estos se ejecutan concurrentemente entre ellos y con hilo main().*

*A continuación se crea un hilo por cada argumento, es decir 4, que ejecuta la función May pero se espera la finalización de cada hilo (pthread_join) antes de lanzar el siguiente, de forma que sólo hay un Hilo May en ejecución en cada momento.*
*Por lo que concurrentemente puede haber comomáximo 4+1+1=6 hilos*

b)

*La llamada pthread_exit asegura que si el hilo principal termina antes de que lo hagan los hilos Print, estos no serán eliminados antes de que concluyan su ejecución.*
*Si no se incluye pthread_exit() podrían no aparecer impresas algunas de las cadenas*

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

*Todos los hilos impresores se lanzan en el instante 0. La impresión se demora un número de segundos igual a la longitud de la cadena. Es decir, se imprime:*

*T=2 arg1 (no)*
*T=3 arg3 (las)*
*T=5 arg4 (hojas)*
*T=8 arg2 (desgrape)*

*La conversión se realiza ordenadamente, mediante hilos May que se suceden en el tiempo y esperan 2s tras cada conversión antes de finalizar. Por tanto, los tiempos de las conversiones son:*
*T=0 arg1 (no)*
*T=2 arg2 (desgrape)*
*T=4 arg3 (las)*
*T=6 arg4 (hojas)*

```
          no    las    hojas      desgrape
  0s-------2s-----3s------5s---------8s----

  0s-------2s---------4s-------6s------8s----

          NO      DESGRAPE   LAS     HOJAS
```

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

( 1.25 puntos)

**9**

a) ¿Cómo definiría un programa concurrente?

Es un único programa cuyo código al ejecutarlo está constituido por varias actividades concurrentes que pueden evolucionar de forma independiente. Dichas actividades trabajan en común para resolver un problema y se coordinan y comunican entre sí.

b) ¿Qué es una condición de carrera?

Una condición de carrera es una situación no deseable que provoca que un código que ejecutado secuencialmente es correcto deje de serlo al ejecutarlo concurrentemente.
Una condición de carrera se produce cuando la ejecución de un conjunto de operaciones concurrentes sobre una variable compartida deja a la variable en un estado inconsistente con las especificaciones de corrección. Además el resultado de la variable depende de la velocidad relativa en que se ejecuten las operaciones.

c) ¿Qué es una sección critica?

Son las zonas de código en las que se acceden para lectura o escritura a las variables compartidas por varios hilos o procesos. Y por tanto zonas de código en las que hay que sincronizar su ejecución.

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

El protocolo tiene que cumplir tres condiciones: Exclusión mutua, progreso y espera limitada.
Exclusión mutua: si un proceso está ejecutando su sección crítica, ningún otro proceso puede estar ejecutando la suya.
Progreso: si ningún proceso está ejecutando su sección crítica y hay otros que desean entrar a las suyas, entonces la decisión de qué proceso entrará a la sección crítica se toma en un tiempo finito y sólo depende de los procesos que desean entrar.
Espera limitada: Después de que un proceso haya solicitado entrar en su sección crítica, existe un límite en el número de veces que se permite que otros procesos entren a sus secciones críticas.

e) ¿En qué consiste la espera activa?
Son soluciones al problema de sincronización en las que l protocolo de entrada impide que un proceso entre en su sección crítica, haciendo que dicho proceso ejecute un **bucle vacío que consume tiempo de CPU**

Un ejemplo de protocolo a la espera activa es la instrucción test-and-set ()es una solución al problema de la sincronización, aunque ella por si sólo no garantiza espera limitada.