PEC2

Presentación

Esta PEC plantea una serie de actividades con el objetivo de que el estudiante se familiarice con la temática de los últimos módulos de la asignatura.

Competencias

Transversales

- Capacidad para la comunicación escrita en el ámbito académico y profesional

Específicas

- Capacidad para analizar un problema en el nivel de abstracción adecuado a cada situación y aplicar las habilidades y conocimientos adquiridos para abordarlo y resolverlo.

Enunciado

1. **[3 puntos, 1 punto por apartado]** Responder justificadamente las siguientes preguntas:

   a) Cuáles son las semejanzas y las diferencias entre un enlace simbólico (soft link) y un enlace físico (hard link). Mostrar las diferencias mediante un ejemplo real.

   b) Ejecutar los siguientes comandos de manipulación de fichero en una consola de Linux y explicar en qué consiste cada uno.

```
> mkdir Test
> touch Test/File1.txt
> ls -la Test
> chmod 600 Test
> ls -la Test
```

   ¿A que es debido el resultado mostrado por el último comando ls?

c) Mientras se ejecuta el siguiente programa, denominado Pgm1.c:

```
int main(int argc, char *argv[])
{
    int pid;
    pid = fork();
    if (pid<0) perror("Error in fork()");
    if (pid==0)
    {
         int Total=0, x;
         for(x=0;x<1000;x++)
             Total=+x;
         exit(0);
    }
    sleep(atoi(argv[1]));
    exit(0);
}
```

Hago un listado de mis procesos en un sistema Unix obtengo un proceso denominado “Pgm1 <defunct>” que yo no he ejecutado, ¿A qué es debido la aparición de este proceso? ¿Cuándo y cómo desaparecerá este proceso?

2. **[3 puntos, 1.5 por apartado]** Utilizando las llamadas al sistema Unix vistas en la asignatura, escribir los siguientes programas en lenguaje C:

   a) Una aplicación concurrente formada por N procesos hijos, en la cual cada proceso imprimirá por pantalla el pid del proceso padre, su pid y orden lógico de creación (un 1 el primer hijo, un 2 por el segundo, etc.). El proceso padre no puede finalizar hasta que los hijos hayan acabado.

   El proceso padre y los hijos tienen que utilizar ficheros ejecutables diferentes (Father2a.c y Child2a.c).

   b) Modificar la aplicación del apartado a, para que mediante la utilización de pipes se garantice que los hijos imprimen su orden lógico en orden decreciente.

3. **[4 puntos, 1 por apartado]** Tenemos una aplicación concurrente formada por N hilos de ejecución. Cada uno de estos hilos de ejecución reciben como parámetro un identificador que identifica el orden lógico en que se ha creado el hilo de ejecución (desde 1 hasta N). De momento, los hilos, únicamente imprimen su identificador por pantalla:

```
Thread(int id)
{
   charmsg;
   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));
}
```

Utilizando las siguientes operaciones de semáforos:

- `sem_init(semaphore s, int valor)`. Inicializa el semáforo s con nsems instancias iniciales (valor inicial).
- `sem_wait(s)`. Pide una instancia del semáforo s. Espera que el valor del semáforo sea mayor que 0 y cuando lo es lo decrementa de forma atómica.
- `sem_wait_mult(s,n)`. Pide N instancias del semáforo s. Espera que el valor del semáforo sea mayor que N-1 y cuando lo es lo decrementa en N de forma atómica.
- `sem_signal(s)`. Incrementa de forma atómica el valor del semáforo.
- `sem_signal_mult(s,n)`. Aumenta de forma atómica el valor del semáforo en N.

Se pide:

a) Asumiendo que se crean N hilos, garantizar con semáforos que los mensajes de los hilos se imprime en orden creciente (primero el mensaje del hilo 1, después el del hilo 2, etc.).

Implementar el código genérico que ejecutarán todos los hilos (*Thread*).

```
/* Declare and initialize semaphores and global variables */
```

```
Thread(int id)
{
   char msg;




   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));




}
```

b) Asumiendo que se crean N hilos, garantizar con semáforos que el hilo 2 no pueda mostrar su mensaje hasta que el hilo 1 lo haya mostrado y que el resto de hilos lo tengan que mostrar después del hilo2.

Implementar el código a ejecutar por los hilos 1 y 2 (*Thread1 y Thread2*) y el código del resto de hilos (*ThreadRest*).

```
/* Declare and initialize semaphores and global variables */
```

```
Thread1(int id)
{
   char msg;


   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));


}
```

```
Thread2(int id)
{
   char msg;


   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));


}
```

```
ThreadRest(int id)
{
   char msg;


   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));


}
```

c) Asumiendo que se crean N hilos, calcular de forma concurrente la suma de todos los identificadores de los hilos. Utilizar una variable global para acumular el resultado total. El último hilo tiene que mostrar por pantalla el resultado total.

Implementar el código a ejecutar por los hilos 1 a N-1 (*ThreadRest*) y el código del último hilo (*ThreadN*).

```
/* Declare and initialize semaphores and global variables */
```

```
ThreadRest(int id)
{
   char msg;



   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));




}
```

```
ThreadN(int id)
{
   char msg;

   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));

   /* Print total sum */
   sprintf(msg," Total sum: %d.\n", Total);
   write(1,msg,strlen(msg));
}
```

d) Asumiendo que se crean N hilos y cada uno de ellos tiene que ejecutar una tarea que puede implicar un tiempo considerable. Se pide que garanticéis la sincronización entre los hilos, de forma que los hilos pares no puedan ejecutar su tareas de forma simultánea con los hilos impares. Pueden haber múltiples hilos pares ejecutando su tarea al mismo tiempo sin generar problemas. Por último, dos hilos impares no pueden ejecutar su tarea al mismo tiempo.

   Implementar el código a ejecutar por los hilos pares (*ThreadPair*) y los hilos impares (*ThreadOdd*).

```
/* Declare and initialize semaphores and global variables */
```

```
ThreadPair(int id)
{
   char msg;




   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));
   Task(id);




}
```

```
ThreadOdd(int id)
{
   char msg;



   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));
   Task(id);




}
```

Recursos

Básicos:

- Módulos 5, 6, y 7 de la asignatura.
- Documento "Introducción a la programación de UNIX" (disponible en el aula) o cualquier otro manual similar.
- El aula "Laboratorio de Sistemas Operativos" (puede plantear sus dudas relativos al entorno Unix, programación, ...).
- Complementarios:
- La bibliografía de la asignatura.

Criterios de valoración

Se valorará la justificación de las respuestas presentadas. Se agradecerán las respuestas breves y concisas. El peso de cada pregunta está indicado en el enunciado.

En la corrección se tendrán en cuenta los siguientes aspectos:

- Las respuestas deberán estar articuladas a partir de los conceptos estudiados en teoría y en la guías de la asignatura.
- Se agradecerá la claridad y la capacidad de síntesis en las respuestas.
- Se valorará esencialmente la correcta justificación de las respuestas, apoyada por los fundamentos teóricos.

Formato y fecha de entrega

La solución se entregará en un fichero texto (formato. Txt o. Pdf)

El nombre del archivo tendrá el siguiente formato: "Apellido1Apellido2PEC2.txt".Los apellidos se escribirán sin acentos. Por ejemplo, el estudiante Marta Vallès y Marfany utilizará el nombre de **archivo siguiente: VallesMarfanyPEC2.txt**

La fecha límite para la entrega de la PEC es el **jueves 27 de diciembre de 2018.**