DISEÑO Y ADMINISTRACIÓN DE SISTEMAS OPERATIVOS **Febrero 2013**

Material permitido: **NINGUNO.** Tiempo: **2 horas.** N1	**Aviso 1:** Todas las respuestas deben estar razonadas. **Aviso 2:** Escriba sus respuestas con una letra **lo más clara posible.** **Aviso 3: No use Tipp-ex** o similares (atasca el escáner).

ESTE EXÁMEN CONSTA DE 5 PREGUNTAS

1. Explique **razonadamente** si las siguientes afirmaciones son verdaderas o falsas:

i) *(1p)* Un *proceso ejecutándose en modo usuario* puede acceder a otras partes de su propio espacio de direcciones, como aquellas reservadas para estructuras de datos asociadas al proceso usadas por el núcleo.

ii) *(1p)* El núcleo de UNIX realiza la invocación del algoritmo `wakeup()` únicamente dentro de los algoritmos asociados a las llamadas al sistema.

Solución:

i) FALSA (páginas 12-13)

Con el objetivo de poder implementar una protección eficiente del espacio de direcciones de memoria asociado al núcleo y de los espacios de direcciones asociados a cada proceso, la ejecución de los procesos en un sistema UNIX está dividida en dos modos de ejecución: un modo de mayor privilegio denominado *modo núcleo o supervisor* y otro modo de menor privilegio denominado *modo usuario*.
Un *proceso ejecutándose en modo usuario* sólo puede acceder a unas partes de su propio espacio de direcciones (código, datos y pila). Sin embargo, no puede acceder a otras partes de su propio espacio de direcciones, como aquellas reservadas para estructuras de datos asociadas al proceso usadas por el núcleo.

Tampoco puede acceder al espacio de direcciones de otros procesos o del mismo núcleo. De esta forma se evita una posible corrupción de los mismos.

Por otra parte, un *proceso ejecutándose en modo núcleo* puede acceder a su propio espacio de direcciones al completo y al espacio de direcciones del núcleo, pero no puede acceder al espacio de direcciones de otros procesos. Debe quedar claro que cuando se dice que un proceso se está ejecutando en modo núcleo, en realidad el que se está ejecutando es el núcleo pero en el nombre del proceso. Por ejemplo, cuando un proceso en modo usuario realiza una llamada al sistema está pidiendo al núcleo que realice en su nombre determinadas operaciones con el hardware de la máquina.

ii) FALSA (página 193)

El núcleo usa el algoritmo `wakeup()` para despertar a un proceso que se encuentra en el estado *dormido* a la espera de la aparición de un determinado evento. Típicamente la invocación de `wakeup()` se realiza dentro de algún otro algoritmo del núcleo como los asociados a las llamadas al sistema **o las rutinas de manipulación de interrupciones.**

2.*(1.5 p)* Describe las principales limitaciones que presentan las tuberías.

Solución:

Como mecanismo IPC, las tuberías proporcionan una forma eficiente de transferir datos de un proceso a otro. Sin embargo poseen algunas limitaciones importantes:

- Una tubería no puede ser utilizada para transmitir datos a múltiples procesos receptores de forma simultánea, ya que al leer los datos de la tubería estos son borrados.
- Si existen varios procesos que desean leer en un extremo de la tubería, un proceso que escriba en el otro extremo no puede dirigir los datos a un proceso en concreto. Asimismo, si existen varios procesos que desean escribir en la tubería, no existe forma de determinar cuál de ellos envía los datos.
- Si un proceso envía varios mensajes de diferente longitud en una sola operación de escritura en la tubería, el proceso que lee el otro extremo de la tubería no puede determinar cuántos mensajes han sido enviados, o dónde termina un mensaje y donde empieza el otro, ya que los datos en la tubería son tratados como un flujo de bytes no estructurados de tamaño fijo.

3 *(2 p)* ¿qué diferencia existe entre una copia de seguridad y un *snapshot* en un sistema que implementa *copy-on-write*?

Solución:
Una instantánea o *snapshot* es una imagen del sistema de archivos en un instante dado, en ese aspecto es parecida a una copia de seguridad en tanto en cuanto es posible restaurar los archivos a su estado anterior en caso de necesidad.

Cuando se usa la técnica *copy-on-write* la creación de una instantánea es muy rápida y ocupa espacio a medida que los archivos van cambiando, mientras que la copia de seguridad lleva bastante tiempo y suele requerir más espacio. Lo mismo ocurre con la recuperación ya que para volver a un estado anterior del sistema de ficheros basta con montar una instantánea mientras que con la copia de seguridad hay que copiar todos los archivos.

No obstante las instantáneas en un sistema con *copy-on-write* residen en el mismo disco físico que los datos a copiar de modo que si se daña el disco los datos se pierden. Esto no ocurre en las copias de seguridad que normalmente se realizan en un medio de almacenamiento independiente.

4. *(2 p)* Explique razonadamente el significado de la siguiente llamada al sistema:

```
y=times(&x);
```

Solución:
La llamada al sistema `times` permite conocer el tiempo empleado por un proceso en su ejecución.
Si times se ejecuta correctamente entonces rellena la estructura `x` que debe ser del tipo predefinido `tms` con la información estadística relativa a los tiempos de ejecución empleados por el proceso, desde su inicio hasta el momento de invocar a times.

Además en `y` se almacenará el tiempo real transcurrido (en tics) a partir de un instante pasado arbitrario.
Este instante puede ser el momento de arranque del sistema y no cambia de una llamada a otra.
Si `times` falla entonces y contendrá el valor -1.
El tipo `tms` se define de la siguiente forma:

```
struct tms
{clock_t tms_utime
clock_t tms_stime
clock_t tms_cutime
clock_t tms_cstime}
```

El tipo `clock_t` se define para contabilizar los tics de reloj. Por lo tanto el significado de los campos de la estructura x es el siguiente:

- `tms_utime`. Es el tiempo de uso de la CPU (en tics) del proceso ejecutándose en modo usuario.
- `tms_stime`. Es el tiempo de uso de la CPU (en tics) del proceso ejecutándose en modo núcleo.
- `tms_cutime`. Es la suma de los campos `tms_utime` y `tms_cutime` para los procesos hijos. Es decir, el tiempo de uso de la CPU (en tics) de los procesos hijos, los hijos de los hijos, etc. ejecutándose en modo usuario.
- `tms_cstime`. Es la suma de los campos `tms_stime` y `tms_cstime` para los procesos hijos. Es decir, el tiempo de uso de la CPU (en tics) de los procesos hijos y sus descendientes ejecutándose en modo núcleo.

5. Conteste razonadamente a los siguientes apartados:
a) (*1 p*) Explicar el significado de las sentencias enumeradas (**[ ]**) de este programa.
b) (*1.5 p*) Explicar el funcionamiento del programa mostrando la salida en pantalla.

```
     #include <signal.h>
     #include <stdio.h>
     #include <math.h>

     long t1;

     void fun1(int sig);
     void fun2(int sig);

     void main(void)
     {
[1]       signal(SIGVTALRM, fun1);
          signal(SIGALRM, fun2);
          printf("Comienza:\n");
          time(&t1);
[2]       alarm(3);
[3]       pause();
[4]       kill(getpid(),SIGVTALRM);
          printf("Acaba.\n");
     }

     void fun1(int sig)
     {
          long t2;
[5]       time(&t2);
[6]       printf("fun1:Han pasado %d segundos\n",(int)(t2-t1));
          kill(getpid(),SIGTERM);
     }
     void fun2(int sig)
     {
          long t2;
          time(&t2);
          printf("fun2:Han pasado %d segundos\n",(int)(t2-t1));
          kill(getpid(),SIGTERM);
     }
```

Solución :

a) El significado de las sentencias numeradas es el siguiente:

[1]: La llamada al sistema asigna la función `fun1` como el manejador de la señales de tipo `SIGALARM`.

[2]: La llamada al sistema `alarm` solicita una alarma de tiempo real al cabo de 3 segundos.

[3]: La llamada al sistema pause bloquea la ejecución del proceso hasta que reciba cualquier señal que no ignore o tenga bloqueada.

[4]: La llamada al sistema `getpid()` devuelve el `pid` del proceso actual, a continuación la llamada al sistema `kill` envía la señal `SIGVTALRM` dicho proceso.

[5]: La llamada al sistema `time` almacena en la variable local `t2` el valor en segundos del tiempo transcurrido desde las 00:00:00 GMT del día 1 de enero de 1970.

[6]: En esta sentencia se calcula la diferencia de tiempo en segundos desde que se invocó la orden `time(&t1)` y la muestra por la salida estándar.

b) El programa funciona de la siguiente manera. En primer lugar se declara una variable global t1, a continuación se comienza a ejecutar la función `main`. Dicha función establece fun1 como manejador de la señal `SIGVTALRM` y `fun2` como manejador de la señal `SIGALRM`. A continuación se imprime por la salida estándar "`Comienza:`" y se salta a la línea siguiente.

Hecho esto se guarda el valor de la fecha actual en la variable `t1` y se solicita una alarma de tiempo real dentro de 3 segundos. Seguidamente el proceso se bloquea a la espera de una alarma [3].

Tres segundos después se dispara la alarma de tiempo real y el sistema operativo envía una señal `SIGALRM` al proceso. Dicho proceso no ignora ni bloquea la señal `SIGALRM` por lo tanto se desbloquea y ejecuta el manejador `fun2`.

Dicho manejador almacena la fecha actual en la variable local `t2` y después calcula el tiempo que ha transcurrido y lo muestra por pantalla. A continuación se ejecuta `kill(getpid(),SIGTERM)` que envía una señal `SIGTERM` al proceso actual. Dicha señal no es capturada ni ignorada por lo que termina el proceso.

El Sistema Operativo muestra un mensaje por la salida estándar indicando que el proceso ha finalizado. Dependiendo de la distribución este mensaje puede variar, supongamos que se imprime el mensaje "`Terminado`", entonces la salida del programa es la siguiente:

```
Comienza:
fun2:Han pasado 3 segundos
Terminado
```

Es importante darse cuenta de que la orden [4] nunca llega a ejecutarse. Antes de que el proceso pueda continuar (por la línea [4]) deben atenderse todas las señales (incluida `SIGTERM`). De este modo el proceso termina sin poder ejecutar `fun1` e imprimir "`Acaba.`"