AMPLIACIÓN DE SISTEMAS OPERATIVOS (71023016) Febrero 2024

Material permitido: **Ninguno** Tiempo: **120 minutos** **N2**	**Aviso 1**: Todas las respuestas deben estar debidamente razonadas. **Aviso 2**: Escriba con buena letra y evite los tachones. **Aviso 3**: Solución del examen y fecha de revisión en http://www.uned.es/71023016/

1. Conteste **razonadamente** a los siguientes apartados:

a) *(1 p)* ¿Qué operaciones se pueden realizar sobre el espacio de direcciones virtuales de un proceso en los SOBUNIX?

b) *(1 p)* En los SOBUNIX, ¿qué es un nodo-im? ¿qué información contiene?

c) *(1 p)* Explicar la utilidad y el funcionamiento del algoritmo PFRA de Linux.

d) *(1 p)* Describir los dos *tipos de drivers* que se distinguen en MS-DOS.

2. *(2 p)* Explicar **razonadamente** la relación existente entre procesos, procesos ligeros e hilos del núcleo en los SOBUNIX.

3. En el contenido del archivo `etc/passwd` de un cierto SOBUNIX aparece, entre otras, la siguiente línea:

```
sara03:x:220:30:Sara Lopez:/home/usuario1:/bin/bash
```

a) *(1.5 p)* Explicar el significado de cada uno de los elementos presentes en esta línea.

b) *(0.5 p)* ¿Cómo se debería modificar el contenido de la línea anterior para impedir al usuario el acceso al sistema sin eliminar su cuenta?

Material permitido: **Ninguno** Tiempo: **120 minutos** **XP**	**Aviso 1**: Todas las respuestas deben estar debidamente razonadas. **Aviso 2**: Escriba con buena letra y evite los tachones. **Aviso 3**: Fecha de revisión en http://www.uned.es/71023016/

4. *(2 p)* En la Figura 1 se muestra el código C del programa `f24`.

```
main(int argc, char *argv[])
{
    int a,b=0,c,d;
    if (argc!=2) exit(3);
    else
    {
        a=atoi(argv[1]);
        while (fork()==0 && b!=a)
        {
            printf("\nMensaje ZZ[%d]\n", getpid());
            b=b+1;
        }
    }
    c=wait(&d);
    printf("\nMensaje XX[%d]=%d\n", getpid(),getppid());
}
```

Figura 1 – Código C del programa `f24`

Supóngase que al invocar este programa desde la línea de ordenes de un intérprete de comandos se crea un proceso con PID=3510 y que la asignación de los PIDs de los procesos hijos, si se llegaran a crear, se realizaría incrementando en una unidad el PID del proceso padre. Supóngase además que el intérprete de comandos desde donde se lanza `f24` tiene asignado PID= 425. Conteste **razonadamente** a los siguientes apartados:

a) *(1 p)* Explique el significado de las siguientes llamadas al sistema que aparecen en el código del programa:

   i) `while (fork()==0 && b!=a);`

   ii) `c=wait(&d);`

b) *(1 p)* Explicar el funcionamiento del programa si se invoca desde el intérprete de comandos mediante la orden: `f24 3`

AMPLIACIÓN DE SISTEMAS OPERATIVOS (Cód. 71023016)
Solución Examen Febrero 2024

Solución Ejercicio 1

a) Sobre el espacio de direcciones virtuales de un proceso el núcleo puede realizar diferentes operaciones. Estas operaciones se pueden clasificar en dos grandes grupos:

   – *Operaciones que afectan a todo el espacio de direcciones de un proceso*. Como la creación de un nuevo espacio de direcciones, y la duplicación o eliminación de un espacio de direcciones ya existente.
   – *Operaciones que afectan a una región o segmento del espacio de direcciones de un proceso*. Como la creación de una nueva región, la eliminación de una región, la configuración y la comprobación de los permisos de acceso a una región, etc.

   Cada SOBUNIX define qué operaciones sobre el espacio de direcciones soporta y cómo las implementa.

b) En los SOBUNIX, el núcleo para poder trabajar con un archivo, aparte de asignarle un nodo-v, necesita tener cargado en la memoria principal el nodo-i asociado al archivo, ya que contiene los atributos y la localización de sus bloques de datos. El núcleo copia el contenido de un nodo-i en una estructura de datos denominada *nodo-im*[1] que crea en su espacio de direcciones.

   En un nodo-im, aparte de una copia de un nodo-i del disco duro, también se almacenan, entre otros, los siguientes datos: un puntero al nodo-v asociado al nodo-im, punteros para colocar al nodo-im en una de las *colas hash* que mantiene el núcleo en función del número de nodo-i para localizar a los nodos-im, y punteros para colocar al nodo-im en una *lista de nodos-im libres*.

c) Linux utiliza el algoritmo PFRA (*Page Frame Reclaiming Algorithm*, algoritmo para reclamar marcos de página) para seleccionar los marcos de página de la memoria física cuyo contenido puede ser reemplazado.

   El algoritmo PFRA es una variante del algoritmo del reloj y una aproximación del algoritmo LRU. Este algoritmo trabaja con la *lista de marcos de página activos* y con la *lista de marcos de página inactivos* de cada zona. La *lista de marcos de página activos* contiene los descriptores de página de los marcos que almacenan páginas que han sido referenciadas recientemente. Por su parte, la *lista de marcos de página inactivos* contiene los descriptores de página de los marcos que almacenan páginas que llevan algún tiempo sin ser referenciadas. Dentro de cada lista los descriptores de página de los marcos son ordenados mediante una aproximación de la política LRU. Por ello a estas listas también se las denomina de forma global como *listas* LRU.

   Para decidir en que lista debe colocar un marco de página el algoritmo PFRA utiliza los indicadores `PG_referenced` y `PG_active` existentes en el campo `flags` de la estructura `page` asociado a un descriptor de página. Por simplificar la notación se va a denotar a los indicadores `PG_referenced` y `PG_active` con las letras r y a, respectivamente.

   Cada vez que una página i cargada en un marco j es referenciada, el núcleo activa el indicador r del marco, es decir, $r = 1$. Por otra parte, si un marco pertenece a la lista de marcos de página activos entonces $a = 1$, mientras que si pertenece a la lista de marcos de página inactivos entonces $a = 0$.

   Cuando se ejecuta el algoritmo PFRA éste examina el indicador r de los descriptores de página de los marcos de una determina zona. Si r estaba activado ($r = 1$) entonces lo desactiva ($r = 0$).

[1] El nombre que recibe esta estructura depende de cada SOBUNIX, el cual lo asigna en función del tipo de sistema de archivos.

Cuando termina de examinarlos, comienza a realizar una segunda ronda. Nótese que durante el tiempo entre la primera y la segunda comprobación de un mismo marco éste ha podido volver a ser referenciado y su indicador *r* estar activado. Si durante la segunda ronda el algoritmo encuentra un marco *j* con $r = 1$ y cuyo bit *r* hubiera sido desactivado en la primera ronda entonces el algoritmo vuelve a desactivarlo, activa *a* y coloca al marco en la lista de marcos de página activos. Luego para que un marco *j* ingrese en la lista de marcos de página activos debe encontrarse con el bit *r* activado en dos rondas del algoritmo PFRA.

d) En MS-DOS se distinguen dos tipos de drivers de dispositivos de E/S: *drivers residentes* y *drivers instalables*. Los *drivers residentes* se organizan en una capa de software denominada BIOS cuya parte más primitiva es almacenada por el fabricante del computador en una memoria ROM. Durante el proceso de arranque del sistema la BIOS es cargada en la memoria RAM junto con el archivo `io.sys`, el cual contiene rutinas de E/S que extienden la funcionalidad de la BIOS.

Los drivers residentes permiten al sistema operativo comunicarse con el hardware básico de la computadora (disco de arranque, pantalla, teclado, puerto de impresión, reloj, etc). Para poder utilizar otros dispositivos de E/S distintos a los soportados en la BIOS, sus drivers deben ser instalados durante el arranque del sistema. A estos drivers se les denomina *drivers instalables*, y son cargados en la memoria principal en un área distinta de la BIOS. Señalar que un driver instalable también puede sustituir en la memoria principal a un driver residente. De esta forma es posible utilizar versiones más actualizadas de un determinado driver residente.

Solución Ejercicio 2

Un *programa* es un archivo ejecutable que típicamente se encuentra almacenado en memoria secundaria. Considerado como archivo, un programa es una entidad pasiva o estática. Cuando un programa es ejecutado se convierte en una entidad activa o dinámica denominada *proceso* que va pasando por diferentes estados y utiliza distintos recursos del computador. En definitiva, un proceso es un programa en ejecución.

Un *proceso* se puede considerar como una entidad computacional básica que tiene asignado un conjunto de recursos y que durante su existencia ejecuta una serie de instrucciones. Así un proceso queda caracterizado por dos elementos que se pueden tratar de forma independiente:

- *Conjunto de recursos asignados*. Un proceso tiene asociado un espacio de direcciones de memoria virtual (código, datos, pila,...) y otros recursos, tales como archivos abiertos, procesos hijos, manejadores de señales, información de contabilidad, etc.
- *Hilos de control* o simplemente *hilos* (thread). Un hilo es un subconjunto de instrucciones del código del proceso que se puede ejecutar independientemente del resto de instrucciones del proceso. En un computador con N procesadores es posible ejecutar en paralelo N hilos pertenecientes a uno o varios procesos.

En general se distinguen los siguientes tipos de hilos en función de sus propiedades y usos:

- *Hilos del núcleo* (kernel threads). Son hilos asociados al código del núcleo del sistema operativo que son creados y gestionados por el propio núcleo para la realización de funciones especificas, como por ejemplo: dar soporte a procesos, atender operaciones de E/S, tratamiento de interrupciones, etc. Los hilos del núcleo no necesitan estar asociados a ningún proceso.
- *Hilos de usuario* (user threads). Son parte del código de un determinado proceso de usuario. El programador al diseñar un programa es el encargado de establecer el número de hilos de usuario en que se va descomponer el código del programa. Si el código de un programa consta de un único hilo de usuario entonces al proceso asociado se le denomina *proceso monohilo*. Mientras que si se descompone en varios hilos de usuario se denomina *proceso multihilo*. Para la gestión de los hilos de usuario el programador utiliza una *librería de hilos*, como por ejemplo la librería *Pthreads* de POSIX. Una librería de hilos contiene funciones para la creación, destrucción, planificación y sincronización de los hilos de usuario, entre otras operaciones. El sistema operativo generalmente no suele conocer la existencia de los hilos de usuario, aunque debe dar servicio a las llamadas al sistema que pueda realizar la librería de hilos para implementar la gestión de los hilos. Además internamente la librería de hilos puede crear hilos cuya existencia es desconocida por los hilos de usuario que gestiona.
- *Procesos ligeros* (lightweight processes). También denominados *procesadores virtuales*. Son hilos de usuario cuya gestión es realizada por el núcleo[2] del sistema operativo. Un proceso ligero puede estar asociado a uno o varios hilos de usuario. Es la librería de hilos la que se encarga de multiplexar los hilos de usuario de un determinado proceso en un número menor o igual de procesos ligeros. Los procesos ligeros solo se pueden utilizar en aquellos sistemas operativos cuyo núcleo soporte hilos del núcleo, ya que cada proceso ligero tiene que tener asociado un hilo del núcleo para poder ser planificado y ejecutado.

[2]En algunos sistemas operativos el termino *proceso ligero* también se utiliza como sinónimo del término *hilo de usuario* independientemente de si el núcleo conoce la existencia y gestiona dicho hilo.

Solución Ejercicio 3

a) De izquierda a derecha se muestran los siguientes datos: `sara03` es el nombre de usuario, `x` indica que la contraseña se encuentra encriptada dentro del archivo cuyo nombre de ruta absoluta suele ser `/etc/shadow`, `220` es el UID, `30` es el GID, `Sara Lopez` es el nombre completo del usuario, `/home/usuario1` es el nombre de ruta absoluta del directorio de trabajo inicial, y `/bin/bash` es el nombre de ruta absoluta del intérprete de comandos que utiliza por defecto el usuario.

b) Si se desea impedir el acceso a la usuaria `sara03` su línea del archivo debería modificarse de la siguiente forma:

```
sara03:*x:220:30:Sara Lopez:/home/usuario1:/bin/bash
```

Solución Ejercicio 4

a) i) Bucle `while` cuya condición para que se ejecute una iteración es que la llamada al sistema `fork` devuelva el valor 0 y que la variable `b` no tome el valor `a`. Recuérdese que `fork` crea un proceso hijo y devuelve 0 para el proceso hijo y el PID del hijo al proceso padre.

ii) Esta llamada al sistema pone al proceso que la invoca en en estado dormido a la espera de que finalice alguno de sus procesos hijos. Si se ejecuta con éxito devuelve en `c` el PID del primer hijo en estado zombi que encuentra, en caso de error devuelve -1. Además devuelve en `&d` el código de salida del proceso hijo finalizado.

b) Al escribir la orden `f24 3` se comienza a ejecutar el programa `f24`. Supóngase que a la ejecución de dicho programa se le asocia el proceso A, cuyo PID=3510 según el enunciado. En primer lugar se comprueba si el número de argumentos con que ha sido invocado `f24` es distinto de dos. Conviene recordar que el nombre del programa cuenta como argumento. Puesto que `f24 4` tiene dos argumentos la condición del `if` no se cumple y se pasan a ejecutar las sentencias del `else`.

En segundo lugar se invoca a la función de librería `atoi` que convierte el segundo argumento ('`4`') de la invocación de `f24` a número entero y lo almacena en la variable `a`.

En tercer lugar se ejecuta un bucle `while` cuya condición para que se ejecute una iteración es que la llamada al sistema `fork` devuelva el valor 0 y que la variable `b` no tome el valor `a`. Recuérdese que `fork` crea un proceso hijo y devuelve 0 para el proceso hijo y el PID del hijo al proceso padre.

La evaluación de esta condición da como resultado la creación de un hijo B con PID=3511, que al cumplir que `fork` le ha devuelto 0 y que `b` es distinta de 3 ejecuta el contenido del bucle. El proceso padre A no ejecuta el bucle ya que `fork` no le ha devuelto un 0 sino 3511, por lo que ejecuta una llamada al sistema `wait` y se queda a la espera de que su proceso hijo B termine.

El proceso B muestra en pantalla el mensaje

```
Mensaje 1[3511]
```

e incrementa en una unidad el valor de la variable `b` que ahora toma el valor 1. A continuación evalúa la condición del bucle que da como resultado la creación de un hijo C con PID=3512, que al cumplir que `fork` le ha devuelto 0 y que `b` es distinto de 3 ejecuta el contenido del bucle. El proceso padre B no ejecuta el bucle ya que `fork` no le ha devuelto un 0 sino 3512, por lo que ejecuta una llamada al sistema `wait` y se queda a la espera de que su proceso hijo C termine.

El proceso C muestra en pantalla el mensaje

```
Mensaje 1[3512]
```

e incrementa en una unidad el valor de la variable `b` que ahora toma el valor 2. A continuación evalúa la condición del bucle que da como resultado la creación de un hijo D con PID=3513, que al cumplir que `fork` le ha devuelto 0 y que `b` es distinto de 3 ejecuta el contenido del bucle. El proceso padre C no ejecuta el bucle ya que `fork` no le ha devuelto un 0 sino 3513, por lo que ejecuta una llamada al sistema `wait` y se queda a la espera de que su proceso hijo D termine.

El proceso D muestra en pantalla el mensaje

```
Mensaje 1[3513]
```

e incrementa en una unidad el valor de la variable `b` que ahora toma el valor 3. A continuación evalúa la condición del bucle que da como resultado la creación de un hijo E con PID=3514, sin embargo como `b` es igual 3 no ejecuta el contenido del bucle sino que ejecuta la llamada al sistema `wait`, pero como E no tiene hijos no se suspende sino que imprime en pantalla el mensaje

```
Mensaje 2[3514]=3513
```

y finaliza.

Por su parte el proceso padre D tampoco ejecuta el bucle porque para él `fork` no ha devuelto un 0 sino 3514, y además `b=3` por lo que ejecuta una llamada al sistema `wait` y se queda a la espera de que su proceso hijo E termine. Como éste ya ha finalizado imprime en pantalla el mensaje

```
Mensaje 2[3513]=3512
```

y finaliza.

Como su hijo D ya ha terminado se despierta al proceso C que imprime en pantalla el mensaje

```
Mensaje 2[3512]=3511
```

y finaliza.

A continuación puesto que su hijo C ya ha terminado se despierta al proceso B que imprime en pantalla el mensaje

```
Mensaje 2[3511]=3510
```

y finaliza.

Finalmente como su hijo B ya ha terminado se despierta al proceso A que imprime en pantalla el mensaje

```
Mensaje 2[3510]=425
```

y finaliza.