AMPLIACIÓN DE SISTEMAS OPERATIVOS (71023016) **Febrero 2023**

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 las siguientes apartados:

a) *(1 p)* En los SOBUNIX, ¿Qué especificaciones tiene asociada cada *clase de planificación*? ¿Qué dos *tipos de funciones* utiliza el planificador para manipular las clases de planificación?

b) *(1 p)* En los SOBUNIX, ¿qué es un *objeto de archivo abierto*? ¿Cuándo se crea? Enumerar y explicar la información que contiene.

c) *(1 p)* Enumerar los *asignadores de memoria* implementados en Linux.

d) *(1 p)* Enumerar las características principales de un *sistema de archivos* NTFS.

2. *(2 p)* Explicar **razonadamente** qué información es necesario mantener por cada *hilo del núcleo* en los SOBUNIX. ¿Quién gestiona esta información? ¿Dónde se almacena?

3. *(2 p)* Explique **razonadamente** cuál sería el resultado de la ejecución de las siguientes órdenes:

a) `pwd`

b) `ls -l`

c) `sort < fA > fB`

d) `cat /etc/passwd`

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/

4. A continuación se muestra el código C del programa `f23`:

```
main()
{
    int y, x=0;
    y=fork();
    if (y==-1){
        printf("\n Mensaje 1");
    }
    else if (y==0){
        printf("\n A=%d, B=%d",getpid(),getppid());
        x=x+1;
        printf("\nx=%d",x);
    }
    else{
        printf("\C=%d",getpid());
        x=x-1;
        printf("\nx=%d",x);
    }
    printf("\nMensaje 2");
}
```

Figura 1 – Código C del programa `f23`

Supóngase que al invocar este programa desde la línea de ordenes de un intérprete de comandos se crea un proceso con PID=2012 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. Suponer además que el intérprete de comandos desde donde se lanza `f23` tiene asignado un PID= 534. Conteste **razonadamente** a los siguientes apartados:

a) *(0.5 p)* Explique el significado de las siguientes llamadas al sistema que aparecen en el código del programa `f23`:

   i) `fork();`

   ii) `getppid();`

b) *(1.5 p)* Explicar el funcionamiento del programa `f23` en función del orden de planificación que establezca el planificador del sistema operativo para los diferentes procesos que pueda generar la ejecución de este programa.

AMPLIACIÓN DE SISTEMAS OPERATIVOS (Cód. 71023016)
Solución Examen Febrero 2023

Solución Ejercicio 1

a) Algunos SOBUNIX, para dotar de mayor flexibilidad al planificador definen *clases de planificación* cada una de los cuales tiene asociada un rango de valores de prioridad y un mecanismo de cálculo de la prioridad. Cada unidad planificable pertenece a una determinada clase de planificación. Normalmente se dispone de una clase de planificación por cada tipo de trabajo soportado en el sistema: clase de trabajos en tiempo real, clase de trabajos de tiempo compartido o interactivos, etc.

   Las clases de planificación son manipuladas por el planificador usando dos posibles tipos de funciones:

   - Funciones independientes de la clase. El planificador utiliza éstas funciones para la realización de servicios que son comunes a todas las clases, como por ejemplo: el cambio de contexto y la manipulación de las colas de planificación.
   - Funciones dependientes de la clase. El planificador utiliza éstas funciones para la realización de servicios que dependen de la clase, como por ejemplo el cálculo de la prioridad. Cada clase suministra su propia implementación de estas funciones.

b) Para que un proceso pueda poder operar sobre un archivo, éste primero tiene que ser abierto mediante la invocación de la llamada al sistema `open`. La rutina del núcleo del sistema operativo que trata esta llamada al sistema crea en la memoria principal una estructura de datos denominada *objeto de archivo abierto* y asigna a dicha estructura un número entero positivo pequeño denominado *descriptor de archivo* para identificarla.

   Un objeto de archivo abierto contiene la información necesaria para poder operar sobre un archivo:

   - Puntero de lectura/escritura. Indica el byte del archivo donde se realizará la próxima operación de lectura o de escritura. Se especifica como un desplazamiento desde el origen del archivo.
   - Contador de referencias. Especifica el número de descriptores de archivos que hacen referencia al objeto de archivo abierto.
   - Modo de apertura del archivo. Establece las operaciones que se pueden realizar sobre el archivo abierto. Por ejemplo, solo lectura, solo escritura, lectura y escritura, etc.
   - Puntero al nodo virtual del archivo. Un *nodo virtual*, o más abreviadamente *nodo-v*, es una estructura de datos que el núcleo crea en memoria principal para cada archivo activo.

c) El subsistema de gestión de memoria de Linux para atender las necesidades de memoria de los procesos de usuario y de los restantes subsistemas del núcleo implementa los siguientes asignadores de memoria:

   - Asignador a nivel de página. Asigna uno o varios marcos de página contiguos utilizando el *algoritmo buddy*
   - Asignador de la memoria del núcleo. Asigna fragmentos de memoria principal de tamaño menor a una página. Para realizar esta función implementa un *asignador slab*.
   - *Asignador* `vmalloc`. Se utiliza para asignar grandes cantidades de espacio virtual del núcleo contiguo que no requiere que esté contiguo en memoria principal, como por ejemplo cuando se cargan dinámicamente módulos del núcleo.

d) Un sistema de archivos NTFS presenta, entre otras, las siguientes características:

- La unidad básica de asignación de espacio es el cluster.
- Toda la información contenida en un sistema de archivos NTFS *se estructura en archivos*.
- Un archivo (o un directorio) consiste de múltiples flujos de bytes.
- Un archivo puede constar de múltiples flujos de datos.
- Los nombres de los archivos se codifican en código Unicode.
- Un nombre de archivo puede tener una extensión máxima de 255 caracteres.
- Implementa una estructura de directorios de gráfica acíclica con soporte de enlaces duros y de enlaces simbólicos.
- Recuperación. Un sistema de archivos NTFS puede ser recuperado y devuelto a un estado consistente si se producen fallos en el sistema o en el disco.
- Registro por diario (journaling).
- Soporte de cuotas. NTFS dispone de mecanismos para fiscalizar y limitar el espacio en disco disponible para cada usuario.
- Compresión.
- Encriptación.

Solución Ejercicio 2

En los SOBUNIX que soportan hilos del núcleo el sistema operativo debe mantener en su espacio de direcciones una estructura de datos y una pila del núcleo por cada hilo del núcleo. La estructura de datos asociada a un hilo del núcleo contiene la siguiente información:

- *Identificador del hilo núcleo*. Es un número entero positivo que identifica de manera univoca a cada hilo del núcleo.
- *Contexto hardware salvado*. Son los valores que contenían los registros del computador cuando se interrumpió la ejecución del hilo del núcleo debido a un cambio de contexto. El contexto hardware será restaurado cuando el hilo del núcleo vuelva a ser planificado y se continúe con la ejecución del hilo.
- *Parámetros de planificación*. Es la información que utiliza el planificador del núcleo para decidir qué hilo del núcleo planificar, como por ejemplo, la prioridad de planificación del hilo o el tiempo de uso del procesador.
- *Punteros para implementar diferentes colas de hilos del núcleo*. Como colas de hilos preparados para ejecución, colas de hilos dormidos, etc.
- *Puntero a la pila del núcleo asociada al hilo*.
- *Información sobre el proceso ligero asociado*. Como por ejemplo un puntero a la estructura `lwp` del proceso ligero al que está asociado y un puntero a la estructura `proc` del proceso del que forma parte el proceso ligero. Si el hilo del núcleo no está asociado a ningún proceso ligero, entonces estos punteros contienen valores `NULL`.

Por su parte, una *pila del núcleo* es una estructura que contiene los marcos de pila de las funciones invocadas por el hilo durante su ejecución en modo núcleo. Está pila está vacía cuando el hilo del núcleo se ejecuta en modo usuario.

Solución Ejercicio 3

a) Esta orden muestra el nombre de ruta absoluta del directorio de trabajo actual.

b) Esta orden muestra un listado largo un listado largo de los archivos del directorio de trabajo actual incluyendo tamaño, permisos, propietario, permisos, etc.

c) Esta orden ordena (`sort`) por orden alfabético las líneas del archivo `fA` y escribe el resultado en el archivo `fB`.

d) Esta orden muestra el contenido del archivo `/etc/passwd`

Solución Ejercicio 4

a) i) La llamada al sistema `fork()` crea un nuevo proceso. Al proceso que invoca a `fork` se le denomina *proceso padre* y al nuevo proceso que se crea se le denomina *proceso hijo*. Si se ejecuta con éxito esta llamada al sistema devuelve en *c* el PID del proceso hijo para el proceso padre y 0 para el proceso hijo. En caso de error devuelve -1.

ii) La llamada al sistema `getppid()` devuelve para el proceso invocador el PID de su proceso padre.

b) Supóngase que para ejecutar el programa `prog21` el intérprete crea el proceso A con un PID=2012. Cuando se ejecuta este proceso en primer lugar se invoca a la llamada al sistema `fork`. Si la llamada se ejecuta con éxito se crea un proceso hijo B con PID=2013. Si se produjese algún error durante el tratamiento por parte del núcleo de esta llamada al sistema, `fork` devolvería el valor `-1`. Luego la condición del `if` se cumpliría y se imprimiría en la salida estándar (la pantalla por defecto) el mensaje `Mensaje1`. A continuación se imprimiría el mensaje `Mensaje2` y la ejecución del proceso A finalizaría.

En función de que proceso se planifique primero para ejecución, el padre A o el hijo B, el funcionamiento del programa sería el siguiente:

i) El planificador planifica primero al proceso hijo B.

En ese caso la variable `y=0` luego el proceso B ejecuta el conjunto de instrucciones del bloque `else if`. Señalar que las funciones `getpid` y `getppid` son llamadas al sistema que permiten obtener el PID de un proceso y el PID de su proceso padre, respectivamente. Así en primer lugar se imprime en pantalla el mensaje:

```
Soy el hijo PID=2013, mi padre tiene PID=2012
```

A continuación el proceso B suma 1 a la variable `x` y almacena su valor en la misma posición de memoria. Finalmente se muestra en la pantalla el valor de `x` (en este caso 1) y el mensaje `Mensaje2`, y el proceso B finaliza.

Supóngase que finalizado el proceso hijo B se planifica para ejecución el proceso A. En este caso la variable `y` contiene el valor 2013, es decir, el PID del proceso hijo, luego el proceso A ejecuta el conjunto de instrucciones del bloque `else`. Así en primer lugar se imprime en pantalla el mensaje:

```
Soy el padre PID=2012
```

A continuación el proceso resta 1 a la variable `x` y almacena su valor en la misma posición de memoria. Finalmente se muestra en la salida estándar el valor de `x` (en este caso -1) y el mensaje `Mensaje 2`, y el proceso A finaliza.

Nótese que el proceso padre y el proceso hijo tienen espacios de direcciones idénticos pero independientes. Luego los cambios que realice el proceso hijo en sus variables no afectan a los valores de las variables homónimas del proceso padre y viceversa.

ii) El planificador planifica primero al proceso padre A.

En este caso la variable `y` contiene el valor 2013, es decir, el PID del proceso hijo, luego el proceso A ejecuta el conjunto de instrucciones del bloque `else`. Así en primer lugar se imprime en pantalla el mensaje:

```
Soy el padre PID=2012
```

A continuación el proceso resta 1 a la variable `x` y almacena su valor en la misma posición de memoria. Finalmente se muestra en la salida estándar el valor de `x` (en este caso -1) y el mensaje `Mensaje 2`, y el proceso A finaliza.

Supóngase que finalizado el proceso A se planifica para ejecución el proceso hijo B. En ese caso la variable `y=0` luego el proceso B ejecuta el conjunto de instrucciones del bloque `else if`.

Así en primer lugar se imprime en pantalla el mensaje:

```
Soy el hijo PID=2013, mi padre tiene PID=534
```

Nótese que como el proceso A ya había finalizado el padre de B ha pasado a ser el proceso con PID=534 asociado al intérprete de comandos.

A continuación el proceso B suma 1 a la variable `x` y almacena su valor en la misma posición de memoria. Finalmente se muestra en la pantalla el valor de `x` (en este caso 1) y el mensaje `Mensaje2`, y el proceso B finaliza.