# AMPLIACIÓN DE SISTEMAS OPERATIVOS (71023016) Enero 2023

| | |
|---|---|
| Material permitido: **Ninguno**<br><br>Tiempo: **120 minutos**<br>**N1** | **Aviso 1**: Todas las respuestas deben estar debidamente razonadas.<br>**Aviso 2**: Escriba con buena letra y evite los tachones.<br>**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)* ¿Qué es un *archivo mapeado en memoria*? ¿Qué ventajas e inconvenientes presenta su uso?

b) *(1 p)* Explicar cómo se integran los *dispositivos de* E/S dentro del sistema de archivos en los SOBUNIX.

c) *(1 p)* ¿Qué inconveniente presentaba el *planificador* implementado en las *primeras versiones* de Linux? ¿Qué planificador se introdujo para solucionar dicho inconveniente?

d) *(1 p)* Enumerar y describir brevemente los *mecanismos de comunicación entre procesos* multihilos disponibles en Windows.

**2.** *(2 p)* Enumerar las principales acciones que realiza la función del núcleo que se encarga de tratar la llamada al sistema `exit` en los SOBUNIX.

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

```
pedro90:x:101:10:Pedro Blanco:/home/pedro90:/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**<br><br>Tiempo: **120 minutos**<br>**N1** | **Aviso 1**: Todas las respuestas deben estar debidamente razonadas.<br>**Aviso 2**: Escriba con buena letra y evite los tachones.<br>**Aviso 3**: Solución del examen y fecha de revisión en http://www.uned.es/71023016/ |

**4.** A continuación se muestra el código C de los programas `e23` y `datosm`:

```
/*Código del programa e23*/
main()
{
    if(fork()==0)
    {
        execv("datosm",0);
        printf("\n FA23");
    }
    printf("\n 00FE");
}

/*Código del programa datosm*/
main()
{
    printf("\n mensaje3");
}
```

**Figura 1** – Código C de los programas `e23` y `datosm`

Supóngase que al invocar este programa desde la línea de ordenes de un intérprete de comandos se crea un proceso con PID=8099 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 `e23` tiene asignado un PID= 7224. 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) `fork();`

   ii) `execv("datosm",0);`

b) *(1 p)* Explicar el funcionamiento del programa `e23`.

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

## Solución Ejercicio 1

a) Un *archivo mapeado en memoria* es una región del espacio de direcciones virtuales de un proceso en la que se establece una correspondencia byte a byte con parte o con la totalidad de un archivo. Una vez mapeado en memoria el acceso a un determinado byte del archivo puede ser realizado como a cualquier otro contenido del espacio de direcciones virtuales, es decir, indicando la dirección virtual oportuna. Se evita así tener que realizar llamadas al sistema.

   Aparte del ahorro de memoria principal y de la rapidez de los accesos, otra de las ventajas de lo archivos mapeados en memoria es que se pueden usar como mecanismo IPC ya que los cambios que realice un proceso en el archivo mapeado son visibles por todos los procesos que mapeen dicho archivo en sus espacios de direcciones.

   El uso de archivos mapeados también presenta algunos inconvenientes. En primer lugar el tamaño de los archivos que se pueden mapear por completo queda limitado por el tamaño del espacio de direcciones virtuales del proceso, el cual suele menor de 4 GiB en arquitecturas de 32 bits. Para archivos que superan dicho tamaño, el archivo debe ser dividido en trozos cuyo mapeado debe irse alternando en memoria principal. También el tamaño del espacio de direcciones virtuales limita el número de archivos que pueden estar mapeados total o parcialmente simultáneamente en memoria.

   Otro inconveniente está relacionado al uso del mapeo de archivos como mecanismos IPC ya que al igual que sucedía con las regiones de memoria compartida es necesario utilizar algún mecanismo IPC adicional (semáforos o cola de mensajes) para sincronizar el acceso de los diferentes procesos al archivo mapeado.

b) Los SOBUNIX integran los dispositivos de E/S dentro del sistema de archivos principal, generalmente en el directorio `/dev`, para ello asignan a cada dispositivo un archivo de dispositivo modo bloque o modo carácter. Esta integración permite operar con los dispositivos de E/S usando las mismas llamadas al sistema que se utilizan para operar sobre los archivos: `open`, `read`, `write`, etc.

c) El inconveniente que presentaba el planificador implementado en las primeras versiones de Linux es que no estaba bien escalado ya que el tiempo empleado en seleccionar la tarea de mayor prioridad dependía del número de tareas en el estado ejecutándose y del número de procesadores existentes en el computador.

   Durante el desarrollo de la versión 2.5 de Linux el planificador fue rediseñado por completo con objeto de asegurar que el tiempo de selección de la tarea de mayor prioridad fuese independiente del número de tareas en el estado ejecutándose y del número de procesadores. A dicho planificador se le conoce con el nombre de *planificador* O(1)

d) Windows implementa, entre otros, los siguientes mecanismos de comunicación entre procesos multihilos:

   - *Tuberías*. El funcionamiento de las tuberías implementadas en Windows es similar al de los SOBUNIX. La principal novedad de la implementación de las tuberías en Windows es que existen dos modos de funcionamiento:
     - *Modo byte*. Por la tubería se transmite un flujo no estructurado de datos (bytes) con un tamaño máximo fijo. Este modo coincide con la implementación de las tuberías en los SOBUNIX.

    - *Modo mensaje*. Por la tubería se transmite un flujo de bytes estructurado en mensajes de un cierto tamaño. Por ejemplo un flujo de bytes de 256 bytes puede estructurarse en dos mensajes de 128 bytes cada uno.
  - *Conectores* (sockets). Se crean usando la función `Socket`. Si un hilo desea conectarse a un conector ya creado debe usar la función `Connect`. Una vez conectado si desea enviar información debe usar la función `Send`, mientras que para recibir información debe usar la función `Recv`. Cuando se termina de usar un conector debe cerrarlo utilizando la función `CloseSocket`.
  - *Mailslots*. Son un mecanismo de comunicación unidireccional entre procesos multihilos importado del sistema operativo OS/2, permiten enviar un mismo mensaje a múltiples receptores. Sus principales inconvenientes es que son unidireccionales y que no generan de forma explicita un mensaje de confirmación de recepción del envío.
  - *Llamadas a procedimientos remotos* (Remote Procedure Calls, RPCs). Son un mecanismo de comunicación entre procesos que permite a un proceso A invocar a un procedimiento implementado en otro proceso B de la misma máquina local o de una máquina remota. El proceso B ejecuta dicho procedimiento y devuelve el resultado al proceso A. Este mecanismo se suele implementar mediante paso de mensajes de acuerdo a una estructura del tipo cliente-servidor.
  - *Objetos compartidos*. Algunos objetos de Windows pueden ser compartidos por varios procesos. Este es el caso de un *objeto sección* que permite mapear un archivo dentro del espacio de direcciones virtuales de cada uno de los procesos que comparten el objeto. Los cambios que realice un proceso sobre el archivo asociado al objeto sección dentro de su espacio de direcciones serán visibles para todos los procesos que comparten el objeto.

## Solución Ejercicio 2

Independientemente del motivo de la terminación de un proceso A, el sistema operativo realiza esta tarea ejecutando una rutina del núcleo denominada `exit()` en la mayoría de SOBUNIX. Las acciones que realiza `exit()` depende de la implementación de dicha función en cada SOBUNIX. En general `exit()` suele realizar, entre otras, las siguientes acciones:

- Deshabilitar el tratamiento de las señales.
- Cerrar los archivos abiertos por el proceso A.
- Almacenar en la entrada de la tabla de proceso asignada al proceso A los datos estadísticos de uso de recursos y el estatus de salida.
- Poner al proceso A en el estado zombi.
- Hacer que el proceso `init` adopte a los procesos hijos del proceso A.
- Liberar los recursos de memoria asignados al proceso.
- Enviar una señal `SIGCHLD` al proceso padre del proceso A para notificarle la terminación de su hijo.
- Despertar al proceso padre de A si éste estaba dormido esperando por la terminación de su hijo.

## Solución Ejercicio 3

a) De izquierda a derecha se muestran los siguientes datos: `pedro90` 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`, `101` es el UID, `10` es el GID, `Pedro Blanco` es el nombre completo del usuario, `/home/pedro69` 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 al usuario `pedro90` su línea del archivo debería modificarse de la siguiente forma:

```
pedro90:*x:101:10:Pedro Blanco:/home/pedro69:/bin/bash
```

## 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 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 `execv("datosm",0)` reemplaza las regiones del espacio de direcciones del proceso que invoca la llamada por las regiones del programa `datosm` pasado como argumento. Cuando finaliza la llamada al sistema `exec` el proceso invocador pasa a ejecutar el código del programa invocado.

b) Cuando `e23` es invocado desde la línea de órdenes de un intérprete de comandos, éste crea un proceso A con PID=8099 para ejecutar este programa. En primer lugar el proceso A invoca una llamada al sistema `fork`. Supuesto que la llamada se ejecuta con éxito se crea un proceso B hijo de A. Qué proceso se planificará primero, el padre o el hijo, dependerá del planificador.

Se va a suponer que se ejecuta primero el proceso padre. En dicho caso, después de `fork` el proceso A evalúa la condición del `if`. Como la llamada al sistema devuelve al proceso padre el PID del proceso hijo, la condición del `if` no se cumple por lo que imprime en la salida estándar `00FE` y finaliza su ejecución.

Cuando se planifica el proceso B, éste evalúa la condición del `if`. Como `fork` devuelve 0 al proceso hijo, sí se cumple la condición por lo que invoca a la llamada al sistema `execv`. Esta llamada tiene dos parámetros de entrada el nombre de ruta del archivo ejecutable, en este caso `datosm`, y los argumentos que se pasan al ejecutable, en este caso ninguno. Al atenderse esta llamada al sistema el espacio de direcciones del proceso B es sustituido por el espacio de direcciones construido a partir de `datosm`. Luego, el código del proceso B pasa a ser el de `datosm`. En consecuencia, el proceso B imprime en la salida estándar `mensaje3` y finaliza su ejecución.