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 <https://www.dia.uned.es/71023016/>

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

- a) (1 p) Explicar la implementación de los semáforos como mecanismos IPC del System V.
 - b) (1 p) Explicar cómo se inicializa en los SOBUNIX un marco de página de la memoria principal en función del tipo de región a la que pertenece la página del espacio de direcciones virtuales de un proceso que se va alojar en dicho marco.
 - c) (1 p) Enumerar y explicar brevemente la información contenida en una estructura `task_struct` de Linux.
 - d) (1 p) Enumerar las características principales del sistema operativo Windows.
- 2.** (2 p) Enumerar las estructuras de datos que debe mantener el núcleo de un SOBUNIX por cada archivo abierto por un proceso. Describir **adecuadamente** el contenido de cada una de estas estructuras.
- 3.** (2 p) Explique **razonadamente** cuál sería el resultado de la ejecución de las siguientes órdenes:
- a) `ps -Al`
 - b) `ipcs`
 - c) `ls -l`
 - d) `read -p "ans:" v`
 - e) `cp exer[12] test`

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 <https://www.dia.uned.es/71023016/>

4. (2 p) En la Figura 1 se muestra el código C del programa f25. Supóngase que al invocar este programa desde la línea de ordenes de un intérprete de comandos en un SOBUNIX se crea un proceso con PID=334 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 f25 tiene asociado PID=225.

```
#include <signal.h>
main()
{
    int a,b;
    if ((a=fork())==0)
    {
        while(1)
        {
            a=a^2;
            b=raise(SIGCHLD);
            printf("\nMensaje 2[%d]: %d: %d\n",getpid(),b,a);
            sleep(3);
        }
    }
    sleep(8);
    kill(a,SIGUSR1);
    sleep(1);
    kill(getppid(),SIGUSR2);
}
```

Figura 1

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) `if ((a=fork())==0)`
 - ii) `kill(getppid(),SIGUSR2);`
- b) (1 p) Explicar el funcionamiento de f25 si se invoca desde el intérprete de comandos mediante la orden: f25

AMPLIACIÓN DE SISTEMAS OPERATIVOS (Cód. 71023016)

Solución Examen Febrero 2025

Solución Ejercicio 1

- a) Un semáforo implementado como mecanismo IPC es una ampliación del concepto de semáforo general que consiste en un conjunto de N semáforos ($N = 1, 2, 3, \dots$) sobre el cual es posible realizar diferentes operaciones, ya sea sobre todos o algunos de sus elementos.

Para gestionar los conjuntos de semáforos existentes el núcleo mantiene una tabla. Cada entrada de esta tabla contiene, entre otros, los siguientes datos sobre un determinado conjunto de semáforos: creador, propietario, llave, permisos de acceso, puntero al array de semáforos que forma parte del conjunto y número N de semáforos que forman parte del conjunto.

El núcleo asocia a cada conjunto de semáforos un identificador numérico entero positivo *semid* que lo identifica de manera unívoca y que le permite localizar rápidamente su entrada asociada en la tabla que mantiene para soportar los semáforos.

Por cada semáforo j ($j = 0, \dots, N - 1$) perteneciente al conjunto de semáforos *semid* el núcleo mantiene una estructura con los siguientes datos: el valor actual del semáforo, que es un número entero positivo o cero, el número de procesos esperando a que el semáforo valga cero, el número de procesos esperando a que el semáforo se incremente y el PID del proceso que hizo la última operación sobre el semáforo. También mantiene una cola de procesos dormidos en el semáforo.

- b) Una página i de una región del espacio de direcciones virtuales de un proceso no pasa a existir físicamente hasta que es referenciada por primera vez y se produce un fallo de página. Entonces al tratar el fallo se le asignará un marco j de la memoria principal el cual debe ser inicializado adecuadamente en función de la región a la que pertenezca la página i . Si la página i pertenece a la región de código o a la parte inicializada de la región de datos entonces el marco j se inicializa a partir del archivo ejecutable almacenado en la memoria secundaria asociado al proceso. Asimismo si la página i pertenece a la parte no inicializada de la región de datos, al montículo o la región de pila entonces el marco j se inicializa con ceros. En el caso de que la página i pertenezca a una región de librería entonces el marco j se inicializa a partir del archivo de la librería en memoria secundaria.
- c) La estructura `task_struct`, también denominada *descriptor del proceso*, contiene datos y punteros a otras estructuras de datos que el núcleo necesita conocer para poder gestionar una tarea. Esta estructura contiene la siguiente información:

- *Estado de la tarea.*
- *Parámetros de planificación.* Entre ellos la prioridad de la tarea, el tiempo de ejecución y el tiempo promedio que duerme una tarea esperando por un evento. El planificador utiliza esta información para decidir que tarea será planificada para ejecución en un determinado procesador del computador.
- *Identificadores.* Cada tarea tiene asociada un *identificador de tarea* (task identifier, TID), también denominado *identificador del hilo* (thread Identifier), que la identifica de manera única.
En el caso de procesos multihilos cada hilo constituye una tarea y tiene asociado un TID diferente. Además cada tarea tiene asignado un *identificador de grupo de tareas* (Task Group Identifier, TGID), también denominado *identificador de grupo de hilos*, el cual se almacena en el campo `tgid` de la estructura `task_struct`.
- *Información genealógica.* Como por ejemplo: un puntero a la estructura `task_struct` de su proceso padre, la cabecera de la lista doblemente enlazada de sus procesos hijos, y punteros para implementar la lista doblemente enlazada de sus procesos hermanos.

- *Localización del espacio de direcciones virtuales de la tarea.*
- *Credenciales.* Como los identificadores de usuario (UID y EUID), los identificadores de grupo (GID y EGID) y el identificador de sesión (SID).
- *Contexto hardware salvado.* Permite continuar con la ejecución de la tarea en modo usuario tras retornar de la ejecución en modo núcleo de una llamada al sistema o de una interrupción.
- *Puntero a la estructura `thread_info`.*
- *Información relativa al sistema de archivos.* Como un puntero a la tabla de descriptores de archivos abiertos y punteros al nodo-*i* del directorio de trabajo actual y al nodo-*i* del directorio raíz.
- *Información asociada al tratamiento de señales.* Como las máscaras de señales ignoradas, bloqueadas y pendientes. Así como las acciones establecidas para cada tipo de señal y las direcciones de los manejadores de las señales que son capturadas.
- *Información contable sobre uso de recursos.* Como el tiempo ejecutado en modo usuario o en modo núcleo. También límites máximos a los recursos que puede utilizar: número máximo de archivos abiertos, tiempo máximo de uso del procesador, tamaño máximo de su pila o de su segmento de datos, número máximo de marcos de página que puede consumir, etc.

d) Las características principales del sistema operativo Windows son:

- Es un sistema operativo multiprogramado y multitarea, con soporte para multiprocesamiento.
- El núcleo está escrito en su mayor parte en lenguaje C.
- El núcleo posee una estructura de tipo monolítica o simple.
- La planificación se realiza a nivel de hilos del núcleo usando un algoritmo de planificación basada en múltiples colas de planificación y realimentación.
- Dispone de mecanismos de sincronización y comunicación entre procesos o hilos: semáforos, eventos, tuberías, conectores, objetos compartidos, mailslots, etc.
- No suelen utilizar ninguna estrategia de tratamiento de interbloqueos. Dichas estrategias si se quieren implementar deben hacerse a nivel de los programas de aplicación.
- Administra la memoria principal e implementa la memoria virtual usando la técnica de demanda de página.
- Considera a los archivos como una secuencia de bits y delegan en las aplicaciones la interpretación de los mismos.
- Soporta tanto el acceso secuencial como el acceso aleatorio a los archivos.
- En los nombres de archivos no distingue entre minúsculas y mayúsculas.
- Cada sistema de archivos presente en memoria secundaria es tratado como una unidad o volumen lógico independiente, cada una de los cuales posee su propia estructura de directorios. Windows asigna una letra (C, D, E,...) a cada unidad.
- Dentro de cada volumen soporta una estructura de directorios de gráfica acíclica. El directorio raíz de un volumen se denota por el carácter ' \ ', que además se usa para separar los componentes del nombre de ruta de un archivo. Para acceder al sistema de archivos de otra unidad se debe especificar al comienzo de la ruta la letra de la unidad seguido de dos puntos, por ejemplo: C:\.
- La interfaz con el usuario es del tipo interfaz de usuario gráfica (GUI).

Solución Ejercicio 2

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. Nótese que si un proceso abre el mismo archivo dos veces, o si dos procesos abren el mismo archivo, el núcleo creará una estructura de objeto abierto y un descriptor de archivo en cada invocación de la llamada al sistema `open`. Luego se tendrían dos objetos de archivo abierto, cada una con su correspondiente descriptor de archivo.

La llamada al sistema `open` devuelve el descriptor del archivo abierto al proceso que invocó la llamada. Para poder operar sobre un archivo abierto el proceso debe pasar el descriptor del archivo como argumento de las llamadas al sistema asociadas a las operaciones que desee realizar sobre el archivo.

Todos los descriptores de archivos asociados a un proceso se indexan en una tabla denominada *tabla de descriptores de archivos*. Esta tabla es local a cada proceso, es decir, cada proceso tiene su propia tabla de descriptores de archivos. Luego un mismo descriptor de archivo puede referirse en dos procesos distintos a archivos diferentes.

Cada entrada de una tabla de descriptores de archivos contiene un puntero a la estructura de objeto de archivo abierto a la que está asociado el descriptor, y una serie de indicadores que especifican el tratamiento que debe dar al descriptor ante la aparición de determinados eventos. Por construcción las tres primeras entradas (entradas 0 a 2) de la tabla de descriptores de archivos de un proceso están asociadas a los siguientes archivos: archivo estándar de entrada (descriptor 0), archivo estándar de salida (descriptor 1) y archivo estándar de salida de errores (descriptor 2). Por defecto el archivo estándar de entrada está asociado al teclado y el archivo estándar de salida está asociado a la pantalla.

Por su parte, 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. Recuérdese que en SOBUNIX el acceso al contenido de un archivo se realiza por defecto de forma secuencial.
- *Contador de referencias*. Especifica el número de descriptores de archivos que hacen referencia al objeto de archivo abierto. Normalmente el contador vale 1, pero puede tomar valores superiores si se duplica el descriptor usando las llamadas al sistema `dup` o `dup2`. También cuando un proceso padre crea un proceso hijo se duplican los descriptores de archivos del proceso padre para asignárselos al proceso hijo. De esta forma, el proceso hijo puede acceder a los archivos abiertos por el proceso padre.
- *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. El modo de apertura se especifica como un argumento de la llamada al sistema `open`, bien en la forma de un número octal similar en estructura a la máscara de modo de un archivo o en la forma de una constante simbólica. Cada vez que se va a realizar una operación sobre el archivo abierto el núcleo comprueba el modo de apertura del archivo para permitir o denegar la realización de la operación.
- *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. La utilidad y contenido de esta estructura de datos se describirá en la sección 5.3.1.

Un *nodo virtual* o *nodo-v* es una estructura de datos que representa a un archivo activo. El núcleo asigna un *nodo-v* cuando se crea o se abre un archivo durante el tratamiento de las llamadas al sistema `create`

u open. El descriptor del archivo tiene un puntero al objeto de archivo abierto, el cual tiene a su vez un puntero al nodo-v asociado al archivo (ver Figura 5.1).

Nótese que si dos procesos A y B abren el mismo archivo (o si el mismo proceso abre dos veces el mismo archivo) existirán dos descriptores de archivos y dos estructuras de objeto abierto distintas, pero solo una estructura nodo-v.

Solución Ejercicio 3

- Muestra un listado con más información (-l) de todos (-A) los procesos existentes en el sistema.
- Muestra información sobre los recursos IPCs existentes.
- Muestra un listado un listado largo de los archivos y directorios del directorio de trabajo incluyendo tamaño, permisos, propietario, permisos, etc.
- Muestra en la pantalla el mensaje `ans:` y se queda a la espera de que el usuario introduzca por el teclado una cadena de caracteres y pulse la tecla de retorno de carro. Cuando esto suceda la cadena introducida se almacenará en la variable `v`.
- Copia los archivos `exer1` y `exer2` en el subdirectorio `test`.

Solución Ejercicio 4

- La llamada al sistema `fork` crea un proceso hijo. Esta llamada devuelve al proceso padre el PID del hijo y al proceso hijo el valor 0. El valor devuelto se almacena en la variable `a` del espacio de direcciones del padre y en la variable `a` del espacio de direcciones del hijo. Esta sentencia condicional `if` evalúa si la variable `a` es igual a 0, por lo tanto solo es cierta cuando se ejecuta el proceso hijo.
 - La llamada al sistema `kill` envía la señal `SIGUSR2` al proceso con PID igual al resultado de ejecutar la llamada al sistema `getppid()`, es decir, al proceso padre del proceso que invoca a `kill`.
- Al escribir la orden `f25` se comienza a ejecutar el programa `f25`. Supóngase que a la ejecución de dicho programa se le asocia el proceso A, cuyo PID=334 según el enunciado.

El proceso A hace una llamada al sistema `fork` para crear un proceso hijo (proceso B) al cual se le asigna, de acuerdo con el enunciado, PID= 335. La llamada al sistema `fork` devuelve 0 al proceso hijo y 335 al proceso padre. Estos valores se almacenan en la variable `a` del padre y del hijo, respectivamente.

Depende del planificador qué proceso se ejecuta después del `fork`, el padre A o el hijo B. Supóngase que se ejecuta primero el proceso hijo B. Entonces entra en un bucle `while` infinito, en cada pasada del bucle se realizan las siguientes acciones: en primer lugar se realiza una operación binaria XOR ($a=a\oplus 2$), es decir, $a = a \oplus 2 = \bar{a} \cdot 2 + a \cdot \bar{2}$, entre la variable `a`, que contiene el valor 0 en la primera pasada del bucle, y el número 2, almacenando el resultado en `a`.

En segundo lugar se invoca a la llamada al sistema `raise` que envía al proceso B la señal `SIGCHLD`, la acción por defecto asociada a esta señal es ignorarla, por lo que el proceso B no se ve perturbado por dicha señal. Esta llamada al sistema devuelve un 0 si se ejecuta con éxito y -1 en caso contrario. Se va a suponer que se ejecuta con éxito.

En tercer lugar se imprime en pantalla el mensaje

```
Mensaje 1[335]: 0: 2
```

si se trata de una iteración impar del bucle, o el mensaje

```
Mensaje 1[335]: 0: 0
```

si se trata de iteración par.

En cuarto lugar invoca a la llamada al sistema `sleep` que suspende la ejecución del proceso B durante 3 segundos.

Cuando se planifica el proceso padre A invoca a la llamada al sistema `sleep` que suspende su ejecución durante 8 segundos. Cuando reanuda su ejecución invoca a la llamada al sistema `kill` para enviar la señal `SIGUSR1` al proceso hijo B. Esta señal tiene establecida como acción por defecto la terminación del proceso que la recibe.

A continuación el proceso A invoca a la llamada al sistema `sleep` que suspende su ejecución durante 1 segundos. Finalmente invoca a la llamada al sistema `kill` para enviar la señal `SIGUSR2` a su proceso padre, que es el intérprete de comandos (PID=225). Esta señal tiene establecida como acción por defecto la terminación del proceso que la recibe. Por lo que el intérprete de comandos desde donde se invocó al programa `f25` termina y se cierra su ventana.

Debe tenerse en cuenta que durante los 8 segundos que duerme el proceso padre A al proceso hijo B le da tiempo a mostrar por pantalla los siguientes tres mensajes antes de que se cierre la ventana del intérprete de comandos:

```
Mensaje 2[335]: 0: 2
```

```
Mensaje 2[335]: 0: 0
```

```
Mensaje 2[335]: 0: 2
```