SISTEMAS OPERATIVOS II **Septiembre 2013**

Material permitido: **Calculadora NO programable** Tiempo: **2 horas** N	**Aviso 1**: Todas las respuestas deben estar debidamente **razonadas**. **Aviso 2**: Escriba sus respuestas con una **letra lo más clara posible**. **Aviso 3**: Evite los **tachones**. **Aviso 4**: Solución del examen y fecha de revisión en la página web de la asignatura: **http://www.uned.es/533032/**

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

I) (*1 p*) En UNIX un determinado fichero puede estar simultáneamente en dos directorios distintos.

II) (*1 p*) En UNIX si un proceso referencia a una página *i* que ya ha sido colocada por el ladrón de páginas en la lista de página a intercambiar a memoria secundaria entonces el núcleo elimina la página *i* de dicha lista y la página no se intercambia a memoria secundaria.

2. (*2 p*) Explique **razonadamente** la información que contiene el superbloque del sistema de ficheros del UNIX System V.

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

a) (*1 p*) Explique la información que contienen los siguientes cuatro campos de una entrada de la tabla de procesos del sistema UNIX BSD4.3: `p_pri`, `p_usrpri`, `p_cpu` y `p_nice`.

b) (*1 p*) Explique el modo en que se utilizan los campos `p_pri` y `p_usrpri`.

4. En un determinado instante de tiempo la cadena de directorios contenida en la variable de entorno `PATH` de un sistema UNIX es:

```
/bin:/usr/bin:
```

a) (*0.75 p*) ¿Para que se utiliza la variable `PATH`?

b) (*0.75 p*) ¿Qué orden habría que escribir en el intérprete de comandos para añadir el directorio de trabajo actual al final de la cadena de directorios contenida en la variable `PATH`?

5. En el directorio de trabajo actual residen los ficheros `fabr.txt` y `s13` (ver Cuadro 1). Conteste razonadamente a los siguientes apartados:

a) *(1 p)* Explique el significado de las cuatro sentencias enumeradas (**[ ]**) del código de `s13`.

b) *(1.5 p)* Supuesto que se posee permiso de lectura sobre el fichero `fabr.txt` explique **detalladamente** el funcionamiento de este programa si se invoca desde el intérprete de comandos ($) mediante la orden: `$ s13`

SISTEMAS OPERATIVOS II **Septiembre 2013**

Material permitido: **Calculadora NO programable** Tiempo: **2 horas** N	**Aviso 1**: Todas las respuestas deben estar debidamente **razonadas**. **Aviso 2**: Escriba sus respuestas con una **letra lo más clara posible**. **Aviso 3**: Evite los **tachones**. **Aviso 4**: Solución del examen y fecha de revisión en la página web de la asignatura: **http://www.uned.es/533032/**

```
   #include <stdlib.h>
   #include <stdio.h>
   #include <sys/ipc.h>
   #include <sys/sem.h>

   main()
  {
     int x,u;
     key_t y;
     struct sembuf z[2];

     z[0].sem_flg=0;
     z[1].sem_flg=0;

[1]  y=ftok("fabr.txt",'x');
     if (y==(key_t)-1)
     {
          printf("mensaje A");
          exit(3);
     }

[2]  x=semget(y,2,IPC_CREAT|0600);
[3]  u=semctl(x,0,SETALL,0);

     if(fork()==0)
     {
          z[0].sem_num=0;z[0].sem_op=1;
[4]       u=semop(x,z,1);
          z[0].sem_num=1;z[0].sem_op=-1;
          u=semop(x,z,1);
          printf("\nmensaje Z\n",u);
     }
     else
     {
          if(fork()==0)
          {
               sleep(2);
               z[0].sem_num=0;z[0].sem_op=-1;
               u=semop(x,z,1);
               printf("\nmensaje Y\n",u);
               z[0].sem_num=1;z[0].sem_op=1;
               u=semop(x,z,1);
          }

     }
   }
```

Cuadro 1: Código C del programa s13

SOLUCION EXAMEN SEPTIEMBRE 2013

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

I) (*1 p*) En UNIX un determinado fichero puede estar simultáneamente en dos directorios distintos.

II) (*1 p*) En UNIX si un proceso referencia a una página i que ya ha sido colocada por el ladrón de páginas en la lista de página a intercambiar a memoria secundaria entonces el núcleo elimina la página i de dicha lista y la página no se intercambia a memoria secundaria.

Solución:

I) La entrada de un fichero en un directorio constituye un *enlace duro* (o simplemente un *enlace*) para el fichero. Cualquier fichero puede tener uno o más enlaces, en el mismo o en diferentes directorios. Así un fichero no tiene por qué estar limitado a estar en un único directorio y a tener un único nombre. Los enlaces del fichero son iguales en todas sus formas y son simplemente nombres diferentes para el mismo fichero. El fichero puede ser accedido a través de cualquiera de sus enlaces y no hay forma de distinguir cuál es el enlace original. En conclusión la afirmación es **VERDADERA**.

II) Puesto que el contenido de una página física es válido hasta que ésta es reasignada, el núcleo cuando se produce un fallo de página consulta la lista de marcos de página libres por si alguno de sus marcos de página contuviera aún la página que necesita para evitar así tener que leerla del dispositivo de intercambio. No obstante, la página será, de todos modos, intercambiada si ya se ha colocado en la lista de páginas que deben ser transferidas. En conclusión la afirmación es **FALSA**.

2. (*2 p*) Explique **razonadamente** la información que contiene el superbloque del sistema de ficheros del UNIX System V.

Solución:

El superbloque contiene básicamente información administrativa y estadística del sistema de archivos UNIX System V, como por ejemplo:

- El tamaño en bloques del sistema de ficheros.
- El tamaño en bloques de la lista de nodos-i.
- El número de bloques libres y nodos-i libres.
- El comienzo de la lista de bloques libres.
- La lista parcial de nodos-i libres.

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

a) (*1 p*) Explique la información que contienen los siguientes cuatro campos de una entrada de la tabla de procesos del sistema UNIX BSD4.3: `p_pri`, `p_usrpri`, `p_cpu` y `p_nice`.

b) (*1 p*) Explique el modo en que se utilizan los campos `p_pri` y `p_usrpri`.

Solución:

a) La entrada asociada a un proceso en la tabla de procesos posee los siguientes campos que contienen información relacionada con la prioridad de planificación:

- `p_pri`. Contiene la prioridad de planificación actual.
- `p_usrpri`. Contiene la prioridad de planificación actual en modo usuario.
- `p_cpu`. Contiene el tiempo (en tics) de uso de la CPU.
- `p_nice`. Contiene el *factor de amabilidad*, que es controlable por el usuario.

b) El planificador consulta `p_pri` para decidir qué proceso debe planificar. Cuando un proceso se encuentra en modo usuario, su valor `p_pri` es idéntico a `p_usrpri`. Cuando el proceso despierta después de haber entrado en el estado dormido durante una llamada al sistema, su prioridad es temporalmente aumentada para dar preferencia al procesamiento en modo núcleo. Por este motivo el planificador utiliza `p_usrpri` para salvar la prioridad que debe ser asignada al proceso cuando éste retorne al modo usuario y `p_pri` para almacenar su prioridad en modo núcleo.

4. En un determinado instante de tiempo la cadena de directorios contenida en la variable de entorno `PATH` de un sistema UNIX es:

```
/bin:/usr/bin:
```

a) (*0.75 p*) ¿Para qué se utiliza la variable `PATH`?

b) (*0.75 p*) ¿Qué orden habría que escribir en el intérprete de comandos para añadir el directorio de trabajo actual al final de la cadena de directorios contenida en la variable `PATH`?

Solución:

a) La variable `PATH` es una variable de entorno que especifica las rutas de acceso a los directorios donde el intérprete debe buscar el fichero ejecutable asociado a una determinada orden externa. Esta variable no es usada, sin embargo, en la localización de los ficheros ordinarios.

b) La variable `PATH` contiene una cadena de caracteres con el siguiente formato:

```
ruta1:ruta2:...:rutaN
```

donde `ruta1`, `ruta2`,..., `rutaN` son rutas de acceso a directorios.

Para añadir una nueva ruta `ruta_nueva` al final de la cadena contenida en `PATH`, la orden que hay que usar es:

```
PATH=$PATH:ruta_nueva
```

En el caso que se pide en el enunciado `ruta_nueva` se corresponderá con la ruta del directorio de trabajo actual. Alternativamente también se podría utilizar el carácter '.' que representa al directorio de trabajo actual, es decir la orden sería

```
PATH=$PATH:.
```

5. En el directorio de trabajo actual residen los ficheros `fabr.txt` y `s13`. Conteste razonadamente a los siguientes apartados:

a) *(1 p)* Explique el significado de las cuatro sentencias enumeradas (**[ ]**) del código de `s13`.

b) *(1.5 p)* Supuesto que se posee permiso de lectura sobre el fichero `fabr.txt` explique **detalladamente** el funcionamiento de este programa si se invoca desde el intérprete de comandos ($) mediante la orden: `$ s13`

```
   #include <stdlib.h>
   #include <stdio.h>
   #include <sys/ipc.h>
   #include <sys/sem.h>

   main()
  {
      int x,u;
      key_t y;
      struct sembuf z[2];

      z[0].sem_flg=0;
      z[1].sem_flg=0;

[1]   y=ftok("fabr.txt",'x');
      if (y==(key_t)-1)
      {
            printf("mensaje A");
            exit(3);
      }

[2]   x=semget(y,2,IPC_CREAT|0600);
[3]   u=semctl(x,0,SETALL,0);

      if(fork()==0)
      {
            z[0].sem_num=0;z[0].sem_op=1;
[4]         u=semop(x,z,1);
            z[0].sem_num=1;z[0].sem_op=-1;
            u=semop(x,z,1);
            printf("\nmensaje Z\n",u);
      }
      else
      {
            if(fork()==0)
            {
                  sleep(2);
                  z[0].sem_num=0;z[0].sem_op=-1;
                  u=semop(x,z,1);
                  printf("\nmensaje Y\n",u);
                  z[0].sem_num=1;z[0].sem_op=1;
                  u=semop(x,z,1);
            }

      }
   }
```

Código C del programa s13

Solución:

a) El significado de cada una de las sentencias marcadas con **[]** de este código es el siguiente:

[1] Llamada al sistema `ftok` para crear una llave numérica de 32 bits que permite controlar el acceso a una instancia de un mecanismo IPC. Esta llamada tiene dos parámetros de entrada, "`fabr.txt`" que es la ruta de acceso de un fichero que ya debe estar creado y `'x'` que es un carácter. Si la llamada se ejecuta con éxito en `y`, que es una variable del tipo predefinido `key_t`, se almacenará la llave creada. En caso de error en `y` se almacenará el valor (`key_t`)`-1`.

[2] Llamada al sistema `semget` para obtener o crear un conjunto de dos semáforos. Si la llamada se ejecuta con éxito entonces en `x` se almacenará el identificador entero del conjunto de `2` semáforos asociados a la llave `y`. Si no existe un conjunto de semáforos asociado a dicha llave se crea (`IPC_CREAT`) uno nuevo con permisos de lectura y escritura para el propietario (`0600`). Si la llamada falla en `x` se almacena el valor -1.

[3] Llamada al sistema `semctl` para inicializar con el valor `0` todos los semáforos (SETALL) del conjunto de semáforos con identificador `x`. Si la llamada falla en `u` se almacena el valor -1.

[4] Llamada al sistema `semop` para realizar una operación sobre los elementos del conjunto de semáforos `x`. La operación a realizar se especifica en el array de estructuras `z` de tipo `sembuf`. Si no cuenta con los permisos adecuados la llamada falla y en `u` se almacena el valor -1.

b) Al escribir la orden `$ s13` se comienza a ejecutar el programa `s13.` Supóngase que a la ejecución de dicho programa se le asocia el proceso A.

En primer lugar se le asigna el valor `0` a los elementos `sem_flg` del array de estructuras `z` de tipo `sembuf`. Conviene recordar que las estructuras de tipo `sembuf` se utilizan para especificar las operaciones que se desean realizar sobre un conjunto de semáforos. Los elementos `sem_flg` permiten pasar indicadores a la llamada al sistema `semop`.

En segundo lugar se invoca a la llamada al sistema `ftok` para crear una llave numérica de 32 bits que permite controlar el acceso a una instancia de un mecanismo IPC. Supuesto que se ejecuta con éxito en `y`, que es una variable del tipo predefinido `key_t`, se almacenará la llave creada. En caso de error en `y` se almacenará el valor `(key_t)-1`.

En tercer lugar se comprueba, si se ha producido algún error en la ejecución de la llamada al sistema `ftok`. En dicho caso se ejecuta la función `printf` que muestra por pantalla el mensaje

```
mensaje A
```

y se invoca a la llamada al sistema `exit` para terminar la ejecución del proceso A, devolviendo como código de retorno para su proceso padre el valor 3.

Si `ftok` se ha ejecutado con éxito entonces se invoca a la llamada al sistema `semget` para obtener o crear un conjunto de dos semáforos. Si la llamada se ejecuta con éxito entonces en `x` se almacenará el identificador entero del conjunto de `2` semáforos asociados a la llave `y`. Sino existe un conjunto de semáforos asociado a dicha llave se crea (`IPC_CREAT`) uno nuevo con permisos de lectura y escritura para el propietario (`0600`).

Después se invoca a la llamada al sistema `semctl` para inicializar con el valor 0 todos los semáforos (SETALL) del conjunto de semáforos con identificador `x`.

A continuación se invoca a la llamada al sistema `fork` para crear un proceso hijo (proceso B). Esta llamada devuelve 0 al proceso hijo y el pid del hijo al proceso padre. El proceso que se ejecute primero (padre o hijo) dependerá del planificador.

Supongamos que se ejecuta primero el proceso hijo B, entonces tras asignar el valor 0 al elemento `sem_num` y el valor 1 al elemento `sem_op` de la estructura `z[0]` invoca a la llamada `semop` para realizar una operación V (también conocida como incremento o señal) sobre el semáforo 0 del conjunto x. Como el semáforo 0 de `x` había sido inicializado a 0 tras realizar esta operación toma el valor 1.

A continuación el proceso B tras asignar el valor 1 al elemento `sem_num` y el valor -1 al elemento `sem_op` de la estructura `z[1]` invoca a la llamada `semop` para realizar una operación P (también conocida como decremento o espera) sobre el semáforo 1 del conjunto x. Como el semáforo 1 de x había sido inicializado a 0, el proceso B se bloquea en espera de que algún otro proceso realice una operación V sobre este semáforo.

Por su parte el proceso padre A, invoca de nuevo a la llamada al sistema `fork` para crear otro proceso hijo (proceso C). Esta llamada devuelve 0 al proceso hijo y el pid del hijo al proceso padre. El proceso que se ejecute primero (padre o hijo) dependerá del planificador.

Supongamos que se ejecuta primero el proceso padre A, éste como no tiene más instrucciones para ejecutar finaliza.

Se ejecuta entonces el hijo C que invoca a la llamada al sistema `sleep` y suspende su ejecución durante dos segundos. Cuando despierta tras asignar el valor 0 al elemento

`sem_num` y el valor -1 al elemento `sem_op` de la estructura `z[0]` invoca a la llamada `semop` para realizar una operación P sobre el semáforo 0 del conjunto x. Como el semáforo 0 de x había sido puesto a 1 por el proceso B, esta operación pone a 0 el semáforo 0. A continuación el proceso C ejecuta la función `printf` que muestra por pantalla el mensaje

```
mensaje Y
```

Posteriormente el proceso C tras asignar el valor 1 al elemento `sem_num` y el valor 1 al elemento `sem_op` de la estructura `z[1]` invoca a la llamada `semop` para realizar una operación sobre el semáforo 1 del conjunto x. Como resultado de esta operación pone el semáforo 1 a 0 y despierta al proceso B que estaba bloqueado en el semáforo.

Qué proceso se ejecuta (B o C) depende del planificador. Supuesto que se ejecuta el C, éste simplemente finaliza. Entonces se planifica el proceso B que ejecuta la función `printf`, muestra por pantalla el mensaje

```
mensaje Z
```

y finaliza.