DISEÑO Y ADMINISTRACIÓN DE SISTEMAS OPERATIVOS **Septiembre de 2014**

Material permitido: **Calculadora NO programable.** Tiempo: **2 horas.** N	**Aviso 1:** Todas las respuestas deben estar razonadas. **Aviso 2:** Escriba sus respuestas con una letra **lo más clara posible.** **Aviso 3: No use *Tipp-ex*** o similares (atasca el escáner).

ESTE EXAMEN CONSTA DE 5 PREGUNTAS

1. *(1p)* Describa el resultado de la orden `sem=semget(llave, 3, IPC_CREAT | 0600);`

RESPUESTA:

La orden `sem=semget(llave, 3, IPC_CREAT | 0600);` crea **un array** llamado "`sem`" que contiene **tres semáforos generales** asociados a la **clave** `llave` con permiso de lectura y modificación para el propietario.

2. *(2p)* Explique brevemente para qué usa el núcleo la llamada al sistema `wait` y escriba un pseudocódigo asociado con las principales acciones.

RESPUESTA:

La llamada al sistema `wait` permite sincronizar la ejecución de un proceso (A) con la terminación de alguno de sus hijos. La sintaxis de esta llamada es:
`resultado = wait(direc);`
El único parámetro de entrada `direc` es la dirección de una variable entera donde se almacenará el código de retorno para el proceso padre generado por el algoritmo `exit()` al terminar un proceso hijo.
Un posible pseudocódigo que incluya las principales acciones de la llamada al sistema `wait` es (ver Figura 4-6):

```
algoritmo wait
entrada: dirección de la variable del código de retorno para el proceso padre (A)
salida: identificador pid del proceso hijo (B) y código de retorno para A.
        {
        if (el proceso A que espera no tiene procesos hijo)
                {
                return(error);
                }
        for ( ; ;)
                {
                if (el proceso A que realiza la llamada wait tiene procesos hijos en estado zombie)
                        {
                        Busca un hijo en estado zombie (B)
                        Borra los contenidos de la entrada asociada a B en la tabla de procesos
                        Return( pid del hijo y el código de retorno para A)
                        }
                else
                        {
                        Invocar al algoritmo sleep para pasar a A al estado dormido interrumpible
                        }
                }
        }
```

3. *(2p)* Cuando se produce una interrupción, ¿qué algoritmo invoca el núcleo para el tratamiento de las interrupciones? Describa las principales acciones que realiza dicho algoritmo.

RESPUESTA:

Cuando se produce una interrupción, el núcleo (siempre que el nivel de prioridad asociado a la interrupción (npii) sea mayor que el nivel de prioridad de interrupción actual almacenado en el registro de estado del procesador (npi)), invoca al algoritmo inthand() para el tratamiento de las interrupciones. Este algoritmo realiza principalmente las siguientes acciones:

1) Salvar el contexto del proceso actual.

2) Elevar el nivel de prioridad de interrupción. Es decir, se hace npi=npii. Por tanto, las interrupciones que lleguen con un nivel de prioridad de interrupción igual o menor que npi quedan bloqueadas o enmascaradas temporalmente. De esta forma se logra preservar la integridad de las estructuras de datos del núcleo.

3) Obtener el vector de interrupción. Normalmente, las interrupciones aparte del npii pasan al núcleo alguna información que le permite identificar el tipo de interrupción de que se trata. En un sistema con interrupciones vectorizadas, cada dispositivo suministra al núcleo un número único denominado número del vector de interrupción que se utiliza como un índice en una tabla, denominada tabla de vectores de interrupción. Cada entrada de esta tabla es un vector de interrupción, que contiene, entre otras informaciones, un puntero al manejador o rutina de servicio de la interrupción apropiada.

4) Invocar al manipulador o rutina de servicio de la interrupción.

5) Restaurar el contexto del proceso, una vez que se ha concluido la rutina de servicio de la interrupción.

4 *(2p)* Conteste razonadamente a los siguientes apartados:
a) *(1p)* ¿Qué concepto se implementa a partir de las siguientes líneas de código? Explica para qué sirve, su funcionamiento y las principales ventajas.

```
void spin_lock (spinlock_t *s)
      {
      while (test_and_set(s)!=0)
      }
void spin_unlock (spinlock_t *s)
      {
      *s=0;
      }
```

b) *(1p)* Describe el comportamiento de las funciones que son invocadas en el código anterior.

RESPUESTA:

a) El concepto que se implementa a través del código es el de cerrojo con bucle de espera. Un cerrojo con bucle de espera (spin locks) es una primitiva muy simple que permite el acceso en exclusiva a un recurso. Si un recurso está protegido por un cerrojo, un proceso intentando acceder a un recurso no disponible estará ejecutando un bucle, lo que se denomina espera ocupada, hasta que el recurso esté disponible. Un cerrojo suele ser una variable escalar que vale 0 si el recurso está disponible y que vale 1 si el recurso no está disponible. Poner a 1 el cerrojo significa “cerrar el cerrojo” y ponerlo a 0 significa “abrir el cerrojo”. La variable es manipulada usando un bucle sobre una instrucción atómica del tipo comprobar-configurar.

La mayor ventaja de los cerrojos es que son muy económicos en cuanto a tiempo de ejecución. Cuando no hay disputa por un cerrojo, tanto la operación de cierre como la de apertura típicamente requieren únicamente una instrucción cada una. Resultan ideales para estructuras de datos de uso exclusivo que necesitan ser accedidas rápidamente, como por ejemplo la eliminación de un elemento en una lista doblemente enlazada o mientras se realiza una operación del tipo carga-modificación-almacenamiento de una variable. Por tanto, los cerrojos son utilizados para proteger aquellas

estructuras de datos que no necesitaban de protección en un sistema monoprocesador. Los semáforos utilizan un cerrojo para garantizar la atomicidad de sus operaciones.

b) El comportamiento de las funciones invocadas en las líneas de código es el siguiente: si un proceso A desea obtener el uso de un recurso invocará a la función `spin_lock` para cerrar el cerrojo. La función `test_and_set(s)` comprueba y devuelve el valor pasado del cerrojo `s`. En el momento que el cerrojo sea abierto por otro proceso B, `test_and_set()` devolverá 0 y el proceso saldrá del bucle. Además `test_and_set()` pone el cerrojo a 1 para asegurar al proceso A el uso exclusivo del recurso. Cuando proceso termina de usar un recurso, debe invocar a la función `spin_unlock` para abrir el cerrojo.

5. *(3p)* Conteste razonadamente a los siguientes apartados:
a) *(2p)* Explicar el significado de las sentencias enumeradas (**[ ]**) de este programa.
b) *(1p)* Si el programa es compilado bajo el nombre DyASO y describir su salida cuando se invoca de la siguiente forma "`./DyASO hola`". Suponga que los PIDs de los procesos se asignan de forma consecutiva comenzando desde 1000.

```
        #include <sys/types.h>
        #include <sys/ipc.h>
        #include <sys/shm.h>
        #include <string.h>
        #include <stdio.h>
        #include <stdlib.h>

[1]     void main(int n, char* arg[])
        {
[2]     if (n!=2) printf("Argumentos incorrectos");
        else {
             int pid, shmid, par, estado;
             char *cadena;
             key_t llave;
[3]          llave=ftok(arg[0], 'M');
[4]          shmid=shmget(llave,50*sizeof(char),IPC_CREAT | 0600);
             if (shmid==-1)
                  {
[5]               perror("fallo en shmget");
                  exit(1)
                  };
[6]          cadena=shmat(shmid,0,0);
[7]          if ((pid=fork())==-1) exit(2);
                  if (pid==0)
                  {
                       sleep(2);
[8]                    strcpy(cadena,arg[1]);
[9]                    shmdt(cadena);
                       exit(3);
                  }
                  else
                  {
[10]                   par=wait(estado);
                       printf("El proceso %d dice: %s DyASO\n",pid,cadena);
                  }
             }
        }
```

RESPUESTA

Apartado a):

[1] Se declara la función principal (`main`) que admite como entada un array de cadenas `arg[]` de longitud `n`, siendo el primer parámetro de entrada el nombre del fichero ejecutable.

[2] Si el número de argumentos es distinto de dos (el nombre del ejecutable y un argumento adicional) se muestra por la salida estándar el mensaje "Argumentos incorrectos" seguido de un salto de línea y se terminaría la ejecución del programa. En caso contrario, continuaría la ejecución del bloque de código de la cláusula `else`.
[3] La llamada al sistema `ftok` permite crear una clave para ser utilizada posteriormente por un mecanismo IPC. La clave se encuentra vinculada a un fichero y a un carácter. En este caso, el fichero es `arg[0]`, esto es, el propio fichero ejecutable y el carácter es la letra "`M`".

[4] La llamada al sistema `shmget` permite crear una región de memoria compartida de tamaño suficiente para albergar 50 caracteres (`50*sizeof(char)`) asociada a la clave anteriormente creada y con permisos de lectura y escritura para el propietario.

[5] La función `perror()` muestra por la salida estándar el contenido de la variable global `errno` que almacena el descriptor del error que se ha producido en la llamada al sistema anterior. En caso de que se haya producido un error `perror` mostrará la cadena "`fallo en shmget`" seguida de "`:`" y del mensaje descriptor del error que se ha producido.

[6] La llamada `shmat` asigna un espacio de direcciones virtuales al segmento de memoria cuyo identificador `shmid` ha sido creado en **[4]**. El resultado es un puntero a dicha dirección de memoria virtual que se almacena en la variable `cadena`.

[7] La llamada al sistema `fork()` crea un proceso hijo al que devuelve 0, mientras que al padre le devuelve el pid del hijo que se almacena en la variable `pid`. En caso de producirse un error devolverá `-1`, cumpliéndose entonces la condición de la cláusula `if` y realizando la llamada al sistema `exit()` que finaliza la ejecución del proceso devolviendo al proceso padre un estatus `-1`.

[8] La función de librería `strcpy` copia la cadena que contiene el segundo argumento de entrada (`arg[1]`) en la posición de memoria apuntada por `cadena`, esto es, en la región de memoria compartida.

[9] `shmdt` desconecta la región de memoria compartida del espacio de direcciones virtuales del proceso. La región de memoria compartida seguirá existiendo mientras algún proceso la referencie.

[10] La llamada al sistema `wait()` hace que el proceso padre espere a la finalización del proceso hijo (o a la recepción de otra señal) antes de continuar. Esto garantiza que el hijo se ejecutará antes que el padre termine la instrucción **[10]**.

Apartado b)

El funcionamiento del programa es el siguiente:

Cuando se invoca "`DyASO hola`" desde un intérprete de comandos, se crea un proceso asociado al ejecutable DyASO con PID=1000 y se le pasa como argumento su nombre `DyASO` y la cadena "`hola`".

El programa comprueba que el número de argumentos es correcto, al serlo se crea una llave y a continuación una región de memoria compartida asociada a dicha clave. La región se anexiona al espacio de memoria virtual del proceso DyASO y es apuntada por la variable "`cadena`".

Si no se produce ningún error el proceso se bifurca creándose un proceso hijo con PID=1001. Tras la bifurcación tanto el padre como el hijo compartirán y referenciarán la región de memoria compartida.

El código que ejecutará el proceso hijo es el que se encuentra en la cláusula `if`, esto es, dormirá durante 2 segundos, copiará la cadena de entrada en la región de memoria compartida, liberará la región de memoria compartida del espacio de memoria virtual del proceso y terminará. Como la región de memoria compartida todavía es referenciada por el proceso padre seguirá existiendo.

Por otra parte, el proceso padre ejecutará la cláusula `else` del programa esperando hasta la terminación del proceso hijo para a continuación muestra por salida estándar un mensaje con el pid del proceso hijo y el contenido de la región de memoria compartida.

De este modo la salida del programa será:

```
El proceso 1001 dice: hola DyASO
```