Examen Ordinario de Sistemas Operativos - Mayo de 2015

Grado en Ingeniería Informática

Nombre: *Grupo:*

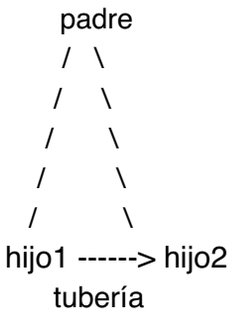
- La fecha de publicación de las notas, así como de revisión se notificarán por Aula Global.
  * Para la realización del presente examen se dispodrá de 2:40 horas.
  * No se pueden utilizar libros ni apuntes
  * Será necesario presentar el DNI o carnet universitario para realizar la entrega del examen

---

Ejercicio 1 (2 puntos) . Autotest.

Ejercicio 2 (3 puntos)

Escribir un programa en C con la siguiente configuración de procesos:



El proceso hijo1 tomará datos del teclado que enviará al proceso hijo2 a través de una tubería, este último mostrará el mensaje recibido por pantalla.

La ejecución de los procesos concluirá cuando el proceso hijo1 reciba la cadena "exit".

Solución

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
#include <string.h>

#define NUM_HIJOS 2 /* número de hijos a crear. */
void hijo1(int fds[2])
```

Nombre: *Grupo:*

```
{
	int numbytes;
	char buf[4096];
	close(fds[0]);
	numbytes = read(STDIN_FILENO, buf, sizeof(buf));
	while (numbytes > 0) {
		if (write(fds[1], buf, strlen(buf)) == -1) {
			perror("fallo en write");
			exit(EXIT_FAILURE);
		}
		if (strncmp("fin\n", buf, strlen("fin\n")) == 0)
			break;
		numbytes = read(STDIN_FILENO, buf, sizeof(buf));
	}
	if (numbytes == -1) {
		perror("fallo en read");
		exit(EXIT_FAILURE);
	}
	close(fds[1]);
}
void hijo2(int fds[2])
{
	int numbytes;
	char buf[4096];
	close(fds[1]);
	numbytes = read(fds[0], buf, sizeof(buf));
	while (numbytes > 0) {
		if (strncmp("fin\n", buf, strlen("fin\n")) == 0)
			break;
		if (write(STDOUT_FILENO, buf, strlen(buf)) == -1) {
			perror("fallo en write");
			exit(EXIT_FAILURE);
```

Nombre: *Grupo:*

```
            }
            numbytes = read(fds[0], buf, sizeof(buf));
        }
        if (numbytes == -1) {
            perror("fallo en read");
            exit(EXIT_FAILURE);
        }
        close(fds[0]);
}

int main(void)
{
        int ret, i, fds[2];
        if (pipe(fds) == -1) {
            perror("fallo en pipe");
            exit(EXIT_FAILURE);
        }
        for (i=0; i<NUM_HIJOS; i++) {
            ret = fork();
            if (ret == 0) {
                switch(i) {
                    case 0:
                        /* tratamiento hijo 1. */
                        hijo1(fds);
                        exit(EXIT_SUCCESS);
                    case 1:
                        /* tratamiento hijo 2. */
                        hijo2(fds);
                        exit(EXIT_SUCCESS);
                }
            } else if (ret > 0) {
                /* tratamiento del padre */
```

Nombre: *Grupo:*

```
            } else if (ret == -1) {
                perror("fallo en fork");
                exit(EXIT_FAILURE);
            }
        }
        // El padre cierra la tubería antes de esperar y salir
        close(fds[0]);
        close(fds[1]);
        ret = wait(NULL);
        while (ret > 0) {
            ret = wait(NULL);
        }
/* si hay error, ignoramos si no hay más hijos a esperar. */
        if (ret == -1 && errno != ECHILD) {
            perror("fallo en wait");
            exit(EXIT_FAILURE);
        }
}
```

Ejercicio 3. (3 puntos)

Disponemos de un sistema Productor-Consumidor, el Productor genera información e incrementa un contador de mensajes, el Consumidor recoge dicha información y posteriormente la difunde, decrementando el contador de mensajes. Los procesos no interaccionan entre sí, excepto a efectos de control. No se dispone de mecanismos para almacenar la información producida por lo que ha de garantizarse en el sistema la alternancia estricta, es decir, cuando un productor genera información ha de esperar para producir nueva información a que esta haya sido recogida por el Consumidor.

Los procesos están programados según el siguiente pseudocódigo:

Examen Ordinario de Sistemas Operativos - Mayo de 2015

Grado en Ingeniería Informática

Nombre: *Grupo:*

procedure Productor is
  while (true) do
  genera_y_almacena;
  incrementar;
  end while;
end procedure;
	procedure Consumidor is
	  while (true) do
	  obten_y_envía;
	  decrementar;
	  end while;
	end procedure;

Las operaciones incrementar y decrementar son atómicas y actúan sobre la misma variable. Se pide:

1. Modifica el código proporcionado para garantizar la alternancia estricta entre el proceso Productor y el proceso Consumidor.(podéis utilizar los semáforos necesarios y las instrucciones genéricas wait y signal)

2. Posteriormente se modifica el sistema disponiendo de la posibilidad de almacenar mensajes por parte del Productor, en espera de ser recogidos por el Consumidor. Si existen procesos pendientes Productor y Consumidor se mantendrá la alternancia estricta, en caso contrario el Productor podrá ceder el turno a otro proceso Productor. Escribe, ayudado de algunas variable enteras compartidas y el correspondiente semáforo la implementación correspondiente.

SOLUCION:

1.

```
semaphore poner :=1;
semaphore quitar:=0;
signal(poner);
```

procedure Productor is
  while (true) do
  genera_y_almacena;
  poner.wait;
  incrementar;
  quitar.signal;
  end while;
end procedure;
	procedure Consumidor is
	  while (true) do
	  obten_y_envía;
	  quitar.wait;
	  decrementar;
	  poner.signal;
	  end while;
	end procedure;

2.

```
semaphore: poner :=0;
semaphore: quitar:=0;
semaphore: mutex:=1;
integer: productores:=0;
```

Examen Ordinario de Sistemas Operativos - Mayo de 2015
Grado en Ingeniería Informática

Nombre: *Grupo:*

```
integer: consumidores:=0;
```

```
procedure Productor is
   while (true) do
      genera_y_almacena;
      mutex.wait;
      productores:=productores + 1;
      if (productores=1 and  consumidores=0) then
        poner.signal;
      end if
     mutex.signal;
     poner.wait;
     incrementar;
     mutex.wait;
     if (consumidores) then
       quitar.signal;
     else
        if (productores) then
          poner.signal;
        else
           quitar.signal;
        end if;
     end if;
     productores:=productores - 1;
     mutex.signal;
   end while;
end procedure;
```

```
procedure Consumidor is
   while (true) do
      obten_y_envia;
      mutex.wait;
      consumidores:=consumidores+1;
      mutex.signal;
      quitar.wait;
      decrementar;
      mutex.wait;
      consumidores:=consumidores-1;
      mutex.signal;
   end while;
end procedure;
```

Ejercicio 4. (2 puntos)

Dado el siguiente sistema de ficheros encadenado basado en el uso de la File-Allocation Table (FAT) y que tiene una reducida capacidad:

- Tamaño de bloque: 1KB.
- Los primeros 10 bloques (del 0 al 9) están reservados para el sistema de ficheros, de los cuales 3 se utilizan para almacenar información del sistema de ficheros (SuperBlock) y del arranque del mismo (Boot). Los restantes 7 bloques se utilizan para almacenar información de los directorios.
- La información de cada directorio ocupa un bloque. Cada entrada del mismo tiene los siguientes campos: nombre del fichero con 64 bytes de extensión; metadatos (nombre de usuario, permisos, etc.) con 63 bytes de extensión; y bloque de comienzo con 1 byte de extensión. Un directorio no puede contener directorios.

Examen Ordinario de Sistemas Operativos - Mayo de 2015

Grado en Ingeniería Informática

Nombre: *Grupo:*

- El sistema de ficheros tiene inicialmente dos directorios llamados (/documentos y /bin) cuyo contenido se muestra a continuación:

Directorio: /documentos

Config.cf	Doc.txt								
Metadata	Metadata								
18	11								

Directorio: /bin

ls	ps	cd	gcc						
Metadata	Metadata	Metadata	Metadata						
38	25	17	13						

- La tabla FAT almacena la información únicamente para datos de los ficheros. Se utiliza el valor -1 para codificar el final del fichero y 0 para indicar que el bloque está libre. A continuación se muestran los contenidos de la tabla FAT completa del sistema de ficheros. El índice superior es el número de bloque. La política de asignación de bloques libres a un fichero nuevo consiste en asignar primero aquellos bloques libres con el menor identificador.

TABLA FAT

10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
-1	15	0	-1	0	29	0	20	-1	0	33	0	0	37	0

25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
23	0	0	0	10	0	-1	0	-1	0	31	0	35	-1	0

Para este sistema de ficheros se pide responder de forma justificada a las siguientes preguntas:

1. ¿Cuál es el tamaño total del sistema de ficheros? ¿Cuál la capacidad efectiva del sistema de ficheros (el espacio en bytes que puedo utilizar para almacenar los datos de los ficheros)?
2. ¿Cuál es el número máximo de directorios que soporta y cuántos ficheros puede tener cada uno de ellos?
3. ¿Qué pueden representar los ficheros almacenados en el directorio /bin? Describe brevemente qué funcionalidad tiene cada uno de ellos.
4. Explique qué bloques almacenan la información del fichero /documentos/Doc.txt. ¿Cuál será el bloque que almacena el byte de datos del fichero 2050?

Nombre: *Grupo:*

5. Indica los cambios en el sistema de fichero que implicarían crear un nuevo fichero, de nombre test.txt en el directorio /documentos con un tamaño de 3 bloques.
6. Un factor importante a la hora de mejorar el rendimiento de un sistema de ficheros es intentar minimizar el número de accesos a las estructuras de datos del mismo, con el fin de reducir el tiempo de localización de un determinado bloque de datos. ¿Qué inconveniente tiene este tipo de sistema de ficheros cuando se accede de forma aleatoria (no secuencial) a ficheros de gran tamaño? ¿Sucede el mismo inconveniente para los sistemas de ficheros basados en índices (como es el basado en UNIX con i-nodos)?

SOLUCIÓN

1. La tabla FAT muestra información únicamente de los bloques de datos de fichero. En total, son 30 bloques representando 30KB de capacidad efectiva. A este número hay que añadir 10 bloques más utilizados internamente por el sistema de ficheros, lo que representa un tamaño para el sistema de ficheros completo de 40KB.
2. Debido a que cada directorio ocupa un bloque y de los 10 reservados 3 son utilizados para para almacenar información del sistema de ficheros y del arranque del mismo, quedarían 7 bloques para almacenar directorios, por lo que el número máximo de directorios sería de 7 . Cada directorio ocupa un bloque (1KB) y cada entrada de un fichero ocupa (64+63+1= 128B) por lo que el número máximo de ficheros que se puede almacenar en un directorio sería 1KB/128B=8 entradas.
3. Dado que el nombre del directorio es /bin y el nombre de cada fichero se deben corresponder con ejecutables de UNIX: ls muestra los contenidos de un directorio, ps los procesos del sistema, cd permite cambiar el directorio de la Shell y gcc es el compilador de C.
4. Consultando la tabla de directorios, el bloque de comienzo del fichero es el 11. Accediendo a la tabla FAT se ve que el bloque 11 está encadenado con el 15, el cual a su vez lo está con el 29, el que a su vez está con el 10 que es terminal. Así que el fichero ocupa la siguiente secuencia de 4 bloques: 11 -> 15 -> 29 -> 10. El byte 2050 del fichero pertenece al tercer bloque, por lo que estará almacenado en el bloque 29.
5.

Directorio: /documentos

Config.cf	Doc.txt	text.txt							
Metadata	Metadata	Metadata							
18	11	12							

TABLA FAT

Examen Ordinario de Sistemas Operativos - Mayo de 2015

Grado en Ingeniería Informática

Nombre: _____ *Grupo:*

10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
-1	15	14	-1	16	29	-1	20	-1	0	33	0	0	37	0

25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
23	0	0	0	10	0	-1	0	-1	0	31	0	35	-1	0

6. Uno de los principales problemas de los sistemas de ficheros encadenados es que, para un acceso dado, es necesario recorrer desde el comienzo la secuencia de bloques del fichero en la tabla FAT. Esto origina un mayor tiempo de acceso. En los sistemas basados en índices, al no existir una lista enlazada el tiempo de acceso se puede reducir de forma significativa.