Sistemas Operativos

David Lopez

Contents

1 conceptos basicos **1**
- 1.1 Funciones del s.o . . . 1
- 1.2 partes principales del s.o . . . 2
- 1.3 clasificacion segun su estructura . . . 2
- 1.4 Como lo usamos ? . . . 2
- 1.5 ejemplo de programa que invoca llamadas al sistema . . . 2

2 Archivos y directorios **3**
- 2.1 Conceptos basicos de archivos . . . 3
- 2.2 Conceptos basicos de directorios . . . 4
- 2.3 Servicios de archivos y directorios . . . 5

3 Procesos **6**
- 3.1 Conceptos basicos . . . 6
- 3.2 threads . . . 7
- 3.3 senales . . . 8
- 3.4 Servicios de procesos . . . 8

1 conceptos basicos

Un sistema operativo no es mas que un progama.

1.1 Funciones del s.o

Este programa, no obstante, es bastante caracteristico, pues se encarga de **gestionar los recursos hardware**, ofrece a los demas programas una api de **llamadas al sistema** y proporciona **mecanismos de interaccion con el usuario** como el shell.

Es necesario gestionar los recursos hardware por que podemos tener varios programas ejecutandose a la vez, y se hace mas importante con los sistemas multiusuario. Los programas competiran por los recursos, por lo que sera necesario administrarlos correctamente.

Sobre las llamadas al sistema habra algunos ejercicios para familiarizarnos, hay que tener en cuenta, que a diferencia de un curso de desarrollo sobre el kernel de un sistema operativo, aqui nos centraremos en programas de espacio de usuario y nos limitaremos, al menos por el momento a invocar a

estas llamadas, no a implementarlas en incluirlas en el s.o , cosa que tambien se puede hacer.

1.2 partes principales del s.o

Podemos distinguir tres partes principales: **nucleo o kernel, servicios e interfaz de usuario**

1.3 clasificacion segun su estructura

- Monolitico (como linux): Tiene todos los componentes integrados en un mismo programa
- Por capas: cada capa ofrece una interfaz a la capa siguiente
- Cliente-Servidor (microkernel): el kernel se reduce a la minima expresion y el resto de utilidades se desarrollan en proceso de usuario

1.4 Como lo usamos ?

Podemos poner el sistema operativo a trabajar, deliberadamente y para que haga la funcion que queremos, escribiendo un programa que realize llamadas al sistema, las interrupciones de los perifericos y los errores tambien provocara la activacion del s.o

1.5 ejemplo de programa que invoca llamadas al sistema

escribir una funcion en C, que actue como el comando cat de unix, es decir, que imprima el contenido del archivo por la salida estandar

```
1  #include <stdio.h>
2  #include<fcntl.h>
3  #include<unistd.h>
4
5  #define BUFFSIZE 512
6
7  void main(int arg, char *argv[]){
8
9    int fd = 0;
10   int cnt = 0;
```

```
11    char buff[BUFFSIZE];
12
13    fd = open(argv[1], O_RDONLY);
14
15    while((cnt = read(fd, buff, BUFFSIZE)) > 0){
16      write(1, buff, cnt);
17    }
18
19    close(fd);
20
21
22
23
24  }
```

En este codigo, las funciones *open, read, write y close* son llamadas al sistema, cuyo man se puede encontrar en: manual de llamadas al sistema, es fundamental saber utilizar esta herramienta y manejar con solvencia las llamadas al sistema mas habituales.

2 Archivos y directorios

2.1 Conceptos basicos de archivos

El sistema operativo nos oculta los detalles de la implementacion fisica del disco, proporcionandonos lo que conocemos como archivo.

Que es un archivo: Es una unidad de almacenamiento logico ***no volatil*** *que agrupa un conjunto de informacion relacionada entre si bajo un mismo nombre*

Carateristicas de un archivo:

- Desde el punto de vista de el sistema operativo este debe proporcionar al menos una estructura de archivo que permita realizar sobre el: darle nombre, proteccion y sistema eficiente de gestion de estos mismos. Ademas debe permitir anadirle los atributos que se deseen.
- El nombre y extension son basicos para un archivo y deben permitir que se le distinga inequivocamente
- El usuario debe percibir sencillez para manipular archivos. La traduccion entre las direcciones fisicas y logicas es lo que depende segun el

sistema operativo pero se basa en el **mapa de archivos** que se almacena como atributo.

- Se distinguen dos tipos principales de acceso a la informacion en los archivos: **Secuencial**, si todos los bytes estan seguidos y en orden o **acceso aleatorio o directo** que permiten el acceso por conjuntos y desordenado.

- Cuando se accede de forma concurrente al mismo archivo, se establecen mecanismos tipicos para tratar la concurrencia: Archivos inmutables, sesiones(copias y versiones) y UNIX(cada proceso trabaja con su imagen independiente del archivo y no se permite que compartan el apuntador.)

2.2 Conceptos basicos de directorios

Para poder acceder a los archivos con facilidad, se crean formas de organizar los nombres de los archivos, son los directorios.

Que es un directorio: Es un objeto que relaciona el nombre del usuario de un archivo y el descriptor interno del mismo.

Caracteristicas de los directorios:

- Actualmente los sistemas operativos se organizan en forma de arbol, que representa los directorios y subdirectorios partiendo de una raiz de forma que existe un unico camino a un archivo.

- Para solventar la problematica de que varios usuarios quieran trabajar sobre el mismo archivo (pero tienen rutas hasta el distintas), se generaliza de arbol a grafo aciclico.

- La forma mas habitual de compartir archivos es crear un **enlace** , que puede ser fisico o simbolico. La diferencia es que mientras que uno apunta al mismo archivo el otro crea una copia de los contenidos en uno nuevo.

- En UNIX se utilizan los **i-nodos** para saber cuantos enlaces fisicos tiene un mismo archivo. Esto provoca que haya que controlar los bucles al recorrer todo el arbol e introduce problemas al borrarlos (se solventa disminuyendo el contador, y hasta que no sea 0, no se borra del todo)

- Cuando estamos trabajando desde algun directorio y nos queremos mover o referir a otro directorio o archivo, el sistema operativo ha de tener en cuenta que las **referencias** que hagamos pueden ser **absolutas (desde la raiz)** o **relativas**. Ver ls -a . y ..

2.3 Servicios de archivos y directorios

Cuando abrimos un archivo, lo que obtenemos es un **descriptor de archivo** que en UNIX se conoce como **i-nodo**. Por ejemplo, para la salida estandar hay unos descritores fijos: 0-in, 1-out, 2-err. Los inodos contienen todos los atributos que nos interesaran para nuestro estudio.

Permisos de acceso: bits: setuid(ejecutable-user), setgid(ejecutable-grupo), rwx(user), rwx(grupo), rwx(resto).

Servicios POSIX para ficheros:

- Crear y abrir un fichero: `int creat(const char * path, mode_t mode);`
- Borrar un fichero por su ruta: `int ulink(const char * path);`
- Abrir un fichero por su ruta en un determinado modo: `int open(const char path, int oflag, mode_t mode);`
- Cerrar un fichero por su descriptor (devuelto por alguna de las funciones anteriores): `int close(int fd);`
- Leer **nbytes** de un archivo a **buf**: `ssize_t read(int fd, void *buf, size_t nbytes);`
- Escribir **nbytes** de **buf** a un archivo: `ssize_t write(int fd, const void *buf, size_t nbytes);`
- Mover el apuntador de posiciono de un archivo **offset** desde **whence** que puede tomar los valores: SEEK_SET (inicio), SEEK_CURR (posicion actual) y SEEK_END: `off_t lseek(int fd, off_t offset, int whence);`
- Consultar atributos de un archivo: `int stat(const char *path, struct stat *buf);` , devuelven una estructura de tipo stat (ver esta estructura para saber que atributos podemos obtener a partir de esta llamada)

Servicios POSIX para directorios:

- Crear un directorio: `int mkdir(const char * path, mode_t mode)`
- Borrar un directorio (solo se puede borrar si esta vacio): `int rmdir(const char * path)`
- Abrir un directorio: `DIR * opendir(const char * pathname)`
- Cerrar un directorio: `int closedir(DIR * dirp);`
- Leer la siguiente entrada de un directorio (nombre del archivo e i-nodo): `struct dirent *readdir(DIR * dirp);`
- Comprobar si un archivo en un directorio esta accesible con unos ciertos permisos: `int access( const char * path, int amode );`
- Cambiar los permisos sobre un archivo: `int chmod(const char * path, mode_t mode);`
- Cambiar el propietario y el grupo de un archivo: `int chown(const char * path, uid_t owner, gid_t group);`
- Definir una mascara que sera aplicada por defecto a todos los objetos de un usuario: `mode_t umask(mode_t cmask);`
- Establecer un enlace fisico entre una nueva entrada de directorio y un archivo ya existente: `int link(const char * existing, const char * new);`
- Establecer un enlace simbolico desde una nueva entrada de directorio a un archivo ya existente: `int symlink( const char * existing, const char * new);`

3 Procesos

3.1 Conceptos basicos

Un proceso es la unidad de procesamiento gestionada por el sistema operativo.

El sistema operativo mantiene una tabla de procesos. En esta tabla se almacena un **bloque de control de proceso** *(estructura task_struct en linux)* por cada proceso. El BCP contiene la siguiente informacion:

- Identifiacion: PID, USER y relaciones padre-hijo
- Estado del procesador
- Informacion de control del proceso
- Memoria asignada
- Recursos asignados

3.2 threads

Un determinado proceso puede comenzar a crear hilos y por tanto a desdoblarse, esto implica que la imagen de memoria del proceso original se va a compartir entre desitintos procesos, la imagen de memoria no se replica como en el fork. De esta forma conseguimos una ventaja importante con respecto a los recursos del sistema. Lo que se comparte y no se comparte, se explica mas abajo.

Un proceso puede contener varios threads, o hilos de ejecucion que comparten algunos recursos e informacion entre ellos.

Cada thread se define como una funcion que se puede lanzar en paralelo con otras.

Los threads del mismo proceso comparten parte de su informacion: Memoria, variables globales, archivos abiertos, procesos hijos, temporizadores y semaforos. Que compartan la memoria genera que no haya proteccion de memoria y los threads son independientes unos de otros hasta cierto punto.

Pero no comparten: contador de programa, pila, registros y estado. En resumen, el contexto de ejecucion.

Las ventajas que proporciona este mecanismo de "dividir el trabajo de los procesos en tareas mas sencillas" es claro y sigue el paradigma de divide y venceras, sin embargo, cuando entramos en el terreno de la programacion concurrente hemos de ser cuidadosos por los tipicos problemas que pueden surgir.

3.3 senales

Se puede describir una senal como una interrupcion a un proceso. El comportamiento de las senales es similar al de las interrupciones en la cpu.

Las senales pueden provenir o bien de un proceso que la manda a otro proceso con su mismo *uid* o el sistema operativo ante las tipicas excepciones que vemos cuando fallan nuestros programas (seg fault, /0...)

Hay muchas senales diferentes y se engloban en tres tipos fundamentales: De hardware, de comunicacion y de E/S asincrona.

Para que un proceso pueda enviar una senal ha de estar preparado para recibirla. Para ello tenemos que indicarle al SO el nombre de la rutina del proceso que ha de tratar ese tipo de senal. Si un proceso recibe una senal sin estar preparado para ello, se suele matar al proceso.

Podemos utilizar las senales como el mecanismo mas sencillo de pasar informacion entre procesos, esto es extremadamente rudimentario, pero se puede usar. Asi, podriamos usar *kill -2 pid* donde pid es el pid de un proceso que esta actualmente activo, y -2 tiene el mismo comportamiento que el control-c.

Manejadores de senales: podemos usar la funcion signal para capturar una determinada senal y cambiar el comportamiento que tiene por defecto. Esto se hace con la funcion *signal*

Para poder mandar senales a algun proceso tenemos que ser los que lo creamos.

3.4 Servicios de procesos

Gestion de procesos:

- Identificador de proceso: PID (pid_t)
- Obtener el identificador de proceso: getpid() devuelve el identificador del proceso que hizo la llamada
- Obtener el identificador del proceso padre: getppid()
- Obtener el identificador de usuario real: getuid()
- Obtener el identificador de usuario efectivo: geteuid()
- Obtener el identificador de grupo real: getgid()

- Obtener el identificador de grupo efectivo: getegid()
- Obtener el valor de una variable de entorno: *getenv(const char * name)
- Fijar el entorno de proceso: *setenv(char **env)
- Crear un proceso: fork(). Clona al proceso que lo creo, pasando a ser su hijo
- Ejecutar un programa: familia exec. Cambia el programa que esta ejecutando un proceso
- Terminar la ejecucion de un proceso: exit(int status). Similar a los return
- Esperar a la finalizacion de un proceso hijo: wait(int *status) o wait_pid(pid_t , int * status, int options)
- Suspender la ejecucion de un proceso: sleep(unsigned int seconds)

Gestion de threads:

- Atributos de un thread: (cada thread tiene asociado unos atributos, y los valores de estos se almacenan en un objeto de tipo *pthread_attr_t*
  - Crear atributos de un thread: int pthread_attr_init(pthread_attr_t *attr);
  - Destruir atributos: int pthread_attr_destroy(pthread_attr_t *attr);
  - Asignar tamano al thread: int pthread_attr_setstacksize(pthread_attr_t *attr);
  - Obtener el tamano de la pila: int pthread_attr_getstacksize(pthread_attr_t *attr)
  - Establecer el estado de terminacion: int pthread_attr_setdetachstate(pthread_attr_t * attr, int detachstate). El valor del segundo argumento, indicara la independencia o no del thread con respecto a otros.
  - Obtener el estado de terminacion: int pthread_attr_setdetacstate(pthread_attr_t * attr, int * detachstate)
- Creacion, identificaciony terminacion de threads:

- Crear un thread: int pthread_create(pthread_t *thread, pthread_attr_r *attr, void * (*start_routine) (void *), void * arg). El primer argumento es el indentificador, el segundo atributos de ejecucion, el tercero el nombre de la funcion a ejecutar cuando se lance y el ultimo es un puntero a void.
- Obtener el identificador de un thread: pthread_t pthread_self(void)
- Esperar la terminacion de un thread: int pthread_join(pthread thid, void **value). Es necesario indicar el thread por el que se quiere esperar.
- Finalizar la ejecucion de un thread: int pthread_exit(void * value)