Manejo de archivos

Control de archivos abiertos

Con la función *fcntl* se tiene control sobre un archivo abierto mediante una llamada previa a *open*. Este control va a consistir en las posibilidades de cambiar los modos permitidos de acceso al archivo, y de bloquear el acceso a parte del mismo o a su totalidad. El bloqueo tiene especial importancia cuando varios procesos trabajan simultáneamente con un archivo, y es imprescindible que los accesos a determinados registros del mismo sean atómicos.

- Un cerrojo de lectura indica que el proceso actual está leyendo del archivo, por lo que ningún otro proceso debe escribir en el área bloqueada. Puede haber varios cerrojos de lectura simultáneos sobre una misma región de un archivo.
- Un cerrojo de escritura indica que el proceso actual está escribiendo en el fichero, por lo que ningún proceso debe leer del área bloqueada. Sólo puede haber un cerrojo de escritura sobre una misma área del fichero.

Los cerrojos fijados por un proceso sobre un archivo se borran cuando el proceso termina. Además, los cerrojos no son heredados por los procesos hijos tras la llamada a *fork*. Como un ejemplo se muestra un proceso que realiza un bloqueo de escritura sobre un archivo.

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main (int argc, char* argv[]) {
 char* file = argv[1];
 int fd;
 struct flock lock;

 printf ("opening %s\n", file);
 /* Open a file descriptor to the file. */
 fd = open (file, O_WRONLY);
 printf ("locking\n");
 /* Initialize the flock structure. */
 memset (&lock, 0, sizeof(lock));
 lock.l_type = F_WRLCK;
 /* Place a write lock on the file. */
 fcntl (fd, F_SETLKW, &lock);

 printf ("locked; hit Enter to unlock... ");
 /* Wait for the user to hit Enter. */
```

```
 getchar ();

 printf ("unlocking\n");
 /* Release the lock. */
 lock.l_type = F_UNLCK;
 fcntl (fd, F_SETLKW, &lock);

 close (fd);
 return 0;
}
```

Estadísticas de un archivo

Existen una serie de llamadas al sistema que permiten acceer y cambiar la información del tipo administrativo y estadístico de un archivo. Las llamadas *stat(path,buf)* y *fstat(fildes,buf)* devuelven la información que se almacena en la tabla de inodos sobre el estado de un archivo concreto. La diferencia entre *stat* y *fstat* es que la primera recibe como primer parámetro un puntero al nombre del archivo, mientras la segunda trabaja con un archivo ya abierto y recibe el descriptor de este.

La información administrativa del archivo se almacena en una estructura de tipo *struct stat*. Este tipo está definido en el archivo de encabezados .

```
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
#include <locale.h>
#include <langinfo.h>
#include <stdio.h>
#include <stdint.h>

struct dirent  *dp;
struct stat     statbuf;
struct passwd  *pwd;
struct group   *grp;
struct tm      *tm;
char            datestring[256];

int main(int argc, char* argv[]) {

 char* dir = argv[1];
```

```
/* Loop through directory entries. */
while ((dp = readdir(dir)) != NULL) {


    /* Get entry's information. */
    if (stat(dp->d_name, &statbuf) == -1)
        continue;

    /* Print out type, permissions, and number of links. */
    printf("%10.10s", sperm (statbuf.st_mode));
    printf("%4d", statbuf.st_nlink);

    /* Print out owner's name if it is found using getpwuid(). */
    if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
        printf(" %-8.8s", pwd->pw_name);
    else
        printf(" %-8d", statbuf.st_uid);

    /* Print out group name if it is found using getgrgid(). */
    if ((grp = getgrgid(statbuf.st_gid)) != NULL)
        printf(" %-8.8s", grp->gr_name);
    else
        printf(" %-8d", statbuf.st_gid);

    /* Print size of file. */
    printf(" %9jd", (intmax_t)statbuf.st_size);
    tm = localtime(&statbuf.st_mtime);
    /* Get localized date string. */
   strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
    printf(" %s %s\n", datestring, dp->d_name);
}
}
```

Administración de directorios

A continuación se presentan una serie de funciones que permiten leer el contenido de un directorio sin preocuparse de la estructura del mismo. La función para abrir un archivo de directorio es *opendir(dirname)*. *Dirname* es un puntero al *path name* del directorio que se desea abrir.

Para leer las entradas de un directorio abierto con opendir, se utiliza la función *readdir(dirp)*. *Dirp* es el puentero al directorio ya abierto.

```
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
```

```
#include <string.h>

#define FOUND 0
#define NOT_FOUND 1
#define OPEN_ERROR 2
#define READ_ERROR 4

int main() {

DIR *dirp = opendir(".");
struct dirent* dp;
char name[256];

while (dirp) {
    errno = 0;
    if ((dp = readdir(dirp)) != NULL) {
        if (strcmp(dp->d_name, name) == 0) {
            closedir(dirp);
            return FOUND;
        }
    } else {
        if (errno == 0) {
            closedir(dirp);
            return NOT_FOUND;
        }
        closedir(dirp);
        return READ_ERROR;
    }
}

return OPEN_ERROR;
}
```

Ejercicios

1. Escriba dos programas que escriban simultáneamente de forma alterna a un archivo una línea de números. Los programas deben bloquear y liberar el archivo cuando terminen de escribir. Un archivo escribirá los números pares en grupos de 10 por línea, y el segundo escribirá los números impares en grupos de 10 por línea.
2. Escriba un programa que muestre en forma de árbol los diferentes archivos y directorios de un directorio particular. Debe indicar a la par de cada archivo su tamaño y fecha de creación.