EXPLICACION PRACTICA 1 SO2 2010/2011
20031707-H JOSE MANUEL MORATO ESCANDELL

1. ENUNCIADO DE LA PRÁCTICA

Escribir un programa en C de nombre `practica1.c` que sirva para visualizar en pantalla la máscara de modo simbólica de uno o varios ficheros. El programa para cada fichero especificado como argumento (bien como una ruta absoluta o relativa), debe consultar la máscara de modo que hay en el nodo-i del fichero y generar a partir de dicha información la máscara de modo simbólica.
El programa ejecutable que se compile a partir de `practica1.c` se debe llamar `mask_sim`. A continuación se muestran algunos ejemplos de la funcionalidad que debe tener este programa:

1) Supóngase que `prueba` es un directorio cuya máscara simbólica es `drwxr-xr-x`, entonces la orden `$ mask_sim prueba` debe mostrar en pantalla la salida `prueba: drwxr-xr-x`

2) Supóngase que `fichero` es un fichero ordinario cuya máscara simbólica es `-rws--srwt`, entonces la orden `$ mask_sim fichero` debe mostrar en pantalla la salida`fichero:-rws--srwt`

3) Supóngase que `apunta` es un enlace simbólico cuya máscara simbólica es `lrwxrwxrwx`, entonces la orden `$ mask_sim apunta` debe mostrar en pantalla la salida `apunta: lrwxrwxrwx`

4) Supóngase que `/dev/cua0` es un fichero de dispositivo modo carácter cuya máscara simbólica es `crw-rw-rw-`, que `/dev/fd0` es un fichero de dispositivo modo bloque cuya máscara simbólica es `brw-rw-rw-`, que `/dev/log` es un conector cuya máscara simbólica es `srw-rw-rw-` y que `/dev/printer` es una tubería con nombre cuya máscara simbólica es `prwxrwxrwx`, entonces la orden

```
$ mask_sim /dev/cua0 /dev/fd0 /dev/log /dev/printer
```

debe mostrar en pantalla la salida

```
/dev/cua0: crw-rw-rw-
/dev/fd0: brw-rw-rw- /dev/log: srw-rw-rw-
/dev/printer: prwxrwxrwx
```

2. EXPLICACIÓN

El núcleo guarda toda la información relativa a los ficheros en el nodo-i correspondiente dentro de la tabla de nodos-i. Esta tabla es una copia en memoria de la lista de nodos-i, que se encuentra en el disco entre el superbloque y los bloques de datos, a la que se le añade información adicional. Esta tabla se copia en memoria para acelerar su acceso pero ello conlleva que se tenga que gestionar adecuadamente la sincronización con el disco. Existe un demonio del sistema (syncer) que se encarga de realizar esta sincronización periódicamente.

Las llamadas al sistema que nos permiten acceder y modificar la información administrativa y estadística de un fichero son:

```
#include <sys/stat.h>
int stat (char *path, struct stat *buf);
int lstat (char *path, struct stat *buf);
int fstat (int fildes, struct stat *buf);
```

La funcionalidad entre stat y fstat es exactamente igual, solo se diferencia en que stat recibe como primer parámetro la ruta absoluta o relativa mediante un puntero al nombre del fichero y fstat utiliza un

descriptor de un fichero previamente abierto.

lstat trabaja de forma similar a stat excepto cuando se trata de un enlace simbólico. En ese caso, lstat devolverá la información relativa al fichero que sirve de enlace y stat devolverá la información del fichero al que apunta el enlace. Esta diferencia será importante a la hora de realizar la práctica; se deberá utilizar la llamada a lstat para poder leer la información relativa al enlace y así poder cumplir el requerimiento del punto 3).

Si abrimos el fichero stat.h, además de las declaraciones para stat, lstat y fstat podemos encontrar una serie de constantes que nos ayudarán a la hora de comprobar el campo st_mode de la estructura stat, que contiene la mascara de modo del archivo.

Las siguientes definiciones de nombres nos serán útiles para comprobar el tipo del fichero que estamos examinando:

```
#define  S_IFMT      0170000     /*  type of file mask */
#define  S_IFIFO     0010000     /*  named pipe (fifo) */
#define  S_IFCHR     0020000     /*  character special */
#define  S_IFDIR     0040000     /*  directory */
#define  S_IFBLK     0060000     /*  block special */
#define  S_IFREG     0100000     /*  regular */
#define  S_IFLNK     0120000     /*  symbolic link */
#define  S_IFSOCK    0140000     /*  socket */
```

Para facilitarnos el trabajo y evitar al mismo tiempo malos usos de estas constantes, el archivo stat.h define las siguientes macros:

```
#define  S_ISBLK(m)    (((m) & S_IFMT) == S_IFBLK)     /* block special */
#define  S_ISCHR(m)    (((m) & S_IFMT) == S_IFCHR)     /* char special */
#define  S_ISDIR(m)    (((m) & S_IFMT) == S_IFDIR)     /* directory */
#define  S_ISFIFO(m)   (((m) & S_IFMT) == S_IFIFO)     /* fifo or socket */
#define  S_ISREG(m)    (((m) & S_IFMT) == S_IFREG)     /* regular file */
#define  S_ISLNK(m)    (((m) & S_IFMT) == S_IFLNK)     /* symbolic link */
#define  S_ISSOCK(m)   (((m) & S_IFMT) == S_IFSOCK)    /* socket */
```

Finalmente, para la comprobación de los diferentes derechos del propietario, grupo al que pertenece el propietario y demás usuarios del sistema, el archivo stat.h declara las siguientes definiciones de nombre:

```
/* Read, write, execute/search by owner */
#define  S_IRWXU          0000700          /* RWX mask for owner */
#define  S_IRUSR          0000400          /* R for owner */
#define  S_IWUSR          0000200          /* W for owner */
#define  S_IXUSR          0000100          /* X for owner */
/* Read, write, execute/search by group */
#define  S_IRWXG          0000070          /* RWX mask for group */
#define  S_IRGRP          0000040          /* R for group */
#define  S_IWGRP          0000020          /* W for group */
#define  S_IXGRP          0000010          /* X for group */
/* Read, write, execute/search by others */
#define  S_IRWXO          0000007          /* RWX mask for other */
#define  S_IROTH          0000004          /* R for other */
#define  S_IWOTH          0000002          /* W for other */
#define  S_IXOTH          0000001          /* X for other */
```

Con todo esto, ya disponemos de las herramientas necesarias para afrontar la practica con éxito, por lo tanto pasaré a explicar el código que se puede encontrar el el archivo practica1.c.

En primer lugar incluimos las librerías necesarias para el programa:

- stdio.h: donde encontramos las llamadas principales relacionadas con la entrada/salida
- stat.h: que, como ya explicado, incluye las llamadas al sistema, estructuras de datos y definiciones de nombres para obtener la información administrativa de un fichero.
- types.h: definición de tipos para el núcleo de UNIX.

Después definimos el prototipo de la función ImprimirMascara que toma como argumento una variable de tipo mode_t y muestra por la salida estándar la mascara simbólica relacionada. Esta función no devuelve ningún valor. Más adelante explicaré la implementación de esta función.

En la próxima linea de código encontramos la función main, que sirve como punto de entrada de un programa en c, devuelve un entero indicando el código de salida del programa y toma como argumentos:

- argc: Número de parámetros con el que se ha llamado al programa
- argv: Array de cadenas que contiene los parámetros de entrada al programa

Dentro de la función main declaramos una variable de tipo estructura stat (fileStat) que nos servirá para almacenar la información del fichero al pasársela por referencia a la llamada lstat.

Inmediatamente después comprobamos el número de parámetros con los que ha sido llamado el programa desde la linea de comandos.

En el caso de que argc sea igual a 2 significará que el usuario a pedido información sobre un único fichero que se encontrará en argv[1] (argv[0] siempre contendrá el nombre del ejecutable), por lo tanto llamaremos a lstat para guardar la información en fileStat y en caso de que no se produzca ningún error imprimiremos en pantalla la mascara simbólica al llamar a ImprimirMascara con el parámetro fileStat.st_mode obtenido de la llamada previa a lstat. En caso de error en la llamada a lstat, imprimiremos en pantalla la ruta del archivo, seguido de dos puntos y la descripción del error a través de la función perror.

En el caso de que argc sea mayor que 2 significará que el usuario ha pedido información sobre varios ficheros en una única llamada al programa. Para presentar la información de cada uno de los archivos, iniciamos un bucle que recorrerá todas las posiciones de argv empezando por la segunda posición (indice 1) y para cada parámetro encontrado llamaremos a lstat e imprimiremos en pantalla la ruta del archivo, seguido de dos puntos y seguido de la mascara simbólica obtenida mediante la llamada a la función ImprimirMascara. En el caso de que la función lstat falle para alguno de las rutas de archivos obtenidas en los parámetros del programa, se imprimirá en pantalla la ruta del archivo, seguido de dos puntos y la descripción del error y se continuará con el siguiente archivo.

Por último si el numero de parámetros es menor que 2, se mostrara un mensaje de error y se informara del modo correcto de utilización del programa al usuario.

Justo después de la función main, encontramos la implementación de la función ImprimirMascara. Esta es la función que se encargara de interpretar el valor obtenido en el campo st_mode de la estructura stat e imprimir por pantalla la mascara simbólica del archivo relacionado.

Para ello, primero determinaremos el tipo del fichero examinado con la ayuda de las macros definidas en stat.h. Una vez echo esto, pasaremos a comprobar los permisos de lectura/escritura/ejecución para el propietario, el grupo del propietario y los demás usuarios del sistema, para conserguirlo solo necesitaremos realizar un AND lógico entre la constante predefinida en cada caso y la mascara obtenida en st_mode.