Práctica 6: Sistema de Ficheros

Esta práctica se ha realizado en una máquina virtual corriendo el sistema operativo Ubuntu 15.04 LTS.

Apartado 1: Creación y atributos de ficheros

***Ejercicio 1.*** La herramienta principal para consultar el contenido y atributos básicos de un fichero es ls. Consultar la página de manual y estudiar el uso de las opciones -a -l -d -h -i -R -1 -F y --color. Estudiar el significado de la salida en cada caso.

LS lista los contenidos de un directorio.

Parámetros:

-a, -all Ignora las entradas que comiencen con .

-l, usa el formato de lista extendido

-d, --directory lista solo los directorios, no sus contenidos

.h, --human-readable con -l y/o -s, imprime el tamaño entendible para humanos

-i, --inodo Imprime el i-nodo de cada fichero

-R, --recursive Lista los subdirectorios recursivamente

-1 Lista un fichero por linea

-F, --classify añade un indicador a cada entrada (uno de estos: \*/=>@|)

--color[=WHEN

] Le da color a la salida, WHEN puede ser 'never', 'auto' o 'always'

***Ejercicio 2.*** Los permisos de un fichero son <tipo><rwx\_propietario><rwx\_grupo><rwx\_resto>:

● ‐ fichero; d directorio; l enlace; c dispositivo carácter; b dispositivo bloque; p FIFO; s socket

● r : lectura (4); w :escritura (2); x :ejecución (1)

Comprobar los permisos del directorio $ HOME del usuario y de / etc/sudoers.d (ls ‐ld) e intentar cambiar a ese directorio.

ls -ld $HOME :

drwxr-xr-x 29 usuario\_local usuario\_local 4096 ene 22 10:52 /home/usuario\_local/

ls -ld /etc/sudoers.d :

drwxr-xr-x 2 root root 4096 abr 22 2015 /etc/sudoers.d/

Se puede cambiar a directorio /etc/sudoers.d/

***Ejercicio 3.*** Los permisos se pueden otorgar de forma selectiva usando la notación octal o la simbólica.

Ejemplo, probar las siguientes órdenes (equivalentes):

● chmod 540 mi\_echo.sh

● chmod u+rx,g+r‐wx,o‐wxr mi\_echo.sh

¿Cómo se podrían fijar los permisos rw‐r‐‐r‐x, de las dos formas?.

chmod 645 ruta de fichero

chmod u+rw-x,g+r-wx,o+rx-w ruta de fichero

Crear un directorio y quitar los permisos de ejecución para usuario, grupo y otros. Intentar cambiar al directorio.

mkdir test

chmod u-x,g-x,o-x test

cd test/

***Ejercicio 4.*** Escribir un programa que, usando la llamada open, cree un fichero con los permisos anteriores. Comprobar el resultado y las características del fichero con la orden ls.

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**#include <stdio.h>**

**int main(int argc, char \*\*argv) {**

**int fd = open(argv[1], O\_CREAT, 0645);**

**return 0;**

**}**

***Ejercicio 5.*** Los permisos por defecto cuando se crea un fichero se derivan de la máscara de usuario (UMASK). El comando umask permite consultar y fijar esta máscara. Usando este comando fijar la máscara de forma que los nuevos ficheros no tengan permiso de escritura para el grupo, y ningún permiso para otros. Comprobar el funcionamiento con los comandos touch y ls.

Mascara por defecto: 0002

0666 & ~000001111(0027) Al invertir la máscara quedará así: 0750

umask 0027

touch prueba

ls -l

**Ejercicio 6.** Modificar el ejercicio 4 para que antes de crear el fichero se fije la máscara igual que en el ejercicio anterior. Una vez creado el fichero debe restaurarse la máscara original del proceso. Comprobar el resultado con el comando ls.

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main(int argc, char \*\*argv) {**

**mode\_t cmask;**

**cmask = umask(0027);**

**int fd;**

**if((fd = open(argv[1], O\_CREAT, 0666)) == -1) {**

**perror("open()");**

**exit -1;**

**}**

**umask(cmask);**

**return 0;**

**}**

***Ejercicio 7.*** El comando ls puede mostrar el inodo con la opción -i. El resto de información del inodo puede obtenerse usando el comando stat. Consultar las opciones del comando y comprobar su funcionamiento.

stat el estado de los ficheros y del sistema de ficheros.

Parámetros:

-L, --dereferente sigue los links

-f, --file-system muestra el estado del sistema de ficheros en vez del estado del fichero

***Ejercicio 8.*** Escribir un programa que emule el comportamiento del comando stat y muestre:

● el número *major* y *minor* asociado al dispositivo,

● el número de inodo del archivo

● el tipo de archivo (directorio, enlace simbólico o archivo ordinario)

● La hora en la que se accedió el fichero por última vez. ¿Qué diferencia hay entre st\_mtime y st\_ctime?.

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <unistd.h>**

**#include <stdlib.h>**

**#include <stdio.h>**

**int main(int argc, char \*\*argv) {**

**struct stat sb;**

**if(stat(argv[1], &sb) == -1) {**

**perror("stat()");**

**return -1;**

**}**

**printf("File type: ");**

**switch (sb.st\_mode & S\_IFMT) {**

**case S\_IFBLK: printf("block device\n"); break;**

**case S\_IFCHR: printf("character device\n"); break;**

**case S\_IFDIR: printf("directory\n"); break;**

**case S\_IFIFO: printf("FIFO/pipe\n"); break;**

**case S\_IFLNK: printf("symlink\n"); break;**

**case S\_IFREG: printf("regular file\n"); break;**

**case S\_IFSOCK: printf("socket\n"); break;**

**default: printf("unknown?\n"); break;**

**}**

**printf("I-node number: %ld\n", (long) sb.st\_ino);**

**printf("Mode: %lo (octal)\n",**

**(unsigned long) sb.st\_mode);**

**printf("Link count: %ld\n", (long) sb.st\_nlink);**

**printf("Ownership: UID=%ld GID=%ld\n",**

**(long) sb.st\_uid, (long) sb.st\_gid);**

**printf("Preferred I/O block size: %ld bytes\n",**

**(long) sb.st\_blksize);**

**printf("File size: %lld bytes\n",**

**(long long) sb.st\_size);**

**printf("Blocks allocated: %lld\n",**

**(long long) sb.st\_blocks);**

**printf("Last status change: %i", ctime(&sb.st\_ctime));**

**printf("Last file access: %i", ctime(&sb.st\_atime));**

**printf("Last file modification: %i", ctime(&sb.st\_mtime));**

**return 0;**

**}**

***Ejercicio 9.*** Los enlaces se crean con la orden ln:

● La opción -s crea un enlace simbólico. Hacer un enlace simbólico a un fichero ordinario y otro a un directorio. Comprobar el resultado con ls -l y ls -i. Determinar el inodo de cada fichero.

mkdir test

touch prueba

ln -s prueba e-simbolico-fich

ln -s test e-simbolico-dir

ls -l

lrwxrwxrwx usuario\_local usuario\_local 4 ene 22 12:22 e-simbolico-dir → test

lrwxrwxrwx usuario\_local usuario\_local 6 ene 22 12:22e-simbolico-fich→prueba

-rw-r----- 1 usuario\_local usuario\_local 0 ene 22 11:47 prueba

drw-r--r-x 2 usuario\_local usuario\_local 4096 ene 22 11:24 test

ls -i

798777 prueba

798770 test

798780 e-simbolico-dir

798779 e-simbolico-fich

● Repetir el apartado anterior con enlaces rígidos. Determinar los inodos de los ficheros y las propiedades con stat (observar el atributo número de enlaces).

ln prueba e-rigido-fich

ln test e-rigido-dir -> ln: <<test>>: no se permiten enlaces rígidos para directorios

ls -li

798773 -rw-r----- 2 usuario\_local usuario\_local 0 ene 22 11:47 e-rigido-fich

798773 -rw-r----- 2 usuario\_local usuario\_local 0 ene 22 11:47 prueba

798780 lrwxrwxrwx 1 usuario\_local usuario\_local 4 ene 22 12:22 e-simbolico-dir -> test

798779 lrwxrwxrwx 1 usuario\_local usuario\_local 6 ene 22 12:22 e-simbolico-fich -> prueba

798770 drw-r--r-x 2 usuario\_local usuario\_local 4096 ene 22 11:24 test

● ¿Qué sucede cuando se borra uno de los enlaces duros?. En el caso de los enlaces simbólicos ¿qué sucede si se borra el enlace? ¿y si se borra el fichero original?

Si se borra uno de los enlaces duros, se decrementa el contador de enlaces, y al compartir el i-nodo, la información no se pierde. En cambio, si se borra el enlace simbólico solo se borra dicho enlace y no se decrementa el contador de enlaces ya que se comporta como un acceso directo. Si se borra el fichero original, el enlace simbólico se rompe y el enlace rigido sigue tenido acceso a la información del fichero.

***Ejercicio 10.*** Las llamadas link y symlink crean enlaces duros y simbólicos respectivamente. Escribir un programa que reciba una ruta a un fichero como argumento. Si la ruta es un fichero regular crear un enlace simbólico y duro (con el mismo nombre terminado en .sym y .hard). Comprobar el resultado con la orden ls.

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <time.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

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

**struct stat sb;**

**char slink[255];**

**char hlink[255];**

**if (argc != 2) {**

**fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);**

**exit(EXIT\_FAILURE);**

**}**

**if (stat(argv[1], &sb) == -1) {**

**perror("stat()");**

**exit(EXIT\_FAILURE);**

**}**

**strcpy(slink, argv[1]);**

**strcat(slink,".sym");**

**strcpy(hlink, argv[1]);**

**strcat(hlink,".hard");**

**switch (sb.st\_mode & S\_IFMT) {**

**case S\_IFREG:**

**if(symlink(argv[1], slink) == -1) {**

**perror("symlink()");**

**exit(EXIT\_FAILURE);**

**}**

**if(link(argv[1], hlink) == -1) {**

**perror("link()");**

**exit(EXIT\_FAILURE);**

**}**

**break;**

**default:**

**printf("The file %s is not a regular file\n",argv[1]);**

**break;**

**}**

**return 0;**

**}**

Apartado 2: Redirecciones y duplicación de descriptores

La shell proporciona operadores (>,>&,>>) que permiten redirigir un fichero a otro, ver los ejercicios

propuestos en la práctica opcional. Esta funcionalidad se implementa mediante las llamadas dup y

dup2.

La llamada de sistema dup() duplica el descriptor de fichero olfd, usando el descriptor con menor número libre para el nuevo descriptor. La llamada de sistema dup2() duplica el descriptor de fichero newfd a oldfd.

***Ejercicio 1.*** Escribir un programa que redirija la salida estándar a un fichero, cuya ruta se pasa como primer argumento. Probar escribiendo varias cadenas en la salida estándar.

**#include <unistd.h>**

**#include <stdio.h>**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**int main(int argc, char \*\*argv) {**

**int fd = open(argv[1], O\_CREAT | O\_WRONLY | O\_TRUNC, 0666);**

**printf("Hola soy Manueh");**

**//dup2(fd,1);**

**close(1);**

**dup(fd);**

**return 0;**

**}**

***Ejercicio 2.*** Modificar el programa anterior para que además de escribir en el fichero la salida estándar también se escriba la salida estándar de error. Comprobar el funcionamiento incluyendo varias sentencias que impriman en ambos flujos. ¿Hay alguna diferencia si las redirecciones se hacen en diferente orden?¿Por qué no es lo mismo “ls > dirlist 2>&1” que “ls 2>&1 > dirlist”

**#include <unistd.h>**

**#include <stdio.h>**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**int main(int argc, char \*\*argv) {**

**int fd = open(argv[1], O\_CREAT | O\_WRONLY | O\_APPEND , 0666);**

**close(1);**

**//dup(fd);**

**close(2);**

**dup2(fd, 1);**

**dup2(fd, 2);**

**printf("Hola soy Manueh\n");**

**if(setuid(0) == -1) {**

**perror("setuid()");**

**}**

**return 0;**

**}**

***Ejercicio 3* (Opcional).** L as funciones dup() y dup2() se mantienen por compatibilidad, ya que toda la funcionalidad es ofrecida por fcntl(). Estudiar qué opciones hay que usar para que fcntl duplique los descriptores.

dup2() es equivalente a fcntl (*old*, F\_DUPFD, *new*)

dup() es equivalente a fcntl(old, F\_DUPFD, 0);

Apartado 3: Cerrojos de ficheros

El sistema de ficheros ofrece un sistema de bloqueos consultivo. Estas funciones se pueden acceder mediante flock o fcntl. En esta sección usaremos únicamente fcntl().

***Ejercicio 1.*** El estado y cerrojos de fichero en uso en el sistema se pueden consultar en el archivo /proc/locks. Estudiar el contenido de este archivo.

Este archivo muestra los archivos bloqueados en la actualidad por el kernel. El contenido de este archivo contiene datos internos de depuración y puede variar enormemente, dependiendo del uso del sistema.

A cada bloqueo se le asigna un único número al inicio de cada línea. La segunda columna se refiere a la clase de bloqueo utilizado; FLOCK, haciendo referencia al estilo antiguo de bloqueos de archivos desde una llamada de sistema flock y POSIX que representa los bloqueos nuevos POSIX desde la llamada de sistema lockf.

La tercera columna puede tener dos valores. ADVISORY O MANDATORY. ADVISORY  significa que el bloqueo no impide que otras personas puedan acceder a los datos; tan sólo previene de que otros intenten establecer un bloqueo. MANDATORY significa que mientras que dura el bloqueo no se permite ningún otro acceso a los datos. La cuarta columna muestra si el bloqueo permite al responsable del mismo acceso de READ o WRITE (lectura y escritura) al archivo. La quinta muestra el ID del proceso que tiene el bloqueo. La sexta columna muestra el ID del archivo bloqueado, en el formato de *MAJOR-DEVICE*:*MINOR-DEVICE*:*INODE-NUMBER*. La séptima y octava columnas muestra el inicio y el final de la región bloqueada del archivo.

***Ejercicio 2.*** Escribir un programa que consulte y muestre en pantalla el estado del cerrojo sobre un fichero. El proceso mostrará el estado del cerrojo (bloqueado, desbloqueado). Además:

● si está desbloqueado, fijará un cerrojo de escritura y escribirá la hora actual. Después

suspenderá su ejecución durante 30 segundos (función sleep) y a continuación liberará el cerrojo.  
● Si el cerrojo está bloqueado terminará el proceso.

**//dup, write**

**#include <unistd.h>**

**//printf, stdout**

**#include <stdio.h>**

**//open, close**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**//errno**

**#include <errno.h>**

**//flock**

**#include <fcntl.h>**

**//strerror**

**#include <string.h>**

**//time**

**#include <time.h>**

**//EXITSUCCESS**

**#include <stdlib.h>**

**int main(int argc, char \*\* argv) {**

**if(argc != 2) {**

**printf("Usage: ./ej2 <fname>\n");**

**return -1;**

**}**

**int fd;**

**fd = open(argv[1], O\_CREAT | O\_RDWR, 0777);**

**if (fd == -1) {**

**/\* Handle error \*/;**

**printf("Error opening file, %i - %s\n", errno, strerror(errno));**

**return -1;**

**}**

**struct flock fl;**

**/\* Make a non-blocking request to place a write lock**

**on bytes 0 to EOF of testfile \*/**

**fl.l\_type = F\_WRLCK; /\* F\_RDLCK, F\_WRLCK, F\_UNLCK \*/**

**fl.l\_whence = SEEK\_SET; /\* SEEK\_SET, SEEK\_CUR, SEEK\_END \*/**

**fl.l\_start = 0; /\* Offset from l\_whence \*/**

**fl.l\_len = 0; /\* length, 0 = to EOF \*/**

**fl.l\_pid = getpid(); /\* our PID \*/**

**if (fcntl(fd, F\_GETLK, &fl) == -1) {**

**/\* Handle unexpected error \*/;**

**printf("Error getting lock, %i - %s\n", errno, strerror(errno));**

**close(fd);**

**return -1;**

**}**

**if(fl.l\_type == F\_WRLCK){**

**printf("Process %ld has a write lock already!\n", fl.l\_pid);**

**printf("%i - %s\n", errno, strerror(errno));**

**close(fd);**

**return -1;**

**} else if (fl.l\_type == F\_RDLCK) {**

**printf("Process %ld has a read lock already!\n", fl.l\_pid);**

**printf("%i - %s\n", errno, strerror(errno));**

**close(fd);**

**return -1;**

**}**

**if (fcntl(fd, F\_SETLK, &fl) == -1) {**

**if (errno == EACCES || errno == EAGAIN) {**

**printf("Already locked by another process\n");**

**/\* We can't get the lock at the moment \*/**

**printf("%i - %s\n", errno, strerror(errno));**

**close(fd);**

**return -1;**

**} else {**

**/\* Handle unexpected error \*/;**

**printf("Unexpected error setting lock, %i - %s\n", errno, strerror(errno));**

**close(fd);**

**return -1;**

**}**

**} else { /\* Lock was granted... \*/**

**/\* Perform I/O on bytes 0 to EOF of file \*/**

**time\_t tim;**

**tim = time(NULL);**

**struct tm \* lt;**

**lt = localtime(&tim);**

**int MAX = 100;**

**const char \* buf[MAX],\* format = "Tiempo %T";**

**int length = strftime(buf, MAX, format, lt);**

**if(length == 0){**

**puts("Couldn't format time");**

**exit(EXIT\_FAILURE);**

**}**

**write(fd, buf, length);**

**/\* Unlock the locked bytes \*/**

**fl.l\_type = F\_UNLCK;**

**fl.l\_whence = SEEK\_SET;**

**fl.l\_start = 0;**

**fl.l\_len = 0;**

**if (fcntl(fd, F\_SETLK, &fl) == -1){**

**/\* Handle error \*/;**

**printf("Error removing lock, %i - %s\n", errno, strerror(errno));**

**close(fd);**

**return -1;**

**}**

**}**

**close(fd);**

**return 0;**

**}**