SISTEMAS OPERATIVOS I - Práctica 3 (con soluciones)
Grado en Ingeniería Informática - Escuela Superior de Informática (UCLM))

1. Actividades de Laboratorio

Escriba un programa C estándar para cada uno de los distintos enunciados con la funcionalidad indicada **utilizando las llamadas al sistema UNIX**. Salvo que se especifique lo contrario, se entenderá que la entrada y salida del programa corresponderá a la entrada estándar y salida estándar y puede realizarse usando la biblioteca estándar C.

1. Construir un programa llamado **micp** que copie el contenido de un archivo de entrada en otro de salida (**micp** <*archivo entrada*><*archivo salida*>). Se aconseja consultar las páginas del manual correspondientes a las llamadas al sistema *open, creat, read, write* y *close*

```c
/* micp.c Copia el contenido de un archivo de entrada en otro de salida. Los
   nombres de los archivos se obtienen de la línea de órdenes */
/* Versión 1 con un buffer de 1 carácter */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  int f1,f2; /* Descriptores de archivo */
  char c; /* Buffer de lectura/escritura */
  int numLeidos; /* Numero de bytes leidos */

  /* Tratamiento de la línea de órdenes */
  if (argc != 3) {
     fprintf(stderr, "Línea de órdenes incorrecta\n");
     return EXIT_FAILURE;
  }

  /* Apertura de archivos */
  if ((f1 = open(argv[1], O_RDONLY)) == -1) {
    fprintf(stderr," Error de apertura en %s\n", argv[1]);
    return EXIT_FAILURE;
  }
  if ((f2 = open(argv[2], O_WRONLY|O_CREAT|O_TRUNC, 0644)) == -1) {
    fprintf(stderr," Error de apertura en %s\n", argv[2]);
    return EXIT_FAILURE;
  }

  /* Bucle de lectura/escritura carácter a carácter */
  while ((numLeidos = read(f1, &c, sizeof(char))) == sizeof(char))
    if (write(f2, &c, sizeof(char)) != sizeof(char)) {
      fprintf(stderr, "Error de escritura\n");
      return EXIT_FAILURE;
    }

  /* Comprobación de error en lectura */
  if (numLeidos == -1) {
    fprintf(stderr, "Error de lectura\n");
    return EXIT_FAILURE;
  }

  close(f1); close(f2); /* Cierre de archivos */
```

```
47    return EXIT_SUCCESS;
}

/* Versión 2 más eficiente pues utiliza un buffer mayor */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

#define MAXBUFFER 4096

int main(int argc, char *argv[])
{
  int f1,f2; /* Descriptores de archivo */
  char buffer[MAXBUFFER]; /* Buffer de lectura/escritura */
  int numLeidos; /* Numero de bytes leidos */

  /* Tratamiento de la línea de ordenes */
  if (argc != 3) {
     fprintf(stderr, "Línea de órdenes incorrecta\n");
     return EXIT_FAILURE;
  }

  /* Apertura de archivos */
  if ((f1 = open(argv[1], O_RDONLY)) == -1) {
    fprintf(stderr," Error de apertura en %s\n", argv[1]);
    return EXIT_FAILURE;
  }
  if ((f2 = open(argv[2], O_WRONLY|O_CREAT|O_TRUNC, 0644)) == -1) {
    fprintf(stderr," Error de apertura en %s\n", argv[2]);
    return EXIT_FAILURE;
  }

  /* Bucle de lectura/escritura con buffer */
  while ((numLeidos = read(f1, buffer, sizeof(buffer))) > 0)
    if (write(f2, buffer, numLeidos) != numLeidos) {
      fprintf(stderr, "Error de escritura\n");
      return EXIT_FAILURE;
    }

  /* Comprobación de error en lectura */
  if (numLeidos == -1) {
    fprintf(stderr, "Error de lectura\n");
    return EXIT_FAILURE;
  }

  close(f1); close(f2); /* Cierre de archivos */
  return EXIT_SUCCESS;
}
```

2. Construir un programa llamado **micp2** que copie el contenido de una zona de una lista de archivos en la salida estándar. La sintaxis de la línea de órdenes sera:

   micp2 -<*desplazamiento archivo*>:<*número bytes*> [<*archivo*>]$^+$

   La zona de los archivos a copiar se indica con un *desplazamiento archivo* desde el inicio del archivo y con el el nº de bytes a copiar (*número bytes*). Se supondrá que los archivos tienen la zona indicada en la línea de órdenes. Se aconseja consultar las páginas del manual correspondientes a las llamadas al sistema *open, read, write, lseek* y *close* y no utilizar las funciones de la biblioteca estándar C para el manejo de la salida estándar

```c
/* micp2.c Copia el contenido de una zona de varios archivos en la salida.
   estándar Los nombres de los archivos se obtienen de la línea de órdenes.
   Se supone que todos los archivos tienen la zona solicitada */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Tamaño del buffer de lectura */
#define MAXBUFFER 4096

/* EscribirArchivo: Escribe en la salida estándar "numBytes" a partir de
   "desp" desde el origen del archivo "nombre" */
void EscribirArchivo(const char *nombre, int desp, int numBytes)
{
  int fd; /* Descriptor del archivo origen */
  char buffer[MAXBUFFER]; /* Buffer de lectura */
  int numLeidos=0; /* Numero de bytes leidos */
  int numLectura; /* Numero de bytes a leer */

  /* Apertura de archivo */
  if ((fd = open(nombre, O_RDONLY)) == -1) {
    fprintf(stderr," Error de apertura en %s\n", nombre);
    exit(EXIT_FAILURE);
  }

  /* Posicionamiento en la zona de lectura */
  if (lseek(fd, desp, SEEK_SET) == -1) {
    fprintf(stderr, "Error de posicionamiento en %s\n", nombre);
    exit(EXIT_FAILURE);
  }

  /* Bucle de lectura/escritura con buffer */
  while (numLeidos < numBytes) {
    /* Cálculo del número de bytes a leer */
    if ((numBytes-numLeidos) >= MAXBUFFER) numLectura = MAXBUFFER;
    else numLectura = numBytes-numLeidos;

    /* Lectura/escritura de la zona del archivo */
    if (read(fd, buffer, numLectura) != numLectura) {
      fprintf(stderr, "Error de lectura en %s\n", nombre);
      exit(EXIT_FAILURE);
    }
    else if (write(1, buffer, numLectura) != numLectura) {
      fprintf(stderr, "Error de escritura\n");
      exit(EXIT_FAILURE);
    }
    numLeidos += numLectura;
  } /* Fin bucle lectura/escritura */

  close(fd); /* Cierre de archivo */
}

int main(int argc, char *argv[])
{
  int desp; /* Desplazamiento desde el origen */
  int bytes; /* Número de bytes a copiar */
  int i;
  char *ptr;
```

```
64    /* Tratamiento de la línea de órdenes */
      if (argc < 3) {
66       fprintf(stderr, "Línea de órdenes incorrecta\n");
         return EXIT_FAILURE;
68    }
      if (argv[1][0] != '-') {
70      fprintf(stderr, "Línea de órdenes incorrecta\n");
        return EXIT_FAILURE;
72    }
      if ((ptr = strchr(argv[1], ':')) == NULL) {
74      fprintf(stderr, "Línea de órdenes incorrecta\n");
        return EXIT_FAILURE;
76    }

78    /* Obtención del desplazamiento y número de bytes */
      desp = atoi(argv[1]+1);
80    bytes = atoi(ptr+1);
      if ((desp < 0) || (bytes < 1)) {
82      fprintf(stderr, "Error en %s\n", argv[1]);
        return EXIT_FAILURE;
84    }

86    /* Bucle sobre la lista de archivos a copiar */
      for (i=2; i<argc; i++) EscribirArchivo(argv[i], desp, bytes);
88
      return EXIT_SUCCESS;
90  }
```

3. Construir un programa llamado **mimv** que cambie el nombre de un nodo del sistema de archivos (**mimv** <*nodo origen*> <*nodo destino o directorio*>). Si el último argumento de la línea de órdenes es un directorio, el nombre del *nodo origen* estará en el *directorio* indicado. Se aconseja consultar las páginas del manual correspondientes a las llamadas al sistema *rename* y *stat*

```
1   /* mimv.c Cambia el nombre de un nodo del sistema de archivos */
    #include <sys/types.h>
3   #include <sys/stat.h>
    #include <unistd.h>
5   #include <stdio.h>
    #include <string.h>
7   #include <stdlib.h>

9   /* Tamaño máximo del camino a un nodo */
    #define PATH_MAX 4096
11
    int main(int argc, char *argv[])
13  {
      char nodo[PATH_MAX+1]; /* Nombre del nuevo nodo */
15    struct stat st;

17    /* Tratamiento de la línea de órdenes */
      if (argc != 3) {
19      fprintf(stderr, "Línea de órdenes incorrecta\n");
        return EXIT_FAILURE;
21    }

23    /* Control de tamaño y valor inicial del nuevo nombre del nodo */
      if ((strlen(argv[2])+1) > sizeof(nodo)) {
25      fprintf(stderr, "Nombre %s demasiado largo\n", argv[2]);
        return EXIT_FAILURE;
27    }
```

```
    strcpy(nodo, argv[2]);

    /* Comprobación de directorio como segundo argumento */
    if (stat(nodo, &st) == 0 ) {
      if (S_ISDIR(st.st_mode)) {
        /* Control de tamaño */
        if ((strlen(argv[2])+strlen(argv[1])+2) > sizeof(nodo)) {
          fprintf(stderr, "Nombre %s/%s demasiado largo\n", argv[2], argv[1]);
          return EXIT_FAILURE;
        }
        sprintf(nodo, "%s/%s", argv[2], argv[1]);
      }
    }

    /* Cambio de nombre del nodo */
    if (rename(argv[1], nodo) == -1) {
      fprintf(stderr,"Error en cambio de nombre\n");
      return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

4. Construir un programa llamado **miln** que cree enlaces de nodos en el sistema de archivos. La sintaxis de la línea de órdenes será:

   a) **miln** [<*opciones*>] <*objetivo*> <*nodo*>

   b) **miln** [<*opciones*>] <*objetivo*>

   c) **miln** [<*opciones*>] [<*objetivo*>]$^+$ <*directorio*>

   En la primera forma se crea un enlace a *objetivo* con el nombre *nodo*. En la segunda forma se crea una enlace a *objetivo* en el directorio actual y en la tercera forma se crea un enlace a cada uno de los objetivos en el directorio indicado. Por defecto, se crean enlaces *físicos o duros* salvo que se utilice la opción *-s* que se crearan enlaces *simbólicos o blandos*. Si es utiliza la opción *-f* y el nombre del enlace que debe crearse ya existe, se deberá borrar el nodo previamente existente y crear luego el enlace. Las opciones se escribirán en el formato habitual de UNIX. Se aconseja consultar las páginas del manual correspondientes a las llamadas al sistema *link, symlink, unlink* y *stat*

```
/* miln.c Crea nuevos enlaces a un nodo */
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

/* Tamaño máximo del camino a un nodo */
#define PATH_MAX 4096

/* LimpiarNombre: Elimina el carácter / final si existe */
void LimpiarNombre(char *nombre)
{
  int pos;

  pos=strlen(nombre)-1; /* Posición al último carácter */
  if (nombre[pos] == '/') nombre[pos]='\0'; /* Eliminar / final */
}

/* Enlazar: Crea enlace duros o blandos eliminando el nodo si es solicitado */
void Enlazar(const char *objetivo, const char *nodo, int soft, int forzar)
{
```

```
24    printf("%s - %s | %d:%d\n", objetivo, nodo, soft, forzar);
      if (forzar) { /* Se elimina el nodo si existe y se solicita */
26      unlink(nodo);
      }
28    if (soft) { /* Creación de enlace simbólico si se solicita */
        if (symlink(objetivo, nodo) == -1) {
30        fprintf(stderr, "Error en creación del enlace %s\n", nodo);
          exit(EXIT_FAILURE);
32      }
      }
34    else { /* Creación de enlace físico por defecto */
        if (link(objetivo, nodo) == -1) {
36        fprintf(stderr, "Error en creación del enlace %s\n", nodo);
          exit(EXIT_FAILURE);
38      }
      }
40  }

42  int main(int argc, char *argv[])
    {
44    char nodo[PATH_MAX+1]; /* Nombre nuevo del nodo */
      char *objetivo; /* Puntero al nombre del objetivo */
46    char *directorio; /* Puntero al nombre del directorio */
      int flag_s=0; /* Opción -s (0=No, 1=Sí) */
48    int flag_f=0; /* Opción -f (0=0, 1=Sí) */
      int c, i;
50    char *ptr;
      struct stat st;
52
      /* Tratamiento de las opciones */
54    while (--argc > 0 && (*++argv)[0] == '-')
        while ((c = *++argv[0]) != '\0')
56        switch (c) {
            case 's':
58            flag_s=1;
              break;
60          case 'f':
              flag_f=1;
62            break;
            default:
64            fprintf(stderr, "Opción ilegal %c\n", c);
              exit(EXIT_FAILURE);
66            break;
          }
68
      /* Tratamiento de la lista de nombres de la línea de órdenes */
70    switch (argc) {
        case 0:
72        fprintf(stderr, "Error en la lista de nombres\n");
          exit(EXIT_FAILURE);
74      case 1:
          objetivo=*argv;
76        LimpiarNombre(objetivo);
          if ((ptr = strrchr(objetivo,'/')) == NULL)
78          strcpy(nodo, objetivo);
          else
80          strcpy(nodo, ptr+1);
          Enlazar(objetivo, nodo, flag_s, flag_f);
82        break;
        case 2:
84        objetivo=*argv;
          argv++; /* Se apunta al segundo nombre */
```

```
86          LimpiarNombre(objetivo);
            LimpiarNombre(*argv);
88
            /* Comprobación de directorio como segundo nombre */
90          if (stat(*argv, &st) == 0 ) { /* El segundo nombre existe */
              if (S_ISDIR(st.st_mode)) { /* El segundo nombre es un directorio */
92              if ((ptr = strrchr(objetivo,'/')) == NULL)
                  sprintf(nodo, "%s/%s", *argv, objetivo);
94              else
                  sprintf(nodo, "%s/%s", *argv, ptr+1);
96            } /* Fin de segundo nombre como directorio */
              else { /* El segundo nombre no es un directorio */
98              strcpy(nodo, *argv);
                if (flag_f == 0) {
100               fprintf(stderr, "El nodo %s ya existe\n", *argv);
                  exit(EXIT_FAILURE);
102             }
              }
104         }
            else /* El segundo nombre es un nodo nuevo */
106           strcpy(nodo, *argv);
            Enlazar(objetivo, nodo, flag_s, flag_f);
108         break;
          default:
110         directorio = argv[argc-1];
            LimpiarNombre(directorio);
112         /* Comprobación del directorio */
            if (stat(directorio, &st) == -1 ) {
114           fprintf(stderr, "Error en nodo %s\n", directorio);
              exit(EXIT_FAILURE);
116         }
            else {
118           if (S_ISDIR(st.st_mode) == 0) {
                fprintf(stderr, "El nodo %s no es un directorio\n", directorio);
120             exit(EXIT_FAILURE);
              }
122         } /* Fin del stat */
            /* Bucle sobre la lista de objetivos */
124         for (i=0; i<argc-1; i++) {
              objetivo=argv[i];
126           LimpiarNombre(objetivo);
              if ((ptr = strrchr(objetivo,'/')) == NULL)
128             sprintf(nodo, "%s/%s", directorio, objetivo);
              else
130             sprintf(nodo, "%s/%s", directorio, ptr+1);
              Enlazar(objetivo, nodo, flag_s, flag_f);
132         }
            break;
134
        }
136     return EXIT_SUCCESS;
    }
```

5. Construir un programa llamado **mils** que imprima por la salida estándar los nombres de los nodos de un directorio (**mils** <*directorio*>). La impresión en la salida estándar consistirá en un nombre por línea. Se aconseja consultar las páginas del manual correspondientes a las llamadas al sistema *opendir, readdir* y *closedir*

```
1 /* mils.c - Imprime en la salida estándar los nombre de los nodos de un
      directorio cuyo nombre se pasa por la línea de órdenes. */
3 #include <sys/types.h>
```

```c
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>

/* ListarDirectorio: Imprime en la salida estándar el nombre los nodos
   almacenados en el directorio "dir" */
void ListarDirectorio(const char *dir)
{
  DIR *d; /* Manejador del directorio */
  struct dirent *nodo;

  /* Apertura del directorio */
  if ((d = opendir(dir)) == NULL) {
    fprintf(stderr, "Error en directorio %s\n", dir);
    exit(EXIT_FAILURE);
  }

  /* Bucle de lectura de nombres en el directorio */
  while ((nodo = readdir(d)) != NULL)
    printf("%s\n", nodo->d_name);

  closedir(d);
}

int main(int argc, char *argv[])
{
  /* Tratamiento de la línea de órdenes */
  if (argc != 2) {
    fprintf(stderr, "Error en la línea de órdenes\n");
    return EXIT_FAILURE;
  }
  ListarDirectorio(argv[1]);
  return EXIT_SUCCESS;
}
```

6. Construir un programa **mitree** que imprima los nombres de los nodos de un directorio cuyo nombre se pasa por la línea de órdenes. El programa debe admitir la presencia en la línea de órdenes, de una opción -R que, cuando esté presente indique que el listado del directorio debe ser recursivo (**mitree** [<-R>] <*directorio*>). Se supondrá que en el directorio (y en sus descendientes) no hay enlaces simbólicos. La impresión en la salida estándar consistirá en un nombre por línea. Los nombres deberán ser nombres relativos al directorio que se pasa como argumento en la línea de órdenes. Se aconseja consultar las páginas del manual correspondientes a las llamadas al sistema *opendir*, *readdir*, *stat* y *closedir*

```c
/* mitree.c - Imprime en la salida estándar los nombre de los nodos de un directorio
   cuyo nombre se pasa por la línea de órdenes. Si se utiliza la opción -R el
   listado del directorio es recursivo. Se supone que no existen enlaces
   simbólicos */
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/* Tamaño máximo del camino a un nodo */
#define PATH_MAX 4096

/* ListarDirectorio: Imprime en la salida estándar el nombre de los nodos del
```

```
17    directorio "dir" según las opciones de "flag" */
   void ListarDirectorio(const char *dir, int flag)
19 {
     DIR *d; /* Manejador del directorio */
21   struct dirent *nodo;
     struct stat st;
23   char nombre[PATH_MAX+1]; /* Directorio + nombre del nodo en el directorio */

25   /* Apertura del directorio */
     if ((d = opendir(dir)) == NULL) {
27     fprintf(stderr, "Error en directorio %s\n", dir);
       return;
29   }

31   /* Bucle de lectura de nombres en el directorio */
     while ((nodo = readdir(d)) != NULL) {
33     /* Control de tamaño */
       if ((strlen(dir)+strlen(nodo->d_name)+2) > sizeof(nombre)) {
35       fprintf(stderr, "Nombre %s/%s demasiado largo\n", dir, nodo->d_name);
         exit(EXIT_FAILURE);
37     }
       sprintf(nombre, "%s/%s", dir, nodo->d_name);
39     printf("%s\n", nombre);
       if (flag) { /* Si listado recursivo */
41       /* Se saltan los directorios actual y padre */
         if (strcmp(nodo->d_name, ".") && strcmp(nodo->d_name, "..")) {
43         if (stat(nombre, &st) == -1) {
             fprintf(stderr, "Error en %s\n", nodo->d_name);
45           exit(EXIT_FAILURE);
           }
47         if (S_ISDIR(st.st_mode)) ListarDirectorio(nombre, flag);
         }
49     }
     }
51   closedir(d);
   }
53
   int main(int argc, char *argv[])
55 {
     int flagR=0; /* Presencia del flag R */
57
     /* Tratamiento de la línea de órdenes */
59   while (--argc > 0 && (*++argv)[0] == '-') {
       switch ((*argv)[1]) {
61       case 'R': flagR = 1;
                  break;
63       default : fprintf(stderr, "Error en las opciones\n");
                  exit(EXIT_FAILURE);
65                break;
       }
67   }

69   ListarDirectorio(*argv, flagR);
     return EXIT_SUCCESS;
71 }
```

7. Construir un programa llamado **mayor** que imprima los nombres de los archivos regulares de un directorio, y de sus subdirectorios recursivamente, y los tamaños de los nodos que sean archivos cuyo tamaño sea mayor que uno dado (**mayor** *<número><directorio>*). La salida consistirá en líneas con los nombres de los archivos y los respectivos tamaños (un archivo por línea). Los nombres deberán ser

nombres relativos al directorio que se pasa como argumento en la línea de órdenes. Se aconseja consultar las páginas del manual correspondientes a las llamadas al sistema *opendir, readdir, stat* y *closedir*

```c
/* mayor.c — Imprime los nombres de los nodos de un directorio de forma
   recursiva que son mayores de un tamaño dado en la línea de órdenes */
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Tamaño máximo del camino a un nodo */
#define PATH_MAX 4096

/* BuscarArchivos: Imprime en la salida estándar el nombre de los archivos
   regulares del directorio "dir" o sus subdirectorios que tengan un tamaño
   mayor a "tamano" */
void BuscarArchivos(const char *dir, long int tamano)
{
  DIR *d; /* Manejador del directorio */
  struct dirent *item;
  char nombre[PATH_MAX+1]; /* Directorio + nombre del nodo en el directorio */
  struct stat st;

  /* Apertura del directorio */
  if ((d = opendir(dir)) == NULL) {
    fprintf(stderr, "Error en el directorio %s\n", dir);
    return;
  }
  /* Bucle de lectura de nodos en el directorio */
  while ((item = readdir(d)) != NULL) {
    /* Se omite el directorio actual y el directorio padre */
    if (strcmp(item->d_name, ".") == 0 || strcmp(item->d_name, "..") == 0)
      continue;
    /* Control de tamaño */
    if ((strlen(dir)+strlen(item->d_name)+2) > sizeof(nombre)) {
      fprintf(stderr, "Nombre %s/%s demasiado largo\n", dir, item->d_name);
      exit(EXIT_FAILURE);
    }
    sprintf(nombre, "%s/%s", dir, item->d_name);
    if (stat(nombre, &st) == -1) {
      fprintf(stderr, "Error en %s\n", nombre);
      exit(EXIT_FAILURE);
    }
    if (S_ISREG(st.st_mode) && st.st_size > tamano)
      printf("%s\t%ld\n", nombre, st.st_size);
    else if (S_ISDIR(st.st_mode)) BuscarArchivos(nombre, tamano);
  }
  closedir(d);
}

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

  /* Tratamiento de la línea de órdenes */
  if (argc != 3) {
    fprintf(stderr, "Error en la línea de órdenes\n");
    return EXIT_FAILURE;
  }
```

```
    tamano = atol(argv[1]);
61  BuscarArchivos(argv[2], tamano);
    return EXIT_SUCCESS;
63  }
```

8. Construir un programa llamado **michmod** que cambie los permisos de un nodo del sistema de archivos. La sintaxis del programa será **michmod** <*permisos en octal*><*nodo*>. Se aconseja consultar las páginas del manual correspondientes a la llamada al sistema *chmod* y a la función de la biblioteca estándar *sscanf*

```
1  /* michmod.c Cambia los permisos de un nodo */
   #include <sys/stat.h>
3  #include <stdio.h>
   #include <stdlib.h>
5
   int main(int argc, char *argv[])
7  {
     unsigned int permisos; /* Permisos en formato numérico */
9
     /* Tratamiento de la línea de órdenes */
11   if (argc != 3) {
       fprintf(stderr, "Línea de órdenes incorrecta\n");
13     return EXIT_FAILURE;
     }
15
     /* Obtención de los permisos en octal */
17   if (sscanf(argv[1], "%o", &permisos) != 1) {
       fprintf(stderr, "No se puede obtener los permisos de %s\n", argv[1]);
19     return EXIT_FAILURE;
     }
21
     if (chmod(argv[2], permisos) == -1) {
23     fprintf(stderr, "No se puede cambiar los permisos a %s\n", argv[2]);
       return EXIT_FAILURE;
25   }

27   return EXIT_SUCCESS;
   }
```

9. Construir un programa llamado **rmall** que elimine totalmente un archivo regular de un directorio incluyendo sus subdirectorios, es decir, se eliminen todos los enlaces físicos del archivo en el directorio indicado y en sus subdirectorios. La sintaxis del programa será **rmall** <*archivo regular*><*directorio*>). Se aconseja consultar las páginas del manual correspondientes a las llamadas al sistema *opendir, readdir, closedir, unlink* y *stat*

```
   /* rmall.c - Elimina de forma definitiva un archivo regular */
2  #include <sys/types.h>
   #include <sys/stat.h>
4  #include <unistd.h>
   #include <dirent.h>
6  #include <stdlib.h>
   #include <stdio.h>
8  #include <string.h>

10 /* Tamaño máximo del camino a un nodo */
   #define PATH_MAX 4096
12
   /* EliminarArchivo: Elimina el nº de enlaces "nlink" al "dev+ino" indicado
14    a partir del directorio "dir" */
   void EliminarArchivo(dev_t device, ino_t inodo, nlink_t nlink, const char *dir)
16 {
```

```
    DIR *d; /* Manejador del directorio */
18  struct dirent *nodo;
    struct stat st;
20  char nombre[PATH_MAX+1]; /* Directorio + nombre del nodo en el directorio */

22  /* Apertura del directorio */
    if ((d = opendir(dir)) == NULL) return;
24
    /* Bucle de lectura de nombres en el directorio */
26  while ((nlink > 0) && (nodo = readdir(d)) != NULL) {
      /* Control de tamaño */
28    if ((strlen(dir)+strlen(nodo->d_name)+2) > sizeof(nombre)) {
        fprintf(stderr, "Nombre %s/%s demasiado largo\n", dir, nodo->d_name);
30      exit(EXIT_FAILURE);
      }
32    sprintf(nombre, "%s/%s", dir, nodo->d_name);
      /* Se saltan los directorios actual y padre */
34    if (strcmp(nodo->d_name, ".") && strcmp(nodo->d_name, "..")) {
        /* Obtención de atributos del nodo */
36      if (stat(nombre, &st) == -1) continue;

38      /* Recorrido recursivo si es un directorio */
        if (S_ISDIR(st.st_mode)) EliminarArchivo(device, inodo, nlink, nombre);
40
        /* Eliminación del enlace si procede */
42      if ((device == st.st_dev) && (inodo == st.st_ino)) {
          if (unlink(nombre) == -1) {
44          fprintf(stderr, "Error al eliminar el enlace %s\n", nombre);
            exit(EXIT_FAILURE);
46        }
          nlink--; /* Se decrementa el nº de enlaces */
48      }
      }
50  }
    closedir(d);
52 }

54 int main(int argc, char *argv[])
   {
56   struct stat st;

58   /* Tratamiento de la línea de órdenes */
     if (argc != 3) {
60     fprintf(stderr, "Error en la línea de órdenes\n");
       return EXIT_FAILURE;
62   }

64   /* Obtención de atributos del nodo */
     if (stat(argv[1], &st) != 0) {
66     fprintf(stderr, "Error en %s\n", argv[1]);
       return EXIT_FAILURE;
68   }
     /* Comprobación de archivo regular */
70   if (S_ISREG(st.st_mode) == 0) {
       fprintf(stderr, "El nodo %s no es un archivo regular\n", argv[1]);
72     return EXIT_FAILURE;
     }
74   /* Eliminación de todos los enlaces a ese archivo */
     EliminarArchivo(st.st_dev, st.st_ino, st.st_nlink, argv[2]);
76
     return EXIT_SUCCESS;
78 }
```