```c
/*
 * Archivo: create.c
 *
 * descripcion: Archivo fuente con las funciones necesarios para la creacion
 * de un archivo .mytar
 *
 * Autores:
 *  Carlos Alejandro Sivira Munoz     15-11377
 *  Cesar Alfonso Rosario Escobar   15-11295
 */


#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <string.h>
#include <unistd.h>
#include <string.h>
#include "create.h"
#include "encryption.h"
#include "parser.h"


/* setHeadFields
 * ----------
 * Asigna los campos de cabecera del archivo .mytar utilizando los atributos
 * de los archivos que se recibieron para empaquetar.
 *
 * Estos son:
 *    modo # uid # gid [ # size] # name_size # name [# link_pointer] #
 *
 *
 *  fd_dest: "file descriptor" del archivo .mytar
 *  state: Estado del archivo actual.
 *  name: Nombre del archivo a empaquetar.
 */
void setHeadFields(int fd_dest, struct stat state, char *name) {
  int len, test, file_type, field_size;
  mode_t mode;
  uid_t uid;
  gid_t gid;
  long size;
  char* pointer;

  mode = state.st_mode;
  uid = state.st_uid;
  gid = state.st_gid;
  size = state.st_size;

  file_type = mode & __S_IFMT;

  /*  Anado el modo del archivo   */
  dprintf(fd_dest,"%d%c", mode ,STUFF_TOKEN);

  /*  Anado el uid del archivo  */
  dprintf(fd_dest,"%d%c", uid, STUFF_TOKEN);

```

```
61    /*  Anado el gid del archivo  */
62    dprintf(fd_dest,"%d%c", gid, STUFF_TOKEN);
63
64    if( file_type != __S_IFDIR )  {
65      /*  Anado el tamano del archivo  */
66      dprintf(fd_dest,"%ld%c", size, STUFF_TOKEN);
67    }
68
69    /*  anado el tamano del nombre */
70    field_size = strlen(name);
71    dprintf(fd_dest,"%d%c", field_size, STUFF_TOKEN);
72
73    /*  Anado el nombre del archivo   */
74    dprintf(fd_dest ,"%s%c", name, STUFF_TOKEN);
75
76    if ( file_type  == __S_IFLNK) {
77      pointer = (char*) malloc(size + 1);
78      readlink(name, pointer, size);
79      pointer[size] = '\0';
80
81      /* Anado el path apuntado por el link */
82      dprintf(fd_dest,"%s%c", pointer, STUFF_TOKEN);
83      free(pointer);
84    }
85
86  }
87
88
89  /* fileWriter
90   * ----------
91   * Escribe de un archivo a otro utilizando sus "file descriptors"
92   *
93   *
94   *  fd_source: "file descriptor" del archivo del que se lee
95   *  fd_dest: "file descriptor" del archivo al que se escribe
96   *    instructions: Estructura que contiene la informacion de las opciones
97   *    de mytar.
98   */
99  void fileWriter(int fd_source, int fd_dest, mytar_instructions inst) {
100
101   char *temp_buffer = (char*) malloc(MAX_RW*sizeof(char)+1);
102   char *buffer = (char*) malloc(MAX_RW*sizeof(char)+1);
103   int read_length, to_write;
104   struct stat st_dest;
105
106   read_length = 1;
107
108   while( (read_length = read(fd_source, buffer, read_length)) != 0) {
109
110     /* modifica el string a escribir si se va a encriptar*/
111     if (inst.mytar_options[Z]) {
112       temp_buffer = (char*) encrypt(buffer, inst.encryption_offset);
113       strncpy(buffer,temp_buffer,MAX_RW);
114       free(temp_buffer);
115     }
116
117     to_write = 0;
118     while(read_length > to_write)
119       to_write += write(fd_dest,buffer+to_write,read_length-to_write);
120   }
```

```
121
122   free(buffer);
123 }
124
125
126 /* handleFileType
127  * --------------
128  *  Esta funcion recibe un 'struct stat' que le permite determinar el tipo
129  *  de archivo, y asignar campos de cabecera de .mytar en funcion de ello.
130  *
131  *
132  *    fd_dest: File descriptor de archivo .mytar.
133  *    pathname: nombre del archivo que se esta procesando.
134  *    current_st: Estado del archivo.
135  *  instructions: Estructura que contiene la informacion de las opciones de
136  *     mytar.
137  *
138  * Retorna NULL, o DIR* en caso de que el archivo procesado sea un directorio.
139  * Esto porque si es un directorio, debo procesar sus campos de cabecera y
140  * luego recorrerlo recursivamente y para esto necesito devolver un apuntador
141  * al directorio abierto a la funcion que llama.
142  *
143  */
144 DIR *handleFileType(int fd_dest, char* pathname, struct stat current_st,
    mytar_instructions inst) {
145
146   DIR *ith_pointer;
147   int current_fd_dest;
148   char *pointer;
149
150   /* El archivo es un directorio */
151   if ( (current_st.st_mode & __S_IFMT) == __S_IFDIR ) {
152
153     ith_pointer = opendir(pathname);
154     if ( ith_pointer == NULL ) {
155       fprintf(stderr,"No se pudo abrir %s\n", pathname);
156       perror("opendir");
157       return NULL;
158     }
159
160     /*Verifica si el modo verboso esta activo*/
161     if (inst.mytar_options[V]){
162       verboseMode(inst, pathname);
163     }
164
165     setHeadFields(fd_dest, current_st, pathname);
166
167     return ith_pointer;
168   }
169
170   /* El archivo es regular */
171   else if ( (current_st.st_mode & __S_IFMT) == __S_IFREG ) {
172
173     current_fd_dest = open(pathname, O_RDONLY);
174     if(current_fd_dest == -1)  {
175       fprintf(stderr,"No se pudo abrir %s\n",pathname);
176       perror("open");
177       return NULL;
178     }
179
```

```
180     if (inst.mytar_options[V]){
181       verboseMode(inst, pathname);
182     }
183 
184     setHeadFields(fd_dest, current_st, pathname);
185     fileWriter(current_fd_dest, fd_dest, inst);
186     close(current_fd_dest);
187   }
188 
189   /* El archivo es un link simbolico */
190   else if ( (current_st.st_mode & __S_IFMT) == __S_IFLNK ) {
191 
192     /*Verifica si es necesario ignorar este archivo*/
193     if (!inst.mytar_options[N]){
194       setHeadFields(fd_dest, current_st, pathname);
195 
196     }
197   }
198 
199   return NULL;
200 }
201 
202 
203 /* traverseDir
204  * ----------
205  * Esta funcion recorre un arbol de directorios anadiendo campos de cabecera al
    archivo
206  * .mytar que se esta procesando, junto con el contenido de los archivos procesados.
207  *
208  * Consigue esto procesando los atributos de cada uno de los archivos encontrados
209  * como entrada de directorio, y anadiendolos ar archivo .mytar
210  *
211  *
212  *  dir: apuntador al directorio que se esta recorriendo
213  *  dirname: nombre del directorio que se esta recorriendo
214  *  fd: file descriptor del archivo .mytar que se esta creando
215  *    instructions: Estructura que contiene la informacion de las opciones de
216  *     mytar.
217  */
218 void traverseDir(DIR *dir, char *dirname, int fd, mytar_instructions inst) {
219 
220   int len;
221   char path[MAX_PATHNAME], pathname[MAX_PATHNAME];
222   DIR *is_dir;
223   struct dirent *current_ent;
224   struct stat current_st;
225 
226   strcpy(path, dirname);
227 
228   while( (current_ent=readdir(dir)) != NULL ) {
229 
230     if( strcmp(current_ent->d_name,".")!=0 && strcmp(current_ent->d_name,"..")!=0)
231     {
232       strcpy(pathname, path);
233 
234       len = strlen( pathname );
235 
236       if (pathname[len-1] != '/')
237         strcat(pathname,"/");
238 
```

```
      /* Extiendo el pathname para que incluya el nombre de la
       * entrada actual */
      strcat(pathname, current_ent->d_name);


      if(lstat(pathname, &current_st) == -1)
        perror("stat");

      /* Verifica que la entrada revisada sea un directorio
       * Si lo es la recorre
       * de lo contrario lee la siguiente entrada
       */
      is_dir = handleFileType(fd, pathname, current_st, inst);
      if (is_dir != NULL) {
        traverseDir(is_dir, pathname, fd, inst);
        closedir(is_dir);
      }

    }
  }

}


/* createMyTar
 * --------------
 * Se utiliza para crear el archivo .mytar. Funciona procesando cada
 * archivo recibido en la linea de comandos para obtener atributos que usar
 * como campos de cabecera para el archivo .mytar.
 *
 * Procesa cada argumento recibido, primero verificando que existe y luego:
 *  Si es un directorio:
 *    lo abre, lo recorre (asignando los respectivos campos) y lo cierra.
 *  En caso contrario:
 *    asigna los campos de cabecera y el contenido del archivo recibido.
 *
 *  files: Archivos a procesar
 *  n_files: Numero de archivos a procesar
 *    instructions: Estructura que contiene la informacion de las opciones de
 *      mytar.
 */
int createMyTar(int n_files, char **files, mytar_instructions inst) {

  int fd, current_fd, i;
  char *local_path = (char*) malloc(MAX_PATHNAME);
  DIR *dir, *current_dir;
  struct stat current_st;


  fd = open(files[0], CREATE_APPEND_MODE, MY_PERM);
  if (fd == -1) {
    fprintf(stderr,"Error creando archivo .mytar\n");
    perror("open\n");

    return -1;
  }

  for(i=1; i<n_files; i++) {

    if( stat(files[i], &current_st) != 0) {
```

```
      perror("stat");
      continue;
    }

    strcat(local_path, files[i]);

    current_dir = handleFileType(fd, local_path, current_st, inst) ;

    if (current_dir != NULL) {
      traverseDir(current_dir, files[i], fd, inst);
      closedir(current_dir);
    }

    strcpy(local_path, "");
  }


  free(local_path);
  close(fd);

  return 0;
}
```