Problema 2.33 (marzo 2014)

En un sistema Linux se desea disponer de un mandato que cambie el dueño y los permisos de los elementos contenidos en un subdirectorio dado. Por ejemplo, para que el jefe pueda acceder a los ficheros que tiene en su directorio home un empleado que deja el trabajo.

El mandato deberá estar basado en la siguiente función:

```
int CambiaSubDir(char *DirectRecorrer, uid_t owner, gid_t group);
```

Esta función ha de recorrer el subdirectorio DirectRecorrer y procesar cada elemento encontrado cambiando su dueño a owner y group y cambiando sus permisos de acuerdo a las siguientes reglas:

- *Si es un enlace simbólico no deberá ser procesado.*
- *Si es un fichero sin permiso de ejecución para su propietario, le pondrá permisos 0600.*
- *Si es un fichero con permiso de ejecución para su propietario, le dejará los permisos que tiene, incluidos los posibles bits SETUID y SETGID.*
- *Si es un directorio le pondrá permisos 0700.*
- *En cualquier otro caso, dejará los permisos que tenga.*

Nota: Para determinar si un fichero tiene activo un bit de permiso basta con hacer un AND con un máscara que tenga todos ceros menos un 1 en la posición del bit a analizar. Por ejemplo, (mode & 00001) será cierto si el fichero tiene permiso de ejecución para el mundo, (mode & 00002) si tiene permiso de escritura para el mundo, etc. y (mode & 04000) si tiene activo el bit SETUID.

a) Indicar justificadamente la identidad efectiva que deberá tener dicho proceso para poder ejecutar con éxito.

b) Indicar justificadamente en qué orden deben hacerse los siguientes pasos: cambiar el dueño, determinar los permisos, determinar el tipo de fichero y cambiar los permisos.

c) Escribir la función CambiaSubDir.

d) Modificar la función para que se llame recursivamente en cada directorio encontrado.

Solución

a) Para poder cambiar el dueño y los permisos de un fichero hace falta ser el dueño o root. Se puede pensar, en un principio, que sería suficiente con la identidad del empleado, puesto que se trata de sus ficheros. Sin embargo, al cambiar el dueño se pierden los bits de SETUID y SETGID, por lo que el dueño original del fichero si quisiese restituirlos no podría porque el fichero ya no sería suyo. Por lo tanto, solamente un proceso con identidad efectiva de root podría ejecutar dichas llamadas.

b) Lo primero es ver el tipo de fichero, puesto que si es un enlace simbólico no hay que procesarlo.

Seguidamente, hay que obtener los permisos, puesto que si cambiamos el dueño antes de obtener los permisos perderíamos los bits de SETUID y SETGID.

Luego se cambia el dueño y grupo y, finalmente se cambian los permisos, incluyendo, en su caso, los bits de SETUID y SETGID.

c) Un aspecto importante a tener en cuenta en el diseño del programa es que los nombres que se obtienen en la estructura de tipo `dirent` es el nombre local dentro del directorio. Si queremos utilizar ese nombre, en servicios que requieran el nombre como `chown` y `chmod`, deberemos concatenarlos al nombre del directorio.

En caso de fallo en alguno de los servicios del sistema operativo la función termina y devuelve un valor que indi-

```
#define MAX_BUF   300
int CambiaSubDir(char *DirectRecorrer, uid_t owner, gid_t group) {
   DIR *dirp;
   struct dirent *dp;
   struct stat atributos
   char fichero[MAX_BUF];
   mode_t mascaraset;
   //Se abre el directorio DirectRecorrer. Puede ser dirección absoluta o relativa
   if (dirp = opendir(DirectRecorrer) == NULL) return 1;
   //Se lee entrada a entrada del directorio empezando por el "." y el ".."
   while ((dp = readdir(dirp)) != NULL) {  //El readdir obtiene el nombre local
      if (strcmp(dp->d_name, "..") == 0) continue; //El directorio padre no hay que tocarlo
      strcpy (fichero, DirectRecorrer);     //Hay que concatenar el nombre local a DirectRecorrer
      strcat (fichero, "/");                //para tener el nombre completo
      strcat (fichero, dp->d_name);
      if (lstat(fichero, &atributos) < 0) return 2;
      if (S_ISLNK(atributos.st_mode)) continue;  //Caso de enlace simbólico
      if (S_ISDIR(atributos.st_mode)) {          //Caso de directorio
         if (chown(fichero, owner, group) < 0) return 3;
         if (chmod(fichero, 00700) < 0) return 4;
         continue;
      }
      mascaraset = atributos.st_mode & 06000;     //Salvamos los bits de SETUID y SETGID
      if (chown(fichero, owner, group) < 0) return 5;
      if (S_ISREG(atributos.st_mode)) {          //Caso de fichero de usuario
         if (atributos.st_mode & 0100)           //Bit de ejecución del dueño
            if (chmod(fichero, atributos.st_mode | mascaraset) < 0) return 6;
         else
            if (chmod(fichero, 00600) < 0) return 7;
         continue;
      }
      if (chmod(fichero, atributos.st_mode | mascaraset) < 0) return 8;
   }
   closedir(dirp);
   return 0;
}
```

d) Lo primero que hay que tener en cuenta es que si no se elimina el caso “..” de la función, al hacerla recursiva cambiaría todo el árbol directorio y, además, entraría en un bucle infinito. Si no se elimina el caso “.” el programa entraría también en un bucle infinito.

Hay que tocar el if del caso directorio cambiándolo, por ejemplo, por el siguiente código:

```
      if (S_ISDIR(atributos.st_mode)) {    //Caso de directorio
         if (chown(fichero, owner, group) < 0) return 2;
         if (chmod(fichero, 00700) < 0) return 3;
//Evitar bucle infinito
         if (strcmp((dp->d_name, ".") != 0) CambiaSubDir(fichero, owner, group);
         continue;
         }
```

Con la solución propuesta los directorios, menos el original, se tratarían dos veces, una al tratar su nombre y otra al tratar el "." .