Práctica 2: gestión de opciones

Introducción

Objetivos

- Gestión de opciones línea de comandos
- Revisión llamadas al sistema para manejo de ficheros y directorios

Concepto básicos

Opciones a comandos en Unix

Las utilidades o comandos de *Unix* usan habitualmente una sintaxis como esta:

```
comando [opciones] [argumentos...]
```

La convención habitual en *Unix*, recogida en el propio estándar POSIX (Utility Argument Syntax) es que las opciones deben venir precedidas por un guión (`-`) seguido de un único carácter (preferiblemente en minúsculas) que identifica la opción (por ejemplo: `comando -f`). Si la opción necesita un argumento adicional, dicho argumento se indica a continuación, habitualmente separado por un espacio en blanco (por ejemplo: `comando -f mifichero`).

Es posible agrupar opciones después de un único guión, aunque en ese caso la opción que puede recibir un argumento adicional es la que aparece en último lugar (por ejemplo: `comando -laf mifichero` es equivalente a `comando -l -a -f mifichero`).

Las utilidades de GNU permiten además opciones *largas*, palabras completas precedidas por dos guiones (`--`), para hacer que los argumentos de línea de comando sean más legibles y comprensibles (por ejemplo: `comando --file mifichero`).

Análisis de argumentos con `getopt`

Para facilitar el análisis de argumentos siguiendo esta convención tradicional de Unix, la librería de estándar de C incorporó la función`getopt` y fue añadida posteriormente al estándar POSIX. La implementación de `getopt` en Linux sigue las convecciones del estándar.

```
#include <unistd.h>
int getopt(int argc, char *argv[], const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

`getopt` recibe:

- Un array de argumentos: el array `argv` con `argc` entradas, que habitualmente se corresponde con los argumentos `argv` y `argc` de la función `main`.
- Una cadena `optstring` que sirve para especificar las opciones válidas. Cada carácter en dicha cadena es una opción valida. Por ejemplo, la cadena `"ab"` indica que las opciones válidas son `-a` y `-b`. El estándar especifica

que son opciones validas cualquier carácter incluido en el conjunto [a-zA-Z0-9], aunque las implementaciones suelen permitir caracteres adicionales. No pueden utilizarse como opción los caracteres `:`,`?` y `-` porque tienen significado especial para `getopt`. Para indicar que una opción espera un argumento adicional, el carácter de la opción está seguido por el carácter `:` en la cadena `optstring`. Por ejemplo, la cadena `"a:b"` indica que las opciones válidas son `-a` y `-b` y que la opción `-a` espera un argumento adicional.

Para procesar todas las opciones que recibe nuestro programa es necesario llamar a `getopt` varias veces (el uso habitual es utilizar un bucle `while`). `getopt` analiza los argumentos, buscando las opciones especificados por `optstring` que aún no han sido procesadas:

- Si `getopt` encuentra una opción válida, devuelve el carácter de dicha opción (como tipo int) y actualiza la variable global `int optind` para que apunte al siguiente argumento de `argv` sin analizar (`optind` se inicializa a 1). Si la opción tenia argumento, la variable global `char *optarg` apunta a dicho argumento.
- Si `getopt` ha encontrado una opción que no es valida, devuelve el carácter `?` para indicar que se ha producido un error y la variable global `optopt` indica en ese caso la opción no reconocida. Por defecto se muestra también en `stderr` que se ha producido un error. Si se quiere evitar el mensaje de error se debe poner a cero la variable global `opterr`, que esta inicializada a 1.
- Si la cadena `optstring` comienza por el carácter `:` y `getopt` ha encontrado una opción válida pero que debería haber recibido un argumento y no se ha indicado el argumento en la línea de comandos, `getopt` devuelve el carácter `:` para indicar el error.
- Si `getopt` ha llegado al final de la cadena de opciones, devuelve un -1. En ese caso, si `optind` es menor que `argc`, `optind` apunta al primer elemento de `argv` que no es una opción (es decir, el primer argumento que no es opción es `argv[optind]`).
- Si la cadena `optstring` comienza por el carácter `:` y `getopt` ha encontrado una opción válida pero que debería haber recibido un argumento y no se ha especificado, `getopt` devuelve el carácter `:` para indicar el error.

El siguiente ejemplo ilustra su uso en un ejemplo en el que se definen dos opciones: `-h` (ayuda) y `-v valor`. La función `getopt` analiza cada argumento de la línea de comandos y devuelve un valor que indica la opción que se ha encontrado. Si se encuentra la opción `-h`, se muestra un mensaje de ayuda y se termina la aplicación. Si se encuentra la opción `-v`, se lee el valor del argumento y se imprime por la salida estándar:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

enum opciones {NOOP,OPCION_H,OPCION_V};

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

  int opcion;
  int valor;
  int flag = NOOP;

  while ((opcion = getopt(argc, argv, ":hv:")) != -1) {
    switch (opcion) {
      case 'h':
        flag=OPCION_H;
        printf("Uso: %s [-h] [-v valor]\n",argv[0]);
        printf("   -h          : imprime esta ayuda\n");
        printf("   -v valor    : valor de entrada\n");
        exit(0);
      case 'v':
        valor = atoi(optarg);
        flag=OPCION_V;
```

```
        break;
      case ':':
        fprintf(stderr, "no se ha especificado valor de entrada para -v \n");
        exit(1);
      case '?':
        fprintf(stderr, "Opción %c no válida\n", (unsigned char) optopt);
        exit(1);
      default:
        fprintf(stderr, "opcion case no esperada\n");
        exit(1);
    }
  }
  if (flag == OPCION_V) {
    printf("Opcion v, valor: %d\n", valor);
  }
  if (optind < argc ){
     printf("El primer argumento que no es opcion es %s\n",argv[optind]);
  } else {
     printf("No hay mas argumentos\n");
  }
  return 0;
}
```

Extensiones de GNU y `getopt_long`

La implementación de `getopt` de la `libc` de GNU puede permutar los argumentos del array `argv` al analizar las opciones, dejando siempre los argumentos que no son opciones al final. Este comportamiento se aparta del estándar POSIX, que no permite el reordenamiento. Se puede forzar el comportamiento POSIX fijando la variable de entorno `POSIXLY_CORRECT` a cualquier valor o comenzando la cadena `optstring` con el carácter +.

Para analizar opciones *largas* de GNU se utiliza la función `getopt_long`. Su comportamiento es similar a `getopt` pero añade además una estructura *option* para definir las opciones largas:

```
#include <getopt.h>

int getopt_long(int argc, char * const argv[], const char *optstring,
                const struct option *longopts, int *longindex);
```

La estructura option se define como:

```
struct option {
    const char *name; // Nombre de la opción larga
    int has_arg;      // (no_argument, required_argument, optional_argument)
    int *flag;        //
    int val;          //
};
```

El campo `name` es el nombre de la opción larga y el campo `has_arg` permite identificar si la opción recibe o no un argumento adicional. Si el campo `flag` es NULL, en caso de encontrarse la correspondiente opción larga `getopt_long` devuelve el valor especificado en el campo `val`, siendo frecuente en este caso devolver el carácter usado en la opción corta equivalente. Si `flag` no es NULL, `getopt_long` devuelve 0, y la variable a la que apunta `flag` toma el valor especificado en el campo `val`.

El siguiente ejemplo extiende el programa que sirvió para ilustrar la función `getopt`, añadiendo las opciones largas `--help` (equivalente a la opción corta `-h`) y `value` (equivalente a la opción corta `-v`):

```
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
```

```c
enum opciones {NOOP, OPCION_H, OPCION_V};

int main(int argc, char *argv[]) {
    int opcion;
    int valor;
    int flag = NOOP;

    // Definición de las opciones largas
    static struct option opciones_largas[] = {
        {"help", no_argument, 0, 'h'},
        {"value", required_argument, 0, 'v'},
        {0, 0, 0, 0} // marca el final del array de opciones largas
    };

    int indice_opcion = 0; // getopt_long almacena el índice de la opción aquí

    while ((opcion = getopt_long(argc, argv, ":hv:", opciones_largas, &indice_opcion)) != -1) {
        switch (opcion) {
            case 'h':
                flag = OPCION_H;
                printf("Uso: %s [--help] [--value valor]\n", argv[0]);
                printf("   --help, -h : imprime esta ayuda\n");
                printf("   --value, -v valor : valor de entrada\n");
                exit(0);
            case 'v':
                valor = atoi(optarg);
                flag = OPCION_V;
                break;
            case ':':
                fprintf(stderr, "no se ha especificado valor de entrada para -v\n");
                exit(1);
            case '?':
                fprintf(stderr, "Opción %c no válida\n", (unsigned char) optopt);
                exit(1);
            default:
                fprintf(stderr, "opcion case no esperada\n");
                exit(1);
        }
    }

    if (flag == OPCION_V) {
        printf("Opcion v, valor: %d\n", valor);
    }

    if (optind < argc) {
        printf("El primer argumento que no es opcion es %s\n", argv[optind]);
    } else {
        printf("No hay mas argumentos\n");
    }

    return 0;
}
```

Ejercicio 1. myls

Escribir un programa con una funcionalidad semejante a la del comando `ls` que cumpla con las siguientes especificaciones:

El programa admite dos posibles opciones mutuamente excluyentes:

- opción `-d`: -d ruta_directorio

- opción `-f`: -f ruta_fichero_regular

Con la opción `-d` el programa debe **comprobar que el argumento que se le ha pasado es un directorio** sobre el que se tienen permisos de lectura y ejecución y en caso de que sea así deberá recorrer las entradas de dicho directorio mostrando:

1. Si la entrada se corresponde con un fichero regular escribirá el nombre de dicho fichero y entre paréntesis el tamaño en kbytes de dicho fichero y el numero de enlaces rígidos (ejemplo: `unfichero.txt (1.5 kb, 1 link)`). Si dicho fichero tiene permisos de ejecución se escribirá un asterisco `*` después del paréntesis (ejemplo: `unejecutable (3 kb, 1 link)*`).
2. Si la entrada se corresponde con un directorio escribirá el nombre de dicho directorio entre corchetes y entre paréntesis el número de enlaces (ejemplo: `[undirectorio] (2 links)`
3. Si la entrada se corresponde con un un enlace simbólico escribirá el nombre de dicho enlace y a continuación, entre parántesis el nombre del fichero al que apunta (ejemplo: `unenlacesim (/home/manuel/unfichero2.txt)`)

Con la opción `-f` el programa debe **comprobar que el argumento que se le ha pasado es un fichero regular** y caso de que sea así deberá mostrar su nombre y entre paréntesis el número de inodo del fichero y su tamaño en kilobytes. (ejemplo: `ejfichero(inodo 18900, 1.5 kb)`)