2023

Memoria

PRÁCTICA 1.2

PAULA CASTILLEJO BRAVO

1. Compilación:

La fórmula para compilar un archivo en lenguaje C, conociendo de antemano que se trabaja en una terminal en el sistema operativo linux, consiste en el comando gcc -I include/ -o manager src/manager.c src/lista.c, en donde la opción -I indica los directorios donde han de buscarse bibliotecas mientras –o compila un programa en C y genera un archivo ejecutable.

2. Ejecución:

Una vez poseyendo el ejecutable del programa, la ejecución de este es un paso más simple desde terminal, ubicándonos en el directorio donde está el ejecutable y escribiendo, en el siguiente orden, ./manager data/solution.txt data/patrones.txt .

3. Discusión:

Una vez se domina los procedimientos de compilación y ejecución del programa, se proporcionará constancia acerca de las pequeñas conclusiones[1] durante todo el proyecto abarcado en las siguientes tres semanas.

3.1. MakeFile

En primera instancia, un archivo makefile es un conjunto de herramientas make para construir y crear ejecutables y objetos en base de los archivos dependientes .c y .h, ubicados en los directorios /src e /include respectivamente. Los nombres lógicos principales que lo componen serían:

```
1    DIROBJ := obj/
2    DIREXE := exec/
3    DIRHEA := include/
4    DIRSRC := src/

5    CFLAGS := -I$(DIRHEA) -c -Wall -ggdb
6    LDLIBS := -lpthread -lrt
7    CC := gcc
```

Desde la línea 1 hasta la 4 se instancian los directorios en su correspondiente variable, en las líneas 5 y 6 indicamos las banderas Wall, la cual reportará errores y advertencias, y lpthread, dónde se pretende cargar las bibliotecas (existentes o inventadas), y en la línea 7 indicamos el compilador. Ahora trataremos con las reglas implícitas en el documento:

```
8    all : dirs manager procesador contador test

9    dirs:
10       mkdir -p $(DIROBJ) $(DIREXE)

11   manager: $(DIROBJ)manager.o $(DIROBJ)lista.o
```

[1] Al tratarse este una práctica realizada de forma individual, no se ha generado ninguna discusión o debate como tal, pero sí existe la ayuda por parte de otros compañeros de clase y del profesorado de la asignatura en cuestión.

```
12          $(CC) -o $(DIREXE)$@ $^ $(LDLIBS)

13      procesador: $(DIROBJ)procesador.o
14          $(CC) -o $(DIREXE)$@ $^ $(LDLIBS)

15      contador: $(DIROBJ)contador.o
16          $(CC) -o $(DIREXE)$@ $^ $(LDLIBS)

17      $(DIROBJ)%.o: $(DIRSRC)%.c
18          $(CC) $(CFLAGS) $^ -o $@

19      test:
20          ./$(DIREXE)manager data/solution.txt data/patrones.txt

21      clean :
22          rm -rf *~ core $(DIROBJ) $(DIREXE) $(DIRHEA)*~ $(DIRSRC)*~
```

En la línea 8, all se usa para enumerar todos los subobjetivos necesarios para construir el proyecto. En las líneas 11, 13 y 15 se forman los ejecutables, siendo manager el del proceso principal. En la línea 17 se procede a establecer los ficheros objetos. En la línea 19, test instaura una ejecución automática del ejecutable junto a los dos ficheros a inspeccionar. En la línea 21, la función clean sirve para limpiar todas las reglas anteriores.

3.2. Lista.c

Reducido al empleo de cinco métodos, el archivo lista.c se ocupa de la creación, destrucción, inserción al final, búsqueda de un valor y cálculo de la longitud de la lista enlazada. Sin embargo, se detallarán los métodos vinculados con la inclusión de elementos en la lista ya que surgen varios conflictos al pretender almacenarse cadenas de caracteres.

Hay que tener en cuenta, sin lugar a dudas, con qué tipo de datos se trabaja, y cómo el lenguaje c no utiliza string, en su lugar forma punteros de tipo char, a los cuales se les debe otorgar una porción de memoria desde su inicialización o podría reescribir una zona ya escrita en un futuro. Esto se exhibe en el siguiente fragmento de código:

```
TNodo *nodoNuevo = malloc(sizeof(TNodo));
nodoNuevo->pSiguiente = NULL;
nodoNuevo->valor = malloc(strlen(valor) * sizeof(char));
strcpy(nodoNuevo->valor, valor);
```

Aquí lo que se pretende hacer es suministrar celdas de memoria del tamaño del tipo de dato TNodo (una estructura facilitada en la librería definitions.h), enlazar el final de la lista a NULL, repetir el procedimiento de proveer de espacio de memoria para la cadena de caracteres e trasmitir los datos con el método strcpy (este cómando copia el contenido del segundo parámetro en el primero recibido).

3.3. Procesador.c

Para concluir, se hablará del archivo procesador.c. En este proceso se comprobará si el patrón suministrado, localizado en el fichero patrones.txt, se corresponde con alguna de las palabras que conforman el texto contenido en el fichero solution.txt . Si descartamos el método main como principal, la otra función con valor atañe a procesar, recibiendo el nombre del fichero en el que se busca conocer si hay alguno de estos patrones, también enviados al método.

```
32          char *token;
33          int n_lineas = 1;
34          char *comprobacion;
35
36          //Se recorre el archivo de texto hasta el final
37          while (fgets(linea, sizeof(linea), fp) != NULL)
38          {
39              //El comando strtock divide los tramos del texto en líieas al fraccionar la cadena por saltos de lineas (\n)
40              token = strtok(linea, "\n");
41              //Se recorre cada linea almacenada en el token
42
43              while (token != NULL)
44              {
45                  //Se corrobora si el patron esta
46                  comprobacion = strstr(token, patron);
47
48                  while(comprobacion != NULL){
49                      printf("[PROCESADOR %d] Patrón '%s' encontrado en linea %d\n", getpid(), patron, n_lineas++);
50                      comprobacion = strstr(comprobacion+1, patron);
51                  }


52                  token = strtok(NULL, "\n");
53              }
54          }
```

Los dos comandos recalcables de esta sección del código son strtok y strstr, en donde se permitirá dividir una cadena usando un delimitador (\n) y se localiza cierta subcadena dentro de una secuencia de caracteres (excluyendo el carácter nulo de terminación), respectivamente. La parte más compleja de compilar fue el conteo del número de líneas para imprimir a cuál de todas está el patrón reconocido en el texto.

4. Innovaciones:

Algunos ajustes destacables en el proyecto serían los siguientes:

4.1. Manager.c

Modificar el valor de i a 1, antes tenía valor 2 y se saltaba el segundo elemento de la lista cuando solo nos interesa sortear el primero, y el menor igual (<=) a un menor.

```
159     for (int i = 1; i < longitud(patrones); i++)
160     {
161         lanzar_proceso_procesador(indice_tabla, getElementoN(patrones, i), n
                ombre_fichero);
162         indice_tabla++;
163     }
```

4.2. Contador

Llamar al método contar con los elementos de las posiciones 1 y 2, este segundo sufriendo un casteo de carácter a entero con ayuda de la función atoi.

```
14      contar(argv[1], atoi(argv[2]));
```