PRACTICA IV

Desarrollo de programas C en un entorno UNIX

Tal como se explica en el tema de gestión de memoria, la generación de un ejecutable requiere la intervención de dos programas, el compilador y el montador. En los sistemas Unix, estos programas son llamados usando un único comando llamado ***cc*** (C compiler). Este comando va llamando mediante fork() y exec() a otros comandos que el usuario normalmente nunca usa directamente y que realizan las fases de preprocesador de directivas con el prefijo # (***cpp***), compilación (***cc1***) y montado del ejecutable con las librerías (***ld***). Con las opciones del comando ***cc*** controlamos cuales de estas fases van a tener lugar.

Por defecto, si no ponemos opciones, el comando ***cc*** va a realizar todas las fases sobre los ficheros pasados como argumentos. El fichero resultante que contiene el programa ejecutable tendrá el nombre ***a.out***.

```
lac-20_$ cc modulo1.c modulo2.c
lac-20_$
```

Este comando compila separadamente el código fuente contenido en los ficheros ***modulo1.c*** y ***modulo2.c*** produciendo los ficheros ***modulo1.o*** y ***modulo2.o***, genera el ejecutable ***a.out*** montando ambos módulos más la librería del lenguaje C ***libc.a***. Finalmente borra los ficheros objeto que ha generado en la fase anterior, por lo que no podremos ver más que el fichero ***a.out*** como resultado del comando.

Podemos alterar el comportamiento de este comando mediante numerosas opciones. Algunas muy prácticas son:

-c — Solo se realiza la compilación. Suprime la fase de montaje, por lo que no se intenta generar ningún ejecutable. No se borran los ficheros objeto producido. Se usa para compilar uno o varios módulos sin tener todavía todos los módulos de código del programa completo, o para compilar cada módulo por separado.

-o nombre — Usa el nombre indicado para el fichero ejecutable en lugar de a.out. Es recomendable usar esta opción cuando se están generando varios programas en el mismo directorio para darles nombres distintos. El nombre no tiene que tener ninguna extensión predeterminada. Es más, en Unix, los ejecutables no suelen tener ninguna extensión.

-g — Incluye información sobre el programa para el depurador o ***debugger***. Si queremos ejecutar el programa paso a paso examinando el contenido de las variables es necesaria esta opción.

-O, -O2, -O3 — Optimiza el código del programa para conseguir una ejecución más rápida. Las optimizaciones dependen del nivel indicado y se refieren a cosas tales como reordenación de instrucciones para adaptarlas a la arquitectura del procesador evitando dependencias entre instrucciones adyacentes o incrementando el número de aciertos en la cache del procesador. Su efecto solo suele notarse en programas con una gran cantidad de cálculos. Estas opciones son incompatibles con ***-g***.

-Idirectorio — Indica que también se busque en el directorio indicado para localizar los ficheros ***.h*** incluidos en el código C mediante las directivas ***#include "fichero.h"***.

Mediante estas opciones podremos compilar todas las partes del programa generando los ficheros objetos correspondientes sin realizar el montado. Por ejemplo:

```
lac-20_$ cc -c prog.c f1.c f2.c f3.c
lac-20_$
```

Este comando genera los ficheros ***prog.o***, ***fl.o***, ***f2.o*** y ***f3.o***. Para realizar el montaje generando el fichero ejecutable se usa el mismo comando sobre los ficheros objeto. El comando ***cc*** reconoce los ficheros .o y sabe que no les hace falta la fase de compilación sino la de montaje:

```
lac-20_$ cc -o prog prog.o fl.o f2.o f3.o
lac-20_$
```

Incluso se pueden mezclar en un solo comando ficheros objeto ya compilados con ficheros de código fuente cuyo módulo objeto no ha sido compilado. En el siguiente ejemplo, el primer comando genera tres ficheros objeto y el segundo comando compila el fichero ***prog.c*** y lo monta con los ficheros .o que generó el primer comando, generando el fichero ejecutable:

```
lac-20_$ cc -c f1.c f2.c f3.c
lac-20_$ cc -o prog prog.c fl.o f2.o f3.o
```

Manejo de librerías

Cuando un programa o aplicación es bastante compleja, conviene dividir el código fuente en numerosos ficheros más pequeños en lugar de usar uno solo. Esto permite además que varios programadores colaboren escribiendo cada uno una parte del programa. En otras ocasiones, podemos usar conjuntos de funciones escritos por otros programadores que las han diseñado para ser usadas en multitud de proyectos. Un ejemplo de esto sería una función para calcular una transformada rápida de Fourier (FFT) sobre los datos de una tabla en memoria, que puede ser útil en una gran diversidad de aplicaciones. Estos conjuntos de funciones ya existentes que pueden usarse para construir otros programas se denominan ***bibliotecas de funciones*** o ***librerías*** (esta última palabra es más usada en español, pero es una mala traducción del termino inglés ***library***). Según su origen, las librerías de código pueden ser comerciales (de pago), públicas (gratuitas) o diseñadas por uno mismo.

Según lo que hemos visto, es necesario indicarle al compilador los nombres de los ficheros que contienen dichas funciones además de los nombres de los archivos de código de nuestra aplicación. Esto presenta varias incomodidades. La primera es que estamos recompilando para cada nueva aplicación todo el código, lo que puede llegar a hacer muy lento el proceso de compilación (hay librerías con miles de ficheros en C que tardan incluso horas en compilar). La segunda incomodidad es la necesidad de pasar los nombres de todos los ficheros .c o .o de la librería al compilador junto con los de nuestros propios ficheros. Si olvidamos un solo archivo, el montador se quejará porque no encuentra algunas funciones y no podrá generar el ejecutable. Esto puede resultar en líneas de comando con cientos de nombres de archivos.

Una solución a este problema son los ficheros con la extensión ***.a***, denominados ***librerías de código***. Un fichero de librería (***.a***) es en realidad una colección de ficheros objeto (***.o***) agrupados en un solo archivo. El montador es capaz de aceptar indistintamente como entrada ficheros objeto (***.o***) sueltos y ficheros de librería (***.a***). Cuando el montador recibe el nombre de un fichero de librería, busca dentro las funciones que necesita y las copia al ejecutable, pero no copia las funciones que no va a usar. Esto implica que aunque una librería contenga miles de funciones, el ejecutable solo va a contener el código de las funciones referenciadas en nuestro código fuente.

Existe además un comando (***ar***) para gestionar ficheros de librería que permite añadir ficheros objeto, borrarlos y listar los ya existentes. Las ventajas obvias son que una vez que compilamos los diversos módulos de una librería e incorporamos los ficheros objeto a un fichero ***.a***, no es necesario

volver a compilar más el código fuente y solo tenemos que indicar el nombre del fichero ***.a*** al compilador cada vez que necesitamos incorporar el código de la librería. Las opciones del comando ***ar*** son:

-q	incluye los ficheros objeto indicados como argumento en una librería (si el fichero librería no existe se crea).
-r	actualiza los ficheros objeto contenidos en la librería con otros más nuevos que se pasan como argumento.
-d	borra los ficheros objeto indicados de la librería.
-t	imprime la lista de ficheros objeto contenidos en la librería.
-x	extrae los ficheros objeto cuyos nombres se indican creando los ficheros ***.o*** correspondiente en el directorio actual (si no se indica ningún nombre se extraen todos los ficheros).

Como ejemplo podemos ver el con tenido de la librería ***/usr/lib/libc.a*** que contiene todas las funciones que forman parte del estándar del lenguaje C. Podemos extraer los módulos que contienen las funciones ***fork()*** y ***read()*** en el directorio actual y crear nuestra propia librería con estos módulos:

```
lac20_% ar -t /usr/lib/libc.a
__.SYMDEF
__libc.o
__T00001.o
__T00002.o
__U00008.o
....
fork.o
fpathconf.o
....
lac20_% ar -x /usr/lib/libc.a fork.o read.o
lac20_% ls *.o
fork.o  read.o
lac20_% ar -q libmia.a fork.o read.o
Creating archive file `libmia.a'
lac20_% ar -t libmia.a
fork.o
read.o
lac20_%
```

Existe una convención para los nombres de los ficheros de librería que consiste en que el nombre comienza por el prefijo ***lib*** y termina con la extensión ***.a***, por ejemplo ***lib***mia***.a***. Esto se debe a que con la opción ***-l*** del compilador solo se indica la parte intermedia del nombre para acortar la longitud línea del comando de compilación y hacerla más legible: ***cc -L. -lmia miprog.c***

Las librerías que vienen con el sistema operativo se encuentran en los directorios ***/lib*** o ***/usr/lib***. Para indicar que queremos incluir una librería determinada, existen las siguientes opciones del compilador:

-l nombre	el montador buscará la librería denominada ***libnombre.a*** o en los directorios /lib o /usr/lib para añadirla al montaje.
-L nombre_directorio	busca en el directorio indicado las librerías indicadas con la opción -l. Si no se encuentra la librería en el directorio indicado, se busca en los directorios por defecto. Para indicar una librería del directorio actual es necesario usar la opción "***-L.***" (el punto es el argumento de -L), que indica que mire en el directorio actual.

Alternativamente se puede indicar el nombre completo del fichero con la librería, pero en los sistemas que soportan librerías dinámicas para la ejecución (equivalentes a las DLLs de Windows), esta forma de indicar las librerías conlleva el uso de librerías estáticas, mientras que el uso de la opción -l implica el uso de librerías dinámicas. Para librerías del usuario creadas con el comando ***ar*** no hay diferencia alguna. Las dos líneas siguientes hacen lo mismo:

```
lac-20_$ cc -o programa miprog.c -L. -lmia
lac-20_$ cc -o programa miprog.c libmia.a
```

Ejemplo

Como ejemplo vamos a desarrollar un programa que imprima los parámetros que se le pasan en la línea de comando. La función ***main()*** tiene dos parámetros denominados ***argc*** y ***argv***. El parámetro ***argv*** es un vector de *strings* en el cual se indica el nombre con el que se invocó el programa y los parámetros de la línea de comandos. La cantidad de elementos que tiene ***argv*** esta contenida en ***argc***.

```
lac-20_$ cat prog.c
main(argc, argv)
int argc;
char *azgv[];
{
     int i;
     for (i=0; i<argc; i++)
          printf("%s \n",argv[i]);
}
lac-20_$ cc prog.c -o prog
lac-20_$ prog pl p2 p3 p4
prog
pl
p2
p3
p4
lac-20_$
```

Podríamos pensar en desarrollar una función que dado un elemento de ***argv*** compruebe si es una opción (comenzará por un guión) y nos devuelva el nombre de la opción. El procedimiento ***param.c*** implementa esta función (en la página siguiente), comprobando si un parámetro es o no una opción, y devolviendo la opción sin guión. También vemos en la página siguiente las modificaciones pertinentes a

```
lac-20_$ cat param.c
int es_opcion(str_in, str_out)
char *str_in, *str_out;
{
     if (str_in[0]=='-')
     {
          strcpy(str_out, &str_in[1]);
          return(1);
     }
     else
          return(0);
}
```

realizar en el programa principal ***prog.c***.

```
lac-20_$ cat prog.c
main(argc, argv)
int argc;
char *argv[];
{
     int i;
     char opcion[20];

     printf("programa: %s \n",argv[0]);
     for (i=1; i<argc; i++)
     {
     if (es_opcion(argv[i],opcion))
          printf("opcion: %s \n",opcion);
     else
          printf ("argumento: %s \n", argv [i] );
     }
}
```

A continuación podemos compilar, montar y ejecutar.

```
lac-20_$ cc prog.c param.c -o prog
lac-20_$ prog pl -opc p2
programa: prog
argumento: pl
opcion: opc
argumento: p2
lac-20_$
```

Ejercicios

1. Copia el fichero ***/home/sotelej/rutinas.tar***. Descomprímelo en tu propio directorio con ***tar xvf rutinas.tar***. Compila los ficheros con código C que han sido creados en el directorio rutinas y genera una librería con el nombre ***libmia.a*** que contenga todas las funciones que están definidas en ellos.
2. Escribe un programa en C que lea de su entrada estándar un fichero organizado en 5 columnas. El programa debe sumar cada columna por separado y cuando llegue al final de fichero calculará e imprimirá la media aritmética de cada columna.

   Por ejemplo la entrada podría ser:

   ```
   12   34   45   7    8
   8    7    54   32   45
   12   34   43   5    5
   ```

   y el programa imprimiría:

   ```
   10 25 47 14 19
   ```

   Las operaciones con aritmética de enteros de este programa se han de realizar con las rutinas de la librería ***libmia.c*** en lugar de usar los operadores del lenguaje C. Es decir, en lugar de poner ***a=b+c***, se usará la función ***suma*** quedando como ***a=suma(b,c)***. El resto de funciones necesarias serán las estándares de C.

   (Ayuda: El programa puede leer una línea con gets(), y luego usar sscanf() para extraer los 5 números de la cadena leida en 5 variables enteras. Esto se ha de repetir hasta que falle la lectura de gets() devolviendo un resultado -1 que indica que no hay más líneas en el fichero o que se ha

pulsado Control-D si la entrada es un terminal. Obviamente hay que sumar las columnas y llevar la cuenta del número de líneas.)

Las funciones provistas en el directorio rutinas son:

```
/* División entera a/b */
int div(int a,int b);
/* Módulo a/b */
int mod(int a, int b);
/* Multiplicación entera */
int mult(int a, int b);
/* Resta a-b */
int resta(int a, int b);
/* Suma a+b */
int suma(int a, int b);
```

3. Escribe un programa, llamado ***micat.c***, que permita volcar por pantalla el contenido de un fichero de la misma forma que lo hace el comando ***`cat`***. El fichero a mostrar por pantalla se puede pasar como argumento en la línea de comandos.
4. Escribe un programa, llamado ***micp.c***, que permita copiar un fichero en otro. El nombre del fichero a copiar y el nombre de destino de la copia se indican como argumentos en la línea de comandos.