Sistemas Operativos - Práctica 1

En esta práctica nos centraremos en temas de compilación y generación de ejecutables en C. En C, tenemos distintos tipos de archivos: archivos de código (.c), archivos de cabecera (.h), archivos object (.o), librerías (.a y .so) y archivos ejecutables. Trabajaremos en el proceso de generación de un archivo ejecutable en C a partir de varios archivos de código fuente con un cierto nivel de dependencia. Veremos cómo se puede utilizar en C parámetros pasados por línea de comandos. Trabajaremos con la creación y uso de librerías dinámicas y estáticas y veremos la diferencia entre ambas. Finalmente, aprenderemos cómo hacer un Makefile.

Descarga y descompacta el archivo SSOO-P1.zip. Encontrarás 3 archivos de código C (main.c, calculator.c y count.c) y 2 archivos header (calculator.h y counter.h). Analiza el contenido de los archivos. Observa las funciones que se llaman desde la función main.c, dónde están declaradas y dónde se encuentra su código fuente.

Archivos en C

En C, los archivos header (.h) se utilizan para declarar variables o funciones, mientras que el código fuente se encuentra en un archivo de código aparte (.c). Esto facilita la mantención y reutilización del código. Los compiladores de C no permiten utilizar una función a no ser que haya sido previamente declarada. Si una función es utilizada en varios sitios, en lugar de repetir la declaración de la función en cada archivo, se define en un solo lugar (.h). Por tanto, para utilizar una función externa, es necesario hacer primeramente un include del archivo header donde se encuentra la declaración de esta función.

Compilación en C

En C, el proceso de generar un ejecutable a partir de código fuente consta de varios pasos. Primeramente el código es preprocesado. El preproceso es un programa que se ejecuta de manera automática cuando hacemos la compilación. Lo que hace el preproceso es modificar o expandir el código fuente. Los comandos de preproceso comienzan con el signo **“#”.** Dos de los más importantes son:

1. **#define:** se utiliza principalmente para definir constantes, por ejemplo:
   #define BIGNUM 1000
   especifica que donde quiera que se encuentre BIGNUM en el código se sustituirá por 1000
2. **#include:** se utiliza para acceder a la declaración de funciones definidas en otro archivo. Por ejemplo:
   #include <stdio.h>
   hace que el preprocesador copie el contenido de <stdio.h> en el código, en la locación donde se encuentra el include.

Luego que el código es preprocesado (y expandido), se realiza la compilación, la cual genera a partir del archivo de código fuente (.c) un archivo object (.o), el cual contiene una versión binaria del código. Para compilar un archivo .c, se utiliza el siguiente comando:

>> gcc -c filename.c -o filename

Los archivos object no son directamente ejecutables, ya que hay que añadir especificaciones para las funciones y librerías externas que han sido incluidas. El linker hace este trabajo, y a partir de varios ficheros .o genera un ejecutable. Para generar un ejecutable a partir de varios archivos object (.o), se utiliza el siguiente comando:
>> gcc file1.o file2.o file3.o -o myprog

A partir del código de la práctica, realiza los siguientes pasos:

1. Abre la consola y ve a la carpeta donde se encuentran los archivos de la práctica:

a. Compila los archivos de código C por separado y genera los archivos object correspondientes (main.o, calculator.o y counter.o)

b. Haz el link de los archivos object main.o, calculator.o y count.o y genera un ejecutable llamado myprog. Como en calculator se utiliza la librería <math.h> será necesario añadir la opción -lm al final del comando.

c. Ejecuta myprog (./myprog) y comprueba que el programa se ejecuta correctamente

d. Observa qué ocurre si eliminas calculator.o del comando que genera el ejecutable. ¿Qué tipo de error da? ¿Por qué ha ocurrido este error, si no hemos modificado el código fuente y todos los include están correctos? Escribe tu respuesta en un archivo llamado Pregunta1.txt

2. Rellena el fichero de script script.sh para que realice los mismos pasos que haz hecho en el Ejercicio (1). Comprueba que se ejecuta correctamente (./script.sh)

Recibir y procesar parámetros por línea de comandos

En C podemos procesar parámetros obtenidos por línea de comandos. Estos son representados como un array de strings llamado argv (argument values), también hay un integer llamado argc (argument count) el cual representa el número de parámetros pasados por consola. Es por eso que la función main se define así:

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

Recuerda que argv[0] siempre es el nombre del programa, así que normalmente no querrás utilizarlo. Analiza este código de ejemplo para procesar los parametros pasados por línea de comandos:

```c

#include<stdio.h>


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

  printf("-------------------------");
  int i;
  for (i = 1; i < argc; i++)  /* Skip argv[0] (which is the program name).
  {
        printf(" \n %s",argv[i]);
  }

 printf("\n-------------------------\n");


}

```

Basandote en el código anterior, realiza el siguiente ejercicio.

3. Crea un archivo de código C llamado main2.c basado en main.c, el cual puede recibir estos parámetros por línea de comandos: -e,-m,-w, -o, con la siguiente funcionalidad:

   -e: se calcula y se imprime por consola la distancia euclideana

   -m: se calcula y se imprime por consola la distancia manhattan

   -w: se calcula y se imprime por consola la cantidad de palabras

   -o: se calcula y se imprime por consola el numero de ocurrencias

   Ten en cuenta que al ejecutar myprog2 se pueden pasar todos, algunos o ninguno de los argumentos por linea de comandos y que estos pueden encontrarse en cualquier orden.

Genera un script llamado script2.sh que genere un ejecutable llamado myprog2 siguiendo los mismos pasos del ejercicio (1)

Librarías estáticas

Las librerías estáticas encapsulan un conjunto de funciones y variables. Cuando se incluyen librerías estáticas en el código, estas son cargadas en tiempo de compilación y son copiadas dentro del ejecutable final. La ventaja de las librerías estáticas radica en la certeza de que el ejecutable final contiene dentro todas las funciones necesarias para su ejecución. Es posible producir un único fichero final, lo cual simplifica su distribución e instalación.

El nombre de la librería siempre comienza por “lib” y tiene una extensión .a

Para crear una librería estática en C a partir de varios ficheros .o utilizamos el siguiente comando:

>> ar rcs libname.a file1.o file2.o file3.o

Para utilizar una librería estática, haremos el #include correspondiente a sus funciones y a la hora de generar el ejecutable lo haremos de la siguiente manera:

>> gcc file4.o file5.o -o myprog libname.a

4. Crea un script llamado script_static.sh que:

a. Cree una librería estática llamada libmyutils.a dentro de la carpeta libs encapsulando el código de calculator.c y count.c

b. Genere un ejecutable llamado myprog_static el cual ejecuta el código de main.c utilizando la librería estática libmyutils.a

Librarías dinámicas

Las librerías dinámicas también encapsulan un conjunto de funciones y variables. La mayor diferencia radica en que estas librerías no son incluidas dentro del ejecutable sino que son cargadas en memoria en tiempo de ejecución. Debido a ello, deben ser distribuidas junto con el archivo ejecutable. La ventaja de tener librerías dinámicas radica en que su código puede ser actualizado y recompilado sin necesidad de modificar los ejecutables que la utilizan, haciendo así más fácil la mantención.

El nombre de una librería dinámica siempre comienza por “lib” y tiene una extensión .so (shared object). Para crear una librería dinámica en C a partir de varios ficheros .o debemos:

1. Generar los ficheros .o con la opción “-fPIC”

>> gcc -c -fPIC file1.c file2.c file3.c

2. Crear la librería a partir de los ficheros .o con el siguiente comando:

>> gcc -shared -o libname.so file1.o file2.o file3.o

Para utilizar una librería dinámica, haremos el #include correspondiente a sus funciones y a la hora de generar el ejecutable lo haremos de la siguiente manera:

>> gcc file4.o file5.o -o myprog libname.so

5. Crea un script llamado script_dynamic.sh que:

a. Cree una librería dinámica llamada libmyutils.so dentro de la carpeta libs encapsulando el código de calculator.c y count.c

b. Genere un ejecutable llamado myprog_dynamic el cual ejecuta el código de main.c utilizando la librería dinamica libmyutils.so

5. Imagina que después de realizar el paso (4) y (5), se hace una modificación en el código de calculator.c. ¿Qué debemos hacer para que esta modificación sea tenida en cuenta cuando se ejecute myprog_static y myprog_dynamic? Piensa en la solución mas eficiente en cada caso (myprog_static y myprog_dynamic) y escribe tu respuesta en un archivo llamado Pregunta5.txt.

Archivos makefile

Un programa C normalmente será compilado no invocando a gcc directamente, sino utilizando el programa make y un archivo llamado Makefile. La idea es simplificar el proceso de generación del ejecutable y asegurarse de que solo el código que ha sido modificado es recompilado. Esto no es tan importante cuando los archivos de código fuente son pocos, pero para grandes proyectos con cientos de archivos de código rápidamente se convierte en un problema! El programa make es algo así como un lenguaje de programación en miniatura, aquí sólo veremos un ejemplo sencillo. Para más información puedes ver: https://www.gnu.org/software/make/manual/

El archivo Makefile contiene reglas, las cuales están compuestas de:

- **Un target**: Usualmente el nombre de un archivo, también puede ser el nombre de una acción, por ejemplo “clean” para remover archivos
- **Una dependencia:** Es otro target que tiene que ser procesado antes que el target actual, normalmente son archivos que son necesarios para crear el target actual
- **Un comando:** Es una acción que se debe ejecutar. Puede consistir de uno o más comandos, cada uno en una línea. Importante: debes hacer un tab al inicio de cada línea de comando en los archivos Makefile, omitir este paso es un error muy frecuente

6. Abre el archivo Makefile y analiza su contenido. ¿Eres capaz de encontrar el target, las dependencias y los comandos en este archivo? Rellena las partes que faltan, de modo que el makefile realice la misma funcionalidad que el Ejercicio (1). Comprueba el resultado ejecutando el comando “make” en la consola

Archivos a entregar:

- script.sh
- script2.sh
- script_static.sh
- script_dynamic.sh
- main2.c
- Respuestas.txt
- Makefile

Deberás guardar estos archivos en una carpeta llamada NombreApellido-P1 y comprimir la carpeta en un .zip llamado: NombreApellido-P1.zip (NombreApellido quiere decir el nombre y apellido del alumno)