Universidad Nacional de La Matanza
***Ingeniería en Informática*-Taller de GNU/Linux 2003**

TP N° 14
Compilando C/C++ bajo GNU/Linux

Objetivos:

- Utilizar en forma básica el compilador de C en un sistema GNU/Linux.

Ejercicios:

Compilando un programa "C" de un solo fuente

La forma más fácil de compilar es cuando se tiene todo el código fuente en un solo archivo. Esto evita el trabajo de sincronizar muchos archivos al compilar. Supongamos que tenemos un archivo "simple_main.c" que queremos compilar.

Archivo **simple_main**

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    printf ("Hola Mundo!!\n");
    return 0;
}
```

Utilizamos la siguiente línea de comando para compilarlo:

$ gcc simple_main.c

Si la compilación resultó exitosa, se obtendrá un archivo "a.out". Si se desea que el compilador genere un archivo con nombre distinto a "a.out" lo especificamos con la opción -o nombre_archivo_salida:

$ gcc simple_main.c -o simple_main

Como resultado obtendremos un archivo ejecutable llamado "simple_main"

1) Crear el archivo "simple_main.c" y compilarlo

Ejecutando el programa resultante

Una vez que hemos creado nuestro archivo ejecutable, lo corremos simplemente tipeando :

$ simple_main

Para esto es necesario que el directorio donde se encuentra el programa figure en el PATH. En caso de que no figure, debemos ejecutar el programa especificando su ruta:

$ ./simple_main

Si aún el programa no ejecuta, puede ser que los permisos no estén correctamente seteados. Debemos utilizar el comando chmod para agregar permiso de ejecución. Si el archivo se transfiere a otro directorio o a otra máquina, es posible que se modifiquen los permisos originales.

2) Ejecutar el archivo simple_main

Creando código con información de debug

Normalmente cuando estamos haciendo un programa, deseamos tener la información para realizar un debugging del código (usar un debugger que nos permita ejecutar el código paso a paso, colocar breakpoints, mirar el contenido de las variables, etc.).
Para que el debugger tenga la capacidad de relacionar entre el programa ejecutable y nuestro código fuente, es necesario indicarle al compilador que inserte , en el código ejecutable, información que ayude al debugger en su tarea. Para esto es necesario agregar el switch -g :

$ gcc -g simple_main.c -o simple_main2

El archivo ejecutable obtenido es de mayor tamaño que el obtenido sin agregar la información de debug. Si lo deseamos podemos quitar esta información de debug con el comando strip:

$ strip simple_main2

El archivo obtenido es aún menor que el que se obtuvo sin utilizar el switch "-g". Esto se debe a que incluso estos archivos contienen cierta información, como por ejemplo nombres de funciones, etc.

3) Compilar agregando información de debug y comparar el archivo obtenido con el archivo anterior.
4) Quitar información de debug con el comando strip y comparar.

Creando código optimizado

Luego de crear un programa y debuguearlo adecuadamente, deberemos compilarlo nuevamente para obtener el código ejecutable más eficiente posible. El compilador nos puede ayudar en optimizar nuestro código, para que ejecute a mayor velocidad, para que ocupe menos espacio, o para alguna combinación de ambos. La forma básica de crear un programa optimizado es la siguiente:

$ gcc -O simple_main.c -o simple_main

El flag -O le dice al compilador que optimice el código. El compilador demorará más ya que tratará de aplicar varios algoritmos de optimización. Podemos especificar el nivel de optimización de la siguiente forma:

$ cc -O2 simple_main.c -o simple_main (nivel 2 de optimización)

Cuanto mayor es el nivel de optimización, es más probable que nuestro código deje de funcionar correctamente. Esto es debido a bugs en el compilador ya que las optimizaciones aplicadas se hacen más complejas.

Warnings adicionales del compilador

Normalmente el compilador solo genera mensajes de error sobre código erroneo que no se ajusta con el estandar de C y advertencias acerca de problemas que normalmente causan errores durante la ejecución del programa. Sin embargo, podemos decirle al compilador que nos dé mas avisos (warnings) que nos ayuden a mejorar la calidad de nuestro código y para resaltar bugs que más tarde deberemos corregir de todas formas. En el gcc esto se hace con la opción "-W". Por ejemplo para que el compilador nos dé todo tipo de advertencias:

$ gcc -Wall single_source.c -o single_source

5) Definir dentro del archivo "simple_main.c" una variable cualquiera, sin utilizarla. Compilar el fuente con y sin warnings. Comparar resultados.

Compilando un programa "C" de múltiples fuentes

Algunos de los inconvenientes de tener todo el código de programa en un solo fuente son los siguientes:

- A medida que el programa crece, el tiempo de compilación se incrementa y para cada pequeño cambio es necesario re-compilar el programa completo.
- Es mucho más complicado que varias personas tranbajen juntas en el mismo proyecto con este tipo de organización.
- Administrar el código se hace mucho más dificultoso.

La solución a esto es separar el código en varios archivos, cada uno conteniendo funciones con algún tipo de relación entre sí. Hay dos formas posibles de compilar un programa C de múltiples archivos. La primera es utilizar una sola línea de comandos para compilar todos los archivos. Supongamos que tenemos un programa que tiene sus fuentes en "main.c" , "a.c" y "b.c":

main.c:

```
#include <stdio.h>

/* define some external functions */
extern int func_a();
extern int func_b();

int
main(int argc, char* argv[])
{
    int a = func_a();
    int b = func_b();
    char c;
    char* bc = &c;
```

```
        printf("hello world,\n");
        printf("a - %d; b - %d;\n", a, b);

        return 0;
    }
```

a.c:

```
int func_a()
{
    return 5;
}
```

b.c:

```
int func_b()
{
    return 10 * 10;
}
```

Podemos compilarlo de esta forma:

$ gcc main.c a.c b.c -o hello_world

Esto ocasionará que el compilador compile cada archivo en forma separada y luego los enlace (linking)

6) Compilar el ejemplo de 3 fuentes.

El problema con esta forma de compilar es que al realizar un cambio en uno de los archivos, es necesario recompilar todos nuevamente. Para evitar esta limitación podemos subdividir el proceso en dos partes: compilación y enlace:

$ gcc -c main.cc
$ gcc -c a.c
$ gcc -c b.c

$ gcc main.o a.o b.o -o hello_world

Los primeros 3 comandos toman el código fuente y lo compilan en algo llamado “objet file”, con el mismo nombre pero con la extensión “.o”. Es el flag “-c” que le dice al compilador que solo genere archivos objeto y no archivos ejecutables. En el archivo objeto, existen símbolos no resueltos todavía.

Para crear el ejecutable final, luego de crear los 3 objetos, utilizamos el cuarto comando para enlazar todo en un solo programa.

De esta forma necesitamos re-compilar solamente los archivos modificados y luego re-enlazar todos los archivos objeto. Supongamos que cambiamos el archivo “a.c” :

$ gcc -c a.c
$ gcc main.o a.o b.o -o hello_world

Este ahorro en tiempo de compilación , se hace más importante en grandes programas con muchos archivos fuente.

7)Modificar “a.c” y re-compilar de esta última forma.

Otras herramientas de programación

Existen, además del compilador, muchas otras herramientas que ayudan al programador a crear, depurar y administrar código fuente y ejecutable. Algunas de estas herramientas son:

- g++ (compilador C++)
- make (automatiza compilación y re-compilación de código)
- gdb (debugger)
- editores para programación, IDEs (rhide, setedit, kdevelop)
- diff (encuentra diferencias entre archivos fuente)
- patch (genera parches)
- repositorio CVS
- y muchos otros...

Resolución de Ejercicios:

1)
2)
3)
4)
5)
6)
7)