Introduction to GNU Toolchain and GNU Make utility

Leandro Luciano Gagliardi
lgagliardi@unsam.edu.ar

Introducción

- GNU fue lanzado en septiembre de 1983 por Richard M. Stallman para crear un sistema operativo completo que fuera open source.
- GNU es un acrónimo recursivo de GNU is Not Unix. Unix era un sistema operativo muy popular en los años 80, por lo que Stallman diseñó GNU para que fuera compatible en su mayor parte con Unix, de modo que fuera conveniente para la gente migrar a GNU.
- El compilador y el linker de GNU se utilizan en la producción de ejecutables para destinos embebidos.
- Las principales licencias del proyecto GNU son la General Public Licence (GPL) y la Lesser General Public License (LGPL). Con el paso de los años se han establecido como las licencias más utilizadas para el Software Libre.
- Similar a Unix, GNU tiene un diseño modular. Esto significa que se pueden insertar componentes de terceros en GNU. Hoy en día mucha gente usa el nombre "Linux" para referirse a una variante de GNU.

GNU Compiler Collection (gcc)

- La colección de compiladores de GNU, o gcc, puede compilar programas escritos en C, C++, Java y otros lenguajes. Ofrece muchas opciones de línea de comandos y extensiones de sintaxis útiles, y también funciona como una potente interfaz para el linker.
- La instalación del compilador se hace mediante el siguiente comando:

$ sudo apt install build-essential

- Para saber la versión de gcc, ejecutar el siguiente comando:

$ gcc --version

Opciones de línea de comandos

Gcc admite una gran lista de opciones de línea de comandos. De hecho, hay trece categorías de opciones para elegir. La siguiente es una lista de las más importantes y de mayor utilidad inmediata para el desarrollo integrado.

Para mayor información, consulte la documentación en línea de gcc para conocer el resto.

https://gcc.gnu.org/onlinedocs/

Opciones de línea de comandos

- **-v**: Esta opción le indica a gcc que imprima todos los comandos que ejecuta durante la compilación. También hace que gcc emita datos internos de la versión y otra información útil para la resolución de problemas.

$ gcc -v

- **-g**: Este comando le indica a gcc que incluya información del debugger en sus archivos .o. Es necesario ejecutar **gdb** si se desea depurar la aplicación.

$ gcc -g bitfields.c -o bitfields.out
$ gdb bitfields.out

Opciones de línea de comandos

- **-c**: Este comando le dice a gcc que se detenga después de crear un archivo de objeto. Básicamente crea hasta los archivos objetos (.o) y no el ejecutable (.out).

$ gcc -c bitfields.c

- **-S**: La opción -S le indica a gcc que se detenga después de traducir un archivo fuente al lenguaje ensamblador, antes de que se invoque el ensamblador. El archivo de salida se llama <filename>.s.

$ gcc -S bitfields.c

Opciones de línea de comandos

- **-Wall**: Gcc admite muchas opciones para la generación de mensajes de advertencia. La opción -Wall activa todas las configuraciones de advertencia más populares de gcc, de las cuales hay muchas.

$ gcc bitfields.c -Wall

```
leandro@sentinel:~/Desktop/02 Embedded C Programming-20240625T223721Z-001/Slides/forSlides$ gcc bitfields.c
leandro@sentinel:~/Desktop/02 Embedded C Programming-20240625T223721Z-001/Slides/forSlides$ gcc bitfields.c -Wall
bitfields.c: In function ‘main’:
bitfields.c:11:13: warning: unused variable ‘a’ [-Wunused-variable]
   11 |         int a;
      |             ^
```

Optimización del código

- **-O0, -O, -O1, -O2, -O3**: Estas opciones indican a gcc que realice distintos niveles de optimización en los archivos de salida. La opción -O produce la menor optimización, mientras que -O3 produce código optimizado de forma agresiva. La opción -O0 indica a gcc que no realice ninguna optimización (la opción predeterminada si no se especifica ningún nivel de optimización).

$ gcc -O bitfields.c

Profundizaremos en este tema más adelante.

Extensiones de la sintaxis

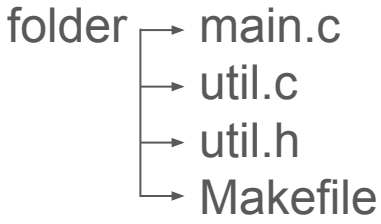
- Gcc ofrece varias extensiones de sintaxis de lenguaje, incluido lenguaje ensamblador en línea.

```c
#include <stdio.h>

// Función para contar los bits en un entero usando assembler en línea
int count_bits(int x) {
    int count;
    asm ("popcnt %1, %0"
         : "=r" (count) // output
         : "r" (x)      // input
         : );           // no clobbered registers
    return count;
}

int main() {
    int number = 29; // 11101 en binario, tiene 4 bits a 1
    int result = count_bits(number);
    printf("Number of 1 bits: %d\n", result);
    return 0;
}
```

Make utility

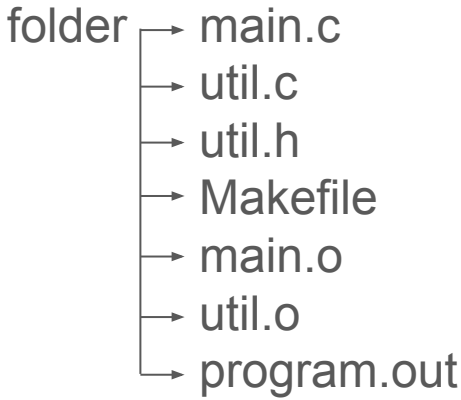
- La utilidad Make utiliza el Makefile para compilar el proyecto completo. El Makefile contiene comandos que le dicen al Make cómo compilar el proyecto dados los archivos fuente. El Makefile contiene las dependencias y las reglas de construcción.

Ejemplo de Makefile




```
# Definición de variables
CC = gcc                    # compilador
CFLAGS = -Wall -g           # flags de compilación
OBJ = main.o util.o         # objetos a generar

# Regla para construir el ejecutable 'program'
program.out: $(OBJ)
        $(CC) $(CFLAGS) -o $@ $^

# Regla para construir 'main.o' a partir de 'main.c'
main.o: main.c util.h
        $(CC) $(CFLAGS) -c $< -o $@

# Regla para construir 'util.o' a partir de 'util.c'
util.o: util.c util.h
        $(CC) $(CFLAGS) -c $< -o $@

# Regla de limpieza para eliminar archivos generados
clean:
        rm -f *.out $(OBJ)
```

Ejecución de Makefile



Execute:

$ make

```
C Programming -> ls
main.c  Makefile  util.c  util.h
C Programming -> make
gcc                -Wall -g         -c main.c -o main.o
gcc                -Wall -g         -c util.c -o util.o
gcc                -Wall -g         -o program.out main.o util.o
C Programming -> ls
main.c  main.o  Makefile  program.out  util.c  util.h  util.o
C Programming ->
```

Source files

main.c

```c
#include "util.h"

int main(){
        imprimir();
        return 0;
}
```

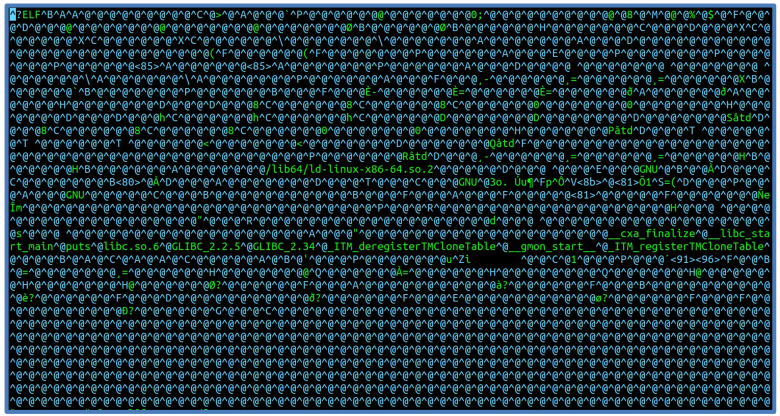
util.h

```c
#ifndef UTIL_H
#define UTIL_H

#include <stdio.h>
void imprimir(void);

#endif //UTIL_H
```

util.c

```c
#include "util.h"

void imprimir(void){
        printf("Hello world!\n");
}
```

Los .o y .out no se podrán ver mediante un editor de texto convencional.



Variables automáticas

- **$@**:
  Representa el nombre del archivo objetivo que se está construyendo.
  Por ejemplo, si la regla es para program, **$@** se expandirá a program.

- **$<**:
  Representa la primera dependencia de la regla.
  Es útil en reglas de compilación donde generalmente hay una única fuente que se convierte en el archivo objetivo.
  Por ejemplo, si la regla es para construir main.o a partir de main.c, **$<** se expandirá a main.c.

- **$^**:
  Representa la lista de todas las dependencias de la regla, separadas por espacios y sin duplicados.
  Es útil cuando se necesita pasar todas las dependencias a un comando, como en el caso de la vinculación de múltiples archivos objeto.

Referencias

GNU webpage: https://www.gnu.org/
gcc documentation: https://gcc.gnu.org/onlinedocs/
make utility documentation: https://www.gnu.org/software/make/manual/make.html

UNIVERSIDAD NACIONAL DE HURLINGHAM

Universidad Nacional de San Martín

Extra Slides

Leandro Gagliardi
lgagliardi@unsam.edu.ar

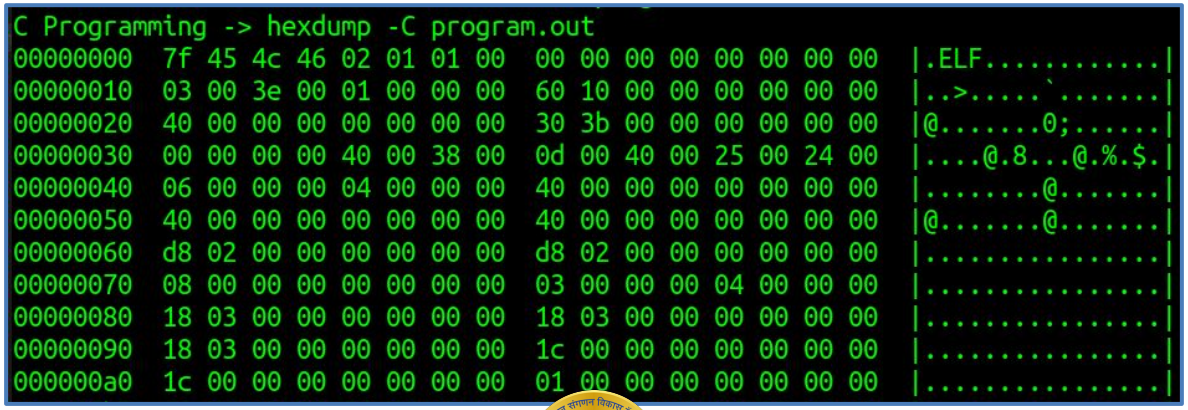
hexedit

- Para poder ver el binario es necesario instalar algún editor hexadecimal.

$ sudo apt-get install hexedit

- Ejecutar el siguiente comando para visualizar el contenido del binario:

$ hexdump -C «nombre del fichero binario a examinar»



od command

- Para poder ver el binario es necesario ejecutar el siguiente comando

$ od -h «nombre del fichero binario a examinar»

```
C Programming -> od -h main.o
0000000 457f 464c 0102 0001 0000 0000 0000 0000
0000020 0001 003e 0001 0000 0000 0000 0000 0000
0000040 0000 0000 0000 0000 0860 0000 0000 0000
0000060 0000 0000 0040 0000 0000 0040 0016 0015
0000100 0ff3 fa1e 4855 e589 00e8 0000 b800 0000
0000120 0000 c35d 0090 0000 0005 0801 0000 0000
0000140 0002 0000 1d00 0000 0000 0000 0000 0000
0000160 0000 0000 0000 0014 0000 0000 0000 0000
0000200 0000 0801 0007 0000 0100 0704 0000 0000
0000220 0101 0008 0000 0100 0702 0000 0000 0101
0000240 0006 0000 0100 0502 0000 0000 0403 6905
0000260 746e 0100 0508 0000 0000 0101 0006 0000
0000300 0400 0000 0000 0502 0506 0000 0000 0301
```

Cambio de nombre en la línea de comandos

```
$ PS1=”C_Programming -> ”
```