Herramientas de desarrollo

gcc, gdb* y *make

Pedro Merino Gómez
Jesús Martínez Cruz
Dpto. Lenguajes y Ciencias de la Computación
Universidad de Málaga

Índice

- Compilando con ***gcc***
- El gestor de proyectos ***make***
- Depurando con ***gdb***

Compilando: *gcc*

- gcc (GNU C compiler) es el compilador en línea de comandos más extendido en el mundo UNIX.
- Sirve además como envoltorio de herramientas anexas al proceso de implementación de código: preprocesador (cpp), ensamblador (as), enlazador (ld)...

El primer ejecutable

1. Para compilar:
   `gcc –c main.c ( Salida: main.o )`
2. Para crear el ejecutable:
   `gcc main.o –o programa`

   Nombre del ejecutable.
   Por defecto: **a.out**
3. Para ejecutarlo:
   `./programa`

Los pasos 1 y 2 se pueden integrar en uno:
`gcc main.c –o programa`

Algunos flags útiles en compilación

- El compilador avisa de errores y **warnings** de código no conforme al C estándar.
  - Sin embargo, la información por defecto podría ocultar algunas posibles fuentes de error en tiempo de ejecución, o que podrían darse al portar código entre plataformas.
- Usamos el flag **–W** para especificar los tipos de warnings que nos interesan (o todos ellos).

```
gcc –Wall main.c –o programa
```


Algunos flags útiles (y II)

- Los flags de **optimización** permiten un código más compacto y/o más veloz.
  - La optimización suele ser conservativa (hasta cierto punto), manteniendo el comportamiento original del programa previo a este paso.
- Usamos el flag **–O** seguido de un número que indica el nivel de optimización (consultar manual del compilador!!).

```
gcc –O2 main.c –o programa
```

Añadiendo archivos .h

- Hay archivos de cabecera que no están donde el compilador espera (/usr/include/ ó ./).
- Para incluirlos al compilar, se utilizan los flags **–I**ruta y **–i**archivo.h

Añadiendo librerías

- Las librerías se construyen con esta nomenclatura:
  lib + nombrelibrería + (.a | .so)

  Por ejemplo: `libcrypt.a`
- Y para enlazarlas con nuestra aplicación, se utilizan los flags **–L**ruta y **–lnombrelibrería**

Compilando muchas fuentes

- Hay que estructurar el código lo máximo posible, usando distintos ficheros de fuentes.
- Para compilarlos y enlazarlos, usamos:

```
gcc main.c rutinas.c protocolos.c –o programa
```

En cualquier orden

¿Y en C++?

- El compilador GNU se denomina **g++**
- Se utiliza de la misma forma que gcc:

```
g++ main.cc –o programa
```


Gestión de proyectos: *make*

- Cuando nuestro proyecto incluye varios ficheros fuente, se repiten ciertas tareas tediosas por cada cambio realizado.

Hay que recompilar y enlazar todos los fuentes de nuevo, de forma ordenada en función de las

¡make automatiza!

Primeros pasos con *make*

- La ejecución del programa es simple:
  `make [regla]`
- Busca en el directorio actual el archivo de reglas, llamado **makefile** o **Makefile**.
- El archivo de reglas contiene las instrucciones para compilar el proyecto.

La estructura del *makefile* (I)

1. Definiciones de variables:

```
CC= gcc
CFLAGS= -g –Wall –D_POSIX_
```

2. Reglas:

```
regla:  dependencias
        acción
```

Otras reglas
o
Cambios en ficheros

$(CC) $(CFLAGS) –o programa

La estructura del *makefile* (y II)

- Ejemplos de reglas:

```
all:        main
main:       main.o
            gcc –g main.o –o main
main.o:     main.c main.h
            gcc –g –c main.c
```


Automatizando aún más (I)

- Algunos consejos para la creación de variables

```
CC= gcc
CFLAGS= -g -Wall
LD= gcc
LDFLAGS= -lcrypt
OBJS= main.o rutinas.o protocolos.o
EXE= programa
```


Automatizando aún más (II)

- Consejos de creacion de reglas:

```
all:    $(EXE)

$(EXE): $(OBJS)
        $(LD) $(LDFLAGS) $(OBJS) –o $(EXE)

main.o: main.c protocolos.h rutinas.h
        $(CC) $(CFLAGS) –c main.c

#... sigue una regla por cada fichero .c
```


Automatizando aún más (III)

- Usando reglas implícitas

```
%.o:    %.c
        $(CC) $(CFLAGS) –c $<
```

%: significa “cualquiera”. Mantiene su valor constante mientras se ejecuta la regla.
$<: cadena que incluye el valor de la dependencia (%.c), o dependencias que cumplen con la regla.

Automatizando aún más (y IV)

- Un problema de las reglas implícitas es que se pierde la flexibilidad para expresar dependencias particulares de cada fichero (por ejemplo los **.h**).
- Usamos la regla **depend**:
  (y llamamos a *make depend* antes de a *make all*)

```
SRCS= fichero1.c fichero2.c fichero3.c
depend:
   rm –f .depend
   makedepend –f- -- $(SRCS) > .depend

include .depend
```

Este archivo contiene las dependencias actualizadas de cada fichero fuente (no incluye comandos).

El depurador: *gdb*

- Es una herramienta imprescindible para reparar errores que han sucedido en tiempo de ejecución.
- El ejecutable de pruebas (sensible a la depuración) ha de ser diferente al generado para producción final.

```
gcc –g main.c –o programa
gdb programa
```


Dentro del depurador (I)

- Para ejecutar el programa:

```
run argumento1 argumento2 ...
```

- Para establecer puntos de ruptura

```
break fichero.c:5   (línea)
break nombrefuncion (función)
```

- Ejecución paso a paso

```
next (un paso de ejecución completo).
step (si existe una llamada a función,
  entra en ella y para)
```


Dentro del depurador (II)

- Variables y expresiones (del contexto actual)

```
print var_name
print i*2
print miarray[8]
```

- ¿Dónde estoy? ¿Por dónde voy?

```
where (Genera la pila de funciones hasta llegar a
       main)
```


Dentro del depurador (y III)

- Los procesos que usan mecanismos de comunicación/sincronización son especialmente difíciles de depurar.
- Para enlazar con un proceso que todavía se esta ejecutándose:

```
gdb programa PID
```

- Para conmutar entre threads:

```
thread NUMERO
```


Depurando *cores*

- En algunos casos, y ante comportamientos anómalos, el kernel aborta la ejecución del programa, generando un fichero *core.*
  - El *core* no es más que un volcado de memoria a disco para ese programa.
- Si se utilizó el flag de depuración **–g**, se puede utilizar **gdb** para inspeccionar las variables, el estado en que quedó, el contenido de la memoria...

```
gdb /ruta/al/programa /ruta/al/core
```