Programación Orientada a Objetos

Grado en Ingeniería Informática

Depuración de proyectos con GDB

Departamento de Ingeniería Informática
Universidad de Cádiz

ver 0.1

Indice

Sección 1 | Introducción

Introducción

Desarrollo de un programa

En el proceso de escritura de un programa aparecen muchos errores que debemos ir corrigiendo hasta llegar a escribir el programa correctamente. Los tipos de errores que podemos encontrar son:

- Errores de sintaxis: Son errores producidos por el incumplimiento de las reglas sintácticas del lenguaje que estamos utilizando. Hasta que no se corrigen estos errores no podemos obtener un código ejecutable del programa. Este tipo de errores se corrigen mediante el **compilador.**
- Errores de ejecución: Son errores que producen cuando se realiza una operación no válida que da lugar a la finalización anormal del programa (división por cero, abrir ficheros no existentes, etc...)
- Errores lógicos: Son errores debidos a la traducción incorrecta de las especificaciones del programa. El programa termina normalmente pero no produce los resultados esperados.

Introducción

Depuradores

- Un **depurador** es una herramienta de programación que nos permite la detección y corrección de errores de ejecución y errores lógicos. Este tipo de errores aparecen a partir del momento en el que tenemos un código sintácticamente correcto del que obtenemos un programa ejecutable. Es al probar el programa cuando se puede detectar la aparición de estos errores.
- El depurador nos permite ejecutar el programa de manera que podemos ir **acotando las zonas** donde se están produciendo los errores,**observar bajo que valores o condiciones** se dan los mismos, etc.
- El depurador nos ayudará a corregir problemas de incorreción puntuales de los programas, pero en ningún caso sirve para arreglar un programa mal especificado o mal concebido.

Introducción

Depuradores de C++

Un depurador o **debugger** permite observar cómo funciona un programa en desarrollo durante su ejecución

- **GDB** o el GNU DeBugger, es el programa usado para depurar los programas en GNU/Linux
- **GDB** además permite depurar programas que terminaron con un fallo (depuración postmortem) e incluso procesos que ya se encuentran en ejecución.
- **GDB** permite hacer cuatro cosas que ayudan a detectar y corregir errores dentro del código del programa
  - Comenzar a ejecutar un programa especificando cualquier cosa que pueda afectar a su comportamiento
  - Hacer que el programa se pare en cualquier línea de código. Se puede incluso especificar bajo qué condiciones se debe parar.
  - Examinar que ocurre cuando el programa esta parado
  - Cambiar cosas dentro del programa que están en ejecución

Sección 2 | Depurando

Depurando

Empezamos con GDB

- Para sacar el máximo partido a gdb es necesario que el programa que vamos a depurar esté compilado con información simbólica, es decir, en el código ejecutable del programa deben aparecer los nombres de las variables que lo conforman, funciones, tipos, números de línea, etc.
- Para compilar un programa con información simbólica es necesario pasar la opción -g al compilador. Veamos un ejemplo:

```
$ gcc -g -o hola hola.c
```

- Si quiere incluir información de depuración adicional, útil exclusivamente para gdb, la opción será:

```
-ggdb en lugar de -g.
```

Depurando

Ejecutamos GDB

- Para invocar a gdb simplemente hay que escribir gdb en la línea de comandos, aunque normalmente, se indica en la llamada el nombre del programa que queremos depurar.

```
$ gdb hola
```

- Una vez arrancado, gdb carga el binario y lo deja dispuesto para comenzar su ejecución. En este momento aparecerá en pantalla un indicador como este:

```
(gdb)
```

Depurando

Ejecutamos GDB

```
miguel@miguel-Aspire-3830TG:~/pruebas/dos$ gdb
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Para las instrucciones de informe de errores, vea:
<http://bugs.launchpad.net/gdb-linaro/>.
(gdb)
```

Depurando

Ayuda en GDB

Eso significa que **gdb** está a la espera de un comando (llamados gdb-commands u órdenes-gdb). Para obtener una lista de dichas órdenes y una pequeña explicación de cada una de ellas se usa help. Veremos una lista similar a la siguiente:

```
(gdb) help
List of classes of commands:

aliases -- Alias de otras órdenes
breakpoints -- Para el programa en ciertos puntos
data -- Examinando datos
files -- Especificando y examinando archivos
internals -- Órdenes de mantenimiento
obscure -- Ocultar características
running -- Corriendo el programa
stack -- Examining the stack
status -- Preguntas de estado
support -- Facilidades de soporte
tracepoints -- Tracing of program execution without stopping the program
user-defined -- Órdenes definidas por el usuario

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb)
```

Depurando

Ayuda en GDB

Cada categoría engloba órdenes y subcategorías. Para obtener información sobre una categoría concreta utiliza la orden help seguida del nombre de la categoría en cuestión, por ejemplo:

(gdb) help running

```
(gdb) help running
Corriendo el programa.

List of commands:

advance -- Continue the program up to the given location (same form as args for
break command)
attach -- Attach to a process or file outside of GDB
continue -- Continue program being debugged
detach -- Detach a process or file previously attached
detach checkpoint -- Separar de un punto de comprobación (experimental)
detach inferiors -- Detach from inferior ID (or list of IDS)
disconnect -- Desconectado desde el objetivo
finish -- Execute until selected stack frame returns
handle -- Specify how to handle a signal
inferior -- Use esta orden para cambiar entre inferiores
interrupt -- Interrupt the execution of the debugged program
jump -- El programa continua siendo depurado en una línea específica o direcció
kill -- Terminar la ejecución del programa que está siendo depurado
kill inferiors -- Kill inferior ID (or list of IDs)
next -- Step program
nexti -- Pasar una instrucción exactamente
reverse-continue -- Continue program being debugged but run it in reverse
---Type <return> to continue, or q <return> to quit---
```

Depurando

Órdenes básicas

- **break [fichero:]funcion** Inserta un punto de ruptura al comienzo de la función indicada, también se puede utilizar un número de línea. Cuando la ejecución del programa alcanza el punto establecido, gdb lo detiene y se solicita una nueva orden.
- **run [arglist]** Ejecuta el programa que queremos depurar (el que anterioremente pasamos en la línea de comandos). Se puede pasar una lista de argumentos al programa como se haría si se invocara directamente.
- **bt** Backtrace o traceo inverso, muestra la pila de ejecución del programa.
- **print <expr>** Muestra el valor de una variable o expresión que pueda evaluarse en tiempo de ejecución

Depurando

Órdenes básicas

- **continue** Continúa ejecutando el programa después de haberse producido cualquier tipo de parada.
- **next** Ejecuta la siguiente línea del programa. Para ello, el programa debe estar parado (stopped) pero en ejecución. Permite hacer un traceo a nivel, es decir, ante una llamada a función la ejecutará de forma atómica y después seguirá con la siguiente sentencia.
- **step** Ejecuta la siguiente línea del programa, pero en este caso realiza un traceo contínuo, es decir, ante una llamada a función la siguiente sentencia a ejecutar será la primera sentencia de dicha función.
- **list** Imprime en pantalla la siguiente línea que se va a ejecutar y su contexto, normalmente las 5 líneas anteriores y las 5 posteriores.
- **quit** Sale de gdb

Depurando

Abreviaturas

Cualquier orden de **gdb** puede ser abreviada siempre que contenga el texto suficiente como para no ser confundida con otra de nombre similar

- **p** print
- **d** display
- **n** next
- **s** step
- **c** continue
- **h** help

Sección 3 | Ejemplo de depuración

Ejemplo
de depuración

Empezaremos con un programa sencillo. Guarda el siguiente código con el nombre test.c

```
#include <stdio.h>

int main() {

 int v = 0;

   int i;

 for (i = 0; i < 5; i++) {

   v += i;

 }

 printf("Resultado: %i\n", v);

 return 0;

}
```

Ejemplo
de depuración

- Lo primero que hemos de hacer es compilar dicho programa con la opción g"para que se incluya la información necesaria

$ gcc -g -o test test.c

- O podríamos utilizar el **make** ya aprendido y además poner la opción especial **-ggdb**

$ make test CC=gcc CFLAGS= ”-ggdb -Wall -ansi -pedantic”

- Una vez compilado, podemos comenzar la depuración invocando a **gbd** con el nombre del archivo compilado.

$ gdb test

Ejemplo

listado del código

El primer comando que veremos es **(l)ist**. Este comando muestra el código fuente del programa que estoy depurando.

```
Leyendo símbolos desde /home/miguel/pruebas/dos/test...hecho.
(gdb) l
1       #include <stdio.h>
2
3       int main() {
4        int v = 0;
5        int i;
6        for (i = 0; i < 5; i++) {
7          v += i;
8        }
9        printf("Resultado: %i\n", v);
10       return 0;
(gdb)
```

Ejemplo

ejecuión

Para ejecutar el programa, utilizamos el comando **(r)un**. Como aún no hemos puesto ningún punto de ruptura, el programa se ejecutará entero.

```
(gdb) r
Starting program: /home/miguel/pruebas/dos/test
Resultado: 10
[Inferior 1 (process 28925) exited normally]
(gdb)
```

Ejemplo

breakpoint

Vamos a colocar ahora un punto de ruptura. Como su nombre indica, un punto de ruptura es un punto donde la ejecución de nuestro programa se detiene. Para colocar un punto de ruptura utilizamos el comando **(b)reakpoint** seguido de la **línea** donde queremos ponerlo o el **nombre de una función**

```
(gdb) br 7
Punto de interrupción 1 at 0x40050c: file test.c, line 7.
(gdb)
```

Ejemplo

breakpoint

Si ahora ejecutamos el programa se detendrá cuando llegue a esa línea. Para continuar la ejecución del programa utilizamos el comando **(c)ontinue**. En nuestro ejemplo, el programa se volverá a detener en el mismo punto de ruptura hasta 5 veces antes de acabar normalmente.

```
(gdb) r
Starting program: /home/miguel/pruebas/dos/test

Breakpoint 1, main () at test.c:7
7           v += i;
(gdb) c
Continuando.

Breakpoint 1, main () at test.c:7
7           v += i;
(gdb) c
Continuando.

Breakpoint 1, main () at test.c:7
7           v += i;
(gdb)
```

En C++ es parecido a C, pero habrá que tener en cuenta el polimorfismo de las funciones

(gdb) break TestClass::testFunc(int)

Ejemplo

paso a paso

Puesto que tenemos un punto de ruptura en el bucle y este da 5 vueltas, hemos tenido que continuar 5 veces para ejecutar el programa completo. También es posible ejecutar el programa línea a línea con el comando **(n)ext**.

```
(gdb) r
Starting program: /home/miguel/pruebas/dos/test

Breakpoint 1, main () at test.c:7
7           v += i;
(gdb) n
6         for (i = 0; i < 5; i++) {
(gdb) n

Breakpoint 1, main () at test.c:7
7           v += i;
(gdb) n
6         for (i = 0; i < 5; i++) {
(gdb) n

Breakpoint 1, main () at test.c:7
7           v += i;
(gdb) n
6         for (i = 0; i < 5; i++) {
(gdb)
```

Ejemplo

valor de una variable

En cualquier momento podemos ver el valor de una variable con el comando **(p)rint** seguido del nombre de la variable.

```
(gdb) r
Starting program: /home/miguel/pruebas/dos/test

Breakpoint 1, main () at test.c:7
7           v += i;
(gdb) n
6         for (i = 0; i < 5; i++) {
(gdb) n

Breakpoint 1, main () at test.c:7
7           v += i;
(gdb) n
6         for (i = 0; i < 5; i++) {
(gdb) n

Breakpoint 1, main () at test.c:7
7           v += i;
(gdb) n
6         for (i = 0; i < 5; i++) {
(gdb) p v
$1 = 3
(gdb)
```

Ejemplo

eliminación de puntos de ruptura

Para eliminar un punto de ruptura utilizamos el comando **delete** y el número del punto a eliminar. Podríamos ver los puntos de ruptura que tenemos con el comando **info breakpoints**

```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040050c in main at test.c:7
(gdb) delete 1
(gdb) inf breakpoints
No breakpoints or watchpoints.
(gdb)
```

Ejemplo

Más opciones de los breakpoints

- **tbreak** Es un breakpoint temporal, es decir solo se pararía el programa una vez en el
- **disable [breakpoint]** Deshabilita el breakpoint referenciado por su número
- **ignore [breakpoint ntimes]** Se salta un breakpoint un número determinado de veces

Postmortem

¿Que es?

Cuando un programa comete una operación no permitida: división por cero, acceso a un fichero inexistente, acceso a memoria que no le pertenece, utilización de un puntero nulo, etc. el proceso que instancia el programa muere, mejor dicho, el sistema operativo lo mata; y en ese momento el sistema vuelca el contenido de la memoria que ocupaba el programa a un fichero al que le da por nombre core. Por eso, cuando un programa falla, el sistema muestra el motivo del fallo e indica que se creó el core. Por ejemplo:

Segmentation fault (core dumped)

Postmortem

funcionamiento

Un archivo **core** es de gran ayuda ya que permite reproducir y eliminar errores esporádicos, es decir, que ocurren en circunstancias muy específicas y difíciles de reproducir, como ocurre por ejemplo en programas de comunicaciones.
Con **gdb** es posible analizar un core; para ello se le invoca de la siguiente forma:

```
$ gdb <fichero> core
```

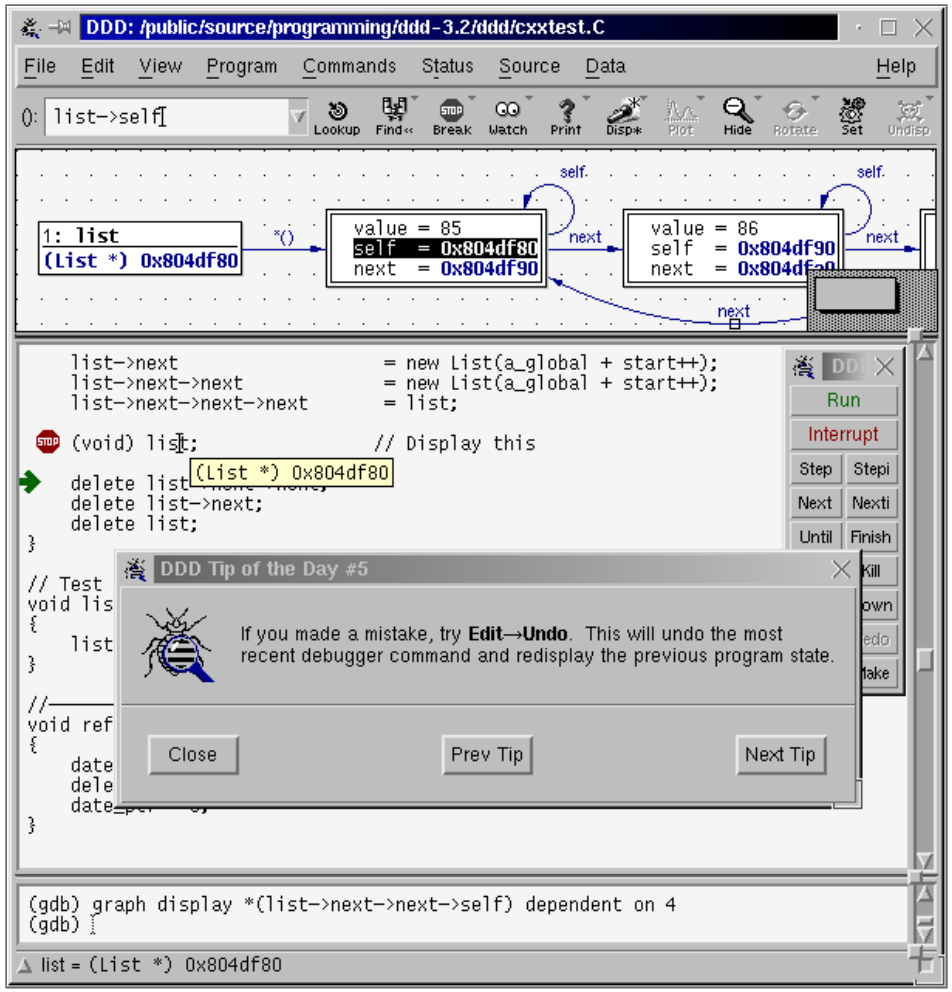
siendo <fichero> el fichero ejecutable que generó el core, es importante que sea la misma versión del programa y no otra. De este modo gdb es capaz de hacer una correspondencia entre el código objeto del programa y la imagen de memoria almacenada en el core y determinar exactamente en que punto falló el programa y porqué.

Sección 4 | DDD

DDD

Introducción

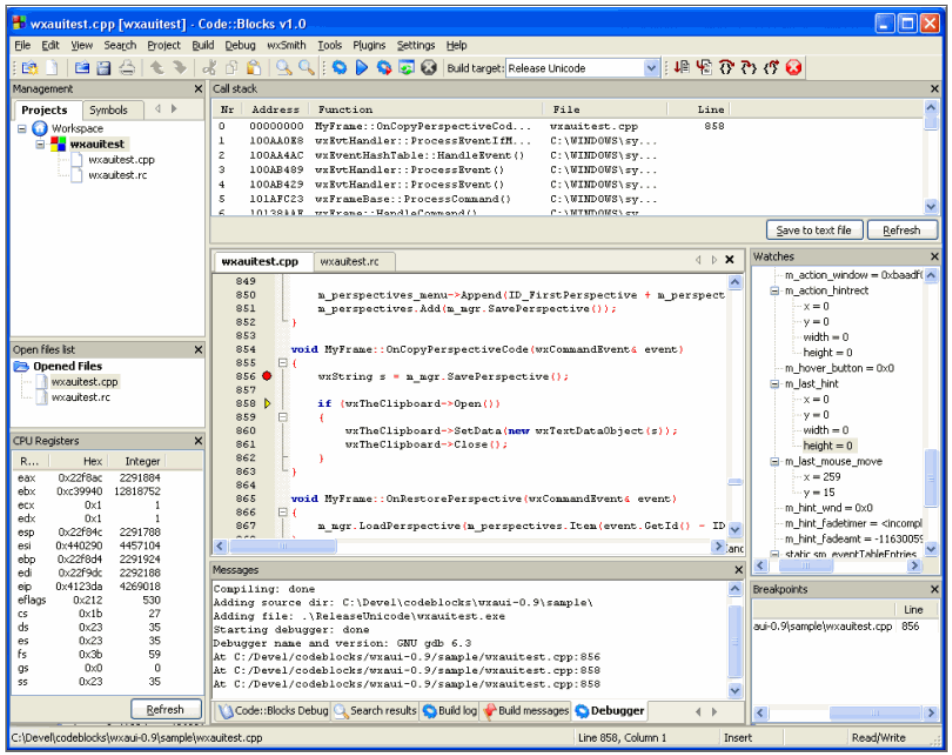
Data Display Debugger o DDD es una popular interfaz gráfica de usuario para depuradores en línea de comandos como GDB.



IDEs

Introducción

Aplicaciones con **Codeblocks**, **Dev-c++**, **Eclipse** traen integrado el debugger



Sección 5 | Bibliografía

Bibliografía

Introducción

- http://arco.esi.uclm.es/~david.villa/doc/repo/gdb/gdb.html
- http://www.lsi.us.es/~javierj/ssoo_ficheros/GuiaGDB.htm
- http://informatica.uv.es/iiguia/HP/docs/mini_gdb.pdf