

GDB 101

Dpto de Informática. IES Rio Arba de Tauste

RSA (adaptado de jjgrodriguez@gmail.com)

Curso 2023-24

Este documento pretende ser breve y sencilla guía práctica al **depurador GDB**

Usaremos como ejemplo este sencillo programa:

```
1  #include <stdio.h>
2
3  int main() {
4
5  int v = 0;
6
7  int i;
8
9  for (i = 0; i < 5; i++) {
10
11      v += i;
12
13  }
14
15  printf("Resultado: %i\n", v);
16
17  return 0;
18
19 }
```

Lo primero es compilar con opción `-g` para que no sea optimizado e incluya la `i` necesaria para la depuración.

```
gcc test.c -g -o depurable.exe
```

Compilado el ejecutable, podemos comenzar la depuración invocando **gdb** con su nombre. Con la opción `-q` evitamos cabeceras introductorias del programa (menos ruido)

```
gdb depurable.exe -q
Reading symbols from depurable.exe...
(gdb)
```

Prompt (gdb) del depurador listo para recibir órdenes. El primer comando es `(l)ist`. Muestra el código fuente en la zona próxima al momento de control.

```
(gdb) l
1  #include <stdio.h>
2
3  int main() {
4
5  int v = 0;
6
7  int i;
8
9  for (i = 0; i < 5; i++) {
10
(gdb)
```

Para ejecutar el programa, comando `(r)un`. Como aún no hemos puesto ningún punto de ruptura, ejecutará sin pausas y solo se parará cuando precise una entrada. Lo repetimos porque en la primera ejecución propone una descarga de debuginfo¹

```
(gdb) r
Starting program: /home/ray/gdb.studio/test/depurable.exe
```

This GDB supports auto-downloading debuginfo from the following URLs:
<<https://debuginfod.ubuntu.com>>

¹Ajustaremos para que no haga falta más veces con el comando `echo set debuginfod enabled > ~/.gdbinit`

```
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[Depuración de hilo usando libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Resultado: 10
[Inferior 1 (process 144499) exited normally]
```

```
(gdb) r
Starting program: /home/ray/gdb.studio/test/depurable.exe
[Depuración de hilo usando libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Resultado: 10
[Inferior 1 (process 144570) exited normally]
(gdb)
```

Ahora vamos a colocar el primer *breakpoint* -punto de ruptura-, donde la ejecución se detiene. Comando (b)reakpoint con la línea donde lo queremos.

```
(gdb) b 10
Punto de interrupción 1 at 0x55555555165: file test.c, line 11.
```

Ya tenemos un punto de ruptura en la línea 10. Si ahora ejecutamos, el programa se detendrá cuando llegue a ella e indicará la siguiente línea que será ejecutada al continuar, en este caso la 11:

```
(gdb) r
Starting program: /home/ray/gdb.studio/test/depurable.exe
[Depuración de hilo usando libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
```

```
Breakpoint 1, main () at test.c:11
11      v += i;
```

Para continuar la ejecución utilizamos (c)ontinue. En nuestro ejemplo, el programa se volverá a detener en el mismo punto de ruptura, dentro de un bucle, y podremos repetir esto 5 veces:

```
(gdb) c
Continuando.
```

```
Breakpoint 1, main () at test.c:11
11      v += i;
(gdb) c
Continuando.
```

```
Breakpoint 1, main () at test.c:11
11      v += i;
(gdb) c
Continuando.
```

```
Breakpoint 1, main () at test.c:11
11      v += i;
(gdb) c
Continuando.
```

```
Breakpoint 1, main () at test.c:11
11      v += i;
(gdb) c
Continuando.
Resultado: 10
[Inferior 1 (process 144699) exited normally]
```

(gdb)

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/ray/gdb.studio/test/depurable.exe

Downloading separate debug info for system-supplied DSO at 0x7ffff7fc6000

[Depuración de hilo usando libthread_db enabled]

Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at test.c:11

11 v += i;

(gdb) n

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

(gdb) n

Breakpoint 1, main () at test.c:11

11 v += i;

(gdb) n

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

(gdb) n

Breakpoint 1, main () at test.c:11

11 v += i;

(gdb) n

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

(gdb) n

Breakpoint 1, main () at test.c:11

11 v += i;

(gdb) n

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

(gdb) n

Breakpoint 1, main () at test.c:11

11 v += i;

(gdb) n

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

(gdb) n

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

(gdb)

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

(gdb) p v

\$1 = 1

(gdb) c

Continuing.

Breakpoint 2, main () at test.c:9

9 v += i;

(gdb) c

Continuing.

Breakpoint 2, main () at test.c:9

```
9 v += i;
```

```
(gdb) p v
```

```
$2 = 6
```

```
(gdb)
```

Para eliminar un punto de ruptura utilizamos el comando `delete` y el número del punto a eliminar. En nuestro ejemplo

```
(gdb) delete 1
```

```
(gdb)
```

Por último, para salir del depurador utilizamos el comando ``(q)uit``.

Hay mucha más información disponible tanto en la página del manual como en los archivos `info`:

```
```bash
```

```
man gdb
```

```
info gdb
```

Hay GUIs para trabajar con **`gdb`**. Uno es DD y puede descargarse en <http://sourceforge.net/projects/ddd>