Depuración de programas con gdb (GNU De-Bugger)

Los depuradores son utilidades que ayudan a eliminar fallos de los programas en tiempo de ejecución, una vez que se han eliminado los fallos de sintaxis en tiempo de compilación. La aplicación `gdb` es una herramienta apta para depurar tanto código como datos. Como ejemplo se va a depurar el programa `copiar.c`, en el que se ha incluido una pequeña modificación para que, aunque en apariencia sea correcto, contenga un error que se manifestará en tiempo de ejecución y que se detectará mediante el depurador. La modificación realizada es en la llamada para abrir el segundo archivo, donde ha de realizarse la copia. En esta llamada, los permisos deberían ser de lectura/escritura y no sólo de lectura. El código fuente del programa es el siguiente:

```
1  #include <stdio.h>
2  #include <fcntl.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <stdlib.h>
6
7  #define PERMS 0644
8  #define BUFSZ 512
9
10 int main(argc, argv)
11       int argc;
12       char *argv[];
13 {
14    int fildes1, fildes2, n_read, n_write;
15    char datos[BUFSZ];
16
17    if (argc != 3)
18    {
19       printf("Error\n␣Sintaxis:␣%s␣archivo␣%s␣en␣archivo␣%s\n", argv[0], argv[1], argv[2]);
20       exit(1);
21    }
22    if ((fildes1 = open (argv[1], O_RDONLY)) == -1)
23    {
24       perror("No␣se␣pudo␣abrir␣el␣archivo␣origen.");
25       exit(1);
26    }
27    if ((fildes2 = open (argv[2], O_CREAT|O_TRUNC|O_RDONLY, PERMS)) == -1)
28    {
29       perror("No␣se␣pudo␣abrir␣el␣archivo␣destino.");
30       exit(1);
31    }
32    do
33    {
34       if ((n_read = read(fildes1, datos, BUFSZ)) == -1)
35       {
36          perror("Error␣de␣lectura.");
37          exit(1);
38       }
39       if ((n_write = write(fildes2, datos, n_read)) == -1)
40       {
41          perror("Error␣de␣escritura.");
42          exit(1);
43       }
44    } while (n_read != 0);
45    close(fildes1);
46    close(fildes2);
47    return 0;
48 }
```

Si se compila y se ejecuta el programa `copiar.c` de la siguiente manera:

```
$ gcc copiar.c -o copiar
```

```
$ ./copiar archivo1 archivo2
Error de escritura.: Bad file descriptor
```

Se observa que no se produce error en la compilación, pero sí en la ejecución. Si en base a la respuesta del programa no es capaz de detectar los fallos de un programa, es necesario el uso de un depurador. Para poder depurar sobre el código fuente debe ser incluida la *Tabla de Símbolos Ampliada* en el archivo ejecutable, en el que se encuentran las equivalencias entre posiciones de memoria de variables y funciones, y los nombres que se les asignó en el programa. Esto se consigue usando el modificador `-g` de `gcc`:

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

Para entrar en el depurador se introduce lo siguiente:

```
$ gdb copiar
```

Tras un mensaje de presentación, se muestra el intérprete de órdenes del `gdb`. Desde aquí se pueden introducir las órdenes de depuración.

Se puede pedir ayuda en cualquier momento mediante `help` (véase Figura 1).

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

aliases --Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined — User-defined commands

Type “help” followed by a class name for a list of commands in that class.
Type “help” followed by command name for full documentation.
Command name abbreviations are allowed if unambiguous.
```

Figura 1: Solicitud de ayuda en `gdb`.

Para pedir información sobre las funciones que se utilizan se usa `info functions`. En primer lugar, aparece la función `main()` del programa, de la que `gdb` posee la información de depuración necesaria; el resto de funciones pertenecen a bibliotecas que fueron compiladas sin información de depuración, por lo que no se podrán depurar en forma de código fuente (véase Figura 2).

```
(gdb) info functions
All defined functions:

File copiar.c:
int main(int, char **);

Non-debugging symbols:
0x08048304  _init
0x0804832c  write
0x0804833c  close
0x0804834c  perror
0x0804835c  __libc_start_main
0x0804836c  printf
0x0804837c  open
0x0804838c  exit
0x0804839c  read
0x080483ac  _start
0x080483d0  call_gmon_start
0x080483f4  __do_global_dtors_aux
0x08048430  frama_dummy
0x080485ac  __do_global_ctors_aux
0x080485d0  _fini
```

Figura 2: Obtención de información sobre las funciones utilizadas por la aplicación en `gdb`.

Para ejecutar el programa, se usa la orden `run` (véase la Figura 3).

```
(gdb) run
Starting program: /root/p4/ejemplo/copiar
Error
 Sintaxis: /root/p4/ejemplo/copiar lineas fichero

Program exited with code 01.
```

Figura 3: Ejecutar un programa en `gdb`.

Para ejecutar correctamente el programa, se deben pasar los dos parámetros por la línea de órdenes, `archivo1` y `archivo2`, de la siguiente forma (véase Figura 4).

```
(gdb) set args archivo1 archivo2
```

Figura 4: Paso de parámetros por la línea de órdenes en `gdb`

Uno de los mecanismos más habituales en la depuración de código es el establecimiento de puntos de ruptura. Se trata de seleccionar una línea de código de tal forma que, cuando se vaya a procesar, se detenga la ejecución a la espera de nuevas órdenes. Si se desea, por ejemplo, detener la ejecución al principio de una función, se introduce `break` seguido del nombre de la función.

En el siguiente ejemplo se indica que se detenga la ejecución nada más comenzar la función `main()` (véase Figura 5(a)). Asimismo, se puede indicar explícitamente dónde se desea que se detenga la ejecución. Por ejemplo, en la línea número 41 (véase Figura 5(b)).

Se puede habilitar o deshabilitar un punto de ruptura mediante `enable` o `disable` respectivamente, seguidos del número de punto de ruptura correspondiente (véase Figura 6).

```
(gdb) break main
Breakpoint 1 at 0x804846f: file copiar.c, line 13.
```

(a) Punto de ruptura al comienzo de una función (`main()`) en `gdb`

```
(gdb) break 41
Breakpoint 2 at 0x804858b: file copiar.c, line 41.
```

(b) Punto de ruptura en una determinada línea en `gdb`

Figura 5: Establecimiento de puntos de ruptura en `gdb`

```
(gdb) break main
Breakpoint 1 at 0x804846f: file copiar.c, line 13.
```

Figura 6: Establecimiento de puntos de ruptura en `gdb`

Para borrar un punto de ruptura se emplea `delete` seguido del número de punto de ruptura (véase Figura 7). Además, se puede listar todos los puntos de ruptura configurados con `info breakpoints` (véase Figura 8).

```
(gdb) delete 2
```

Figura 7: Borrar punto de ruptura en `gdb`

Si se ejecuta de nuevo el programa (véase Figura 9). Como puede observarse, se queda el proceso detenido en la línea 13, donde se estableció el punto de ruptura. Para continuar la ejecución, se introduce `continue`.

Se puede avanzar por líneas del programa paso a paso mediante la orden `step`. Si se encontraran funciones que contasen con información de depuración, se entraría a ejecutar su código también paso a paso. Si no se desea este comportamiento, se debe usar la orden `next`. Con esta última orden, las funciones que se encuentran se ejecutarán como si se trataran de una única línea (véase Figura 10).

Para consultar el valor de una variable se usa la orden `print`, que admite como argumento una expresión con una sintaxis similar a la de la función `printf()` de C, pero sin paréntesis (véase Figura 10).

Se pueden aplicar conversiones *cast* para ver un valor con un formato concreto. Asimismo, se puede cambiar el valor de una variable.

Con la ayuda del depurador es sencillo ver qué línea o parte del código es la que está produciendo el error. Para que no se produzca el error, en este caso, se cambia en la línea 23 O_RDONLY por O_RDWR (véase Figura 11).

```
(gdb) info breakpoints
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0x0804846f in main at copiar.c:13
2   breakpoint     keep n   0x0804858b in main at copiar.c:41
```

Figura 8: Listar los puntos de ruptura configurados en `gdb`.

```
(gdb) run
Starting program: /root/p4/ejemplo/copiar archivo1 archivo2

Breakpoint 1, main (argc=3, argv=0xbffff934) at copiar.c:13
13              if(argc!=3)
```

Figura 9: Ejecución con puntos de ruptura en `gdb`

```
(gdb) next
18    if((fildes1=open(argv[1],O_RDONLY))==-1)
(gdb) next
23    if((fildes2=open(argv[2],O_CREAT|O_TRUNC|O_RDONLY,PERMS))==-1)
(gdb) next
30    if((n_read=read(fildes1,datos,BUFSZ))==-1)
(gdb) print fildes1
$3 = 5
(gdb) print fildes2
$4 = 6
(gdb) print datos
$5 = '\0' ...
(gdb) print n_read
$6 = 1108517584
(gdb) next
35    if((n_write=write(fildes2,datos,n_read))==-1)
(gdb) print datos
$7 = "Hola, este archivo es para probar el funcionamiento del
programa copiar.c\n\0..."
```

Figura 10: Consulta del valor de variables en `gdb`

Para salir del depurador se introduce la orden `quit`. Recompilando y volviendo al depurador hasta la línea en la que se produjo el error anteriormente (véase Figura 12).

```
(gdb) print n_read
$8 = 74
(gdb) next
37    perror("Error de escritura.");
(gdb) next
Error de escritura.: Bad file descriptor
38    exit(1);
```

Figura 11: Detectar un error en `gdb`

```
35    if((n_write=write(fildes2,datos,n_read))==-1)
(gdb) print datos
$7 = "Hola, este archivo es para probar el funcionamiento del
programa copiar.c\n\0..."
(gdb) print n_read
$8 = 74
(gdb) next
40    }while (n_read!=0);
(gdb) print n_write
$3 = 74
```

Figura 12: Consulta de variables tras corregir un error detectado en `gdb`

Se puede continuar depurando línea a línea o, si se desea, ejecutar directamente la aplicación. No es necesario en este último caso salir de `gdb`; simplemente se introduce la orden `run`, confirmando que la aplicación se ejecución correctamente (véase Figura 13).

```
(gdb) delete 1
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /root/p4/ejemplo/copiar archivo1 archivo2

Program exited normally
```

Figura 13: Ejecución correcta del programa en `gdb`

RESUMEN DE ÓRDENES DEL DEPURADOR gdb

1. **Llamada al depurador**

Forma de invocar gdb	Descripción
`gdb`	Entra en el depurador en modo interactivo.
`gdb programa`	Carga el programa y entra en modo interactivo. La ejecución del programa no comienza hasta que le sea indicado mediante una orden.

2. **Órdenes más frecuentemente utilizadas**

Orden	Descripción
`help [orden|grupo]`	Ayuda en línea de la orden o grupo de órdenes especificado.
`list [archivo:]función`	Muestra el código fuente (por defecto 10 líneas). Se puede indicar la parte del código a mostrar pasando como argumento el nombre del archivo donde está o el nombre de la función
`list [archivo:]línea[,línea]`	Muestra el código fuente pasando como argumento la línea central a mostrar o un rango de líneas.
`list`	Continúa el listado previo.
`list -`	Lista las líneas previas.
`break [archivo:]función`	Coloca un punto de ruptura (breakpoint) en el comienzo de la función.
`break [archivo:]línea`	Coloca un punto de ruptura en una línea específica.
`delete break [breakpoint]`	Elimina el breakpoint indicado o todos si no se indica uno específico.
`watch expresión`	Detiene la ejecución del programa cuando se modifica la expresión.
`set args list_argumentos`	Configura los argumentos a pasar al programa en la ejecución.
`show args`	Muestra los argumentos configurados para pasar al programa en la ejecución.
`run [argumentos]`	Comienza la ejecución del programa desde el principio. Los argumentos son áquellos pasados al programa o los definidos previamente.
`print expresión`	Muestra el valor de la expresión.
`printf ‘‘expresión’’`	Muestra el valor de la expresión con formato. Se utiliza del mismo modo que en el `printf` del lenguaje C.
`set variable=expresión`	Modifica el valor de la variable al resultado de la expresión.
`continue`	Continúa la ejecución del programa después de que ha sido detenido.
`next`	Ejecuta la siguiente línea del programa ejecutando el cuerpo de cada función llamada de forma completa.
`step`	Ejecuta la siguiente línea del programa ejecutando paso a paso las funciones.
`quit`	Finalizar `gdb`.