Taller Práctico GDB

Martita Muñoz (martitamunoz@udec.cl)

26 de agosto 2022

1. Introducción

GDB es una herramienta que permite depurar un código. Es posible ver qué ocurre con el programa durante el tiempo de ejecución, como el comportamiento de las instrucciones, uso de los registros, valores de las variables, entre otros detalles. Es muy útil si se desea encontrar un problema dentro del código durante la ejecución sin necesidad de imprimir mensajes durante la ejecución, ya sea un comportamiento incorrecto durante la ejecución o descubrir la causa de un error. GDB funciona en diversos lenguajes (Como Ada, Assembly, Fortran, Pascal, entre otros), y diversas plataformas (UNIX, Microsoft Windows, Mac OS X).

En este taller práctico usaremos GDB en el lenguaje C en un sistema Linux.

2. Comenzando con GDB

Vamos a trabajar con la función $factorial(n) = n!$ que multiplica todos los valores entre 1 y n.
Primero que todo, compile el archivo adjunto `factorial.c` en la terminal de linux:

```
> gcc factorial.c -o factorial
```

Vamos a calcular el resultado de $factorial(4)$ ejecutando la siguiente instrucción:

```
> ./factorial 4
```

Actividad:

- ¿Cuál es el resultado que obtiene? ¿Es correcto el resultado?
- Intente con otros valores de entrada. ¿Qué resultado obtiene?

A continuación, haremos la depuración de `factorial.c`. Primero tendremos que volver a compilar el código sando el flag `-g`:

```
> gcc factorial.c -o factorial -g
```

Posteriormente iniciamos GDB:

```
> gdb ./factorial
```

Se nos abrirá una nueva consola de comandos con un texto similar al siguiente:

```
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
```

```
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./factorial...
(gdb)
```

Actividad:

- Utilice la instrucción `list`. ¿Qué muestra?
- ¿Que muestra `list main`? ¿Y `list factorial`?
- ¿Que muestra `list 12,13`? ¿Y `list 1,16`?

Para comenzar a depurar nuestro programa, tendremos que colocar un punto de quiebre o breakpoint. Un breakpoint es una etiqueta que colocamos en una línea del código que indica en dónde detener la ejecución. Es útil para analizar el estado de ejecución en ciertos puntos críticos.

GDB nos permite colocar indicar el breackpoint utilizando el comando `break` o simplemente `b`

- Con `break <line>` (o `b <line>`) podemos indicar la línea en donde queremos colocar un breakpoint.
- Con `break <function>` (o `b <function>`) podemos indicar la función en donde queremos colocar un breakpoint. El breakpoint se creará al inicio de la función.
- Con `info break` puede listar los breakpoints creados. Cada break se encuentra numerado.
- Con `delete <n_break>` borramos un breakpoint creado.

Nota: si no coloca al menos un breakpoint y ejecuta el programa, este correrá sin detenerse hasta el final.

Actividad:

- Cree un breakpoint al inicio de la función `main` y de la función `factorial`. ¿De cuántas posibles maneras es posible crear dichos breakpoints?

Para iniciar la ejecución usamos la instrucción `run <lista de argumentos>` (o `r <lista de argumentos>`).

Actividad:

- Ejecute el programa con `run 4` (o `r 4`). Describa qué aparece en pantalla.

Si el breakpoint al inicio de la función `main` está creado, se debería detener al inicio de la función (línea 9). Para comenzar a revisar línea por línea, utilizamos el comando `next`, o simplemente `n`.

Actividad:

- Ejecute varias veces el comando `next` (o `n`). ¿Qué sucede? Continúe hasta llegar al final del programa.
- Elimine el breakpoint sobre la función `factorial`, vuelva a ejecutar el programa y avance línea por línea. ¿Existe alguna diferencia? ¿Cuál?

En caso de querer ingresar a revisar las instrucciones de una función que no contiene un breakpoint, utilizamos la instrucción `step` (o `s`).

Actividad:

- Vuelva a ejecutar el programa, esta vez ingresando a la función `factorial` utilizando el comando `step` (o `s`).

- Una vez dentro de la función, utilice el comando `cont` (o `c`) y describa lo que sucede. ¿Cuál es el uso de este comando?

Otros comandos útiles:

- `start <lista de argumentos>`: inicia la ejecución de un programa, pero se detiene automáticamente en la primera línea de la función `main`. Es útil cuando no existen breakpoints y queremos revisar el funcionamiento de un programa línea por línea.
- `where`: lista las funciones llamadas desde la más reciente hacia atrás. En nuestro ejemplo, si utilizamos este comando dentro de la función `factorial`, nos mostrará la función actual seguida del `main` (porque desde `main` llamamos a `factorial`). Es útil cuando estamos depurando un código que realiza muchas llamadas a funciones o para funciones recursivas.

3. Visualizar el contenido de las variables

Hasta el momento hemos visto cómo utilizar GDB, ejecutar un programa con este, hacer el seguimiento y observar las instrucciones que se ejecutan. Ahora es el turno de revisar el comportamiento de las variables: los valores que poseen en cada paso y su comportamiento. Para ello, disponemos de las siguientes instrucciones:

- `print <variable>` (o `p <variable>`): imprime el contenido de la variable en el la línea en que nos encontramos.
- `watch <variable>`: imprime el contenido de la variable cada vez que cambia desde el momento en que es utilizado este comando.
- `display <variable>`: imprime el contenido de la variable cada vez que cambiamos de línea el momento en que es utilizado este comando.

Actividad:

- ¿Qué variables son relevantes en el programa? ¿Cuál es la función de cada una?
- Utilizando cualquiera de las instrucciones presentadas en esta sección, revise el comportamiento de cada variable a lo largo de su ejecución. ¿En dónde se provoca la falla?
- Una vez que identifique el error, corríjalo en el código y vuelva a compilar y ejecutar. Para salir de la consola de GDB, utilice el comando `quit` o `q`.

4. Conclusión

GDB es un depurador de código que facilita la identificación de errores durante la ejecución de un código. En este tutorial vimos su funcionamiento sobre un código en C con el propósito de identificar un error en la ejecución. También vimos que es posible visualizar el funcionamiento del programa a nivel de lenguaje ensamblador.

Actividad:

- Cree una lista con todos los comandos que ha aprendido a lo largo de este tutorial. Esta lista le será útil para cuando utilice GDB para depurar sus propios códigos.

Si desea más información, visite los siguientes sitios:

- https://www.gnu.org/software/gdb/
- https://www.cs.cmu.edu/~gilpin/tutorial/