



**Facultad de Ingeniería en
Electricidad y Computación**

Programación de Sistemas

CCPG1051

Federico Domínguez, PhD.

Unidad 3 – Sesión 4: Depuración de errores con GDB

Agenda

Depuradores

GDB

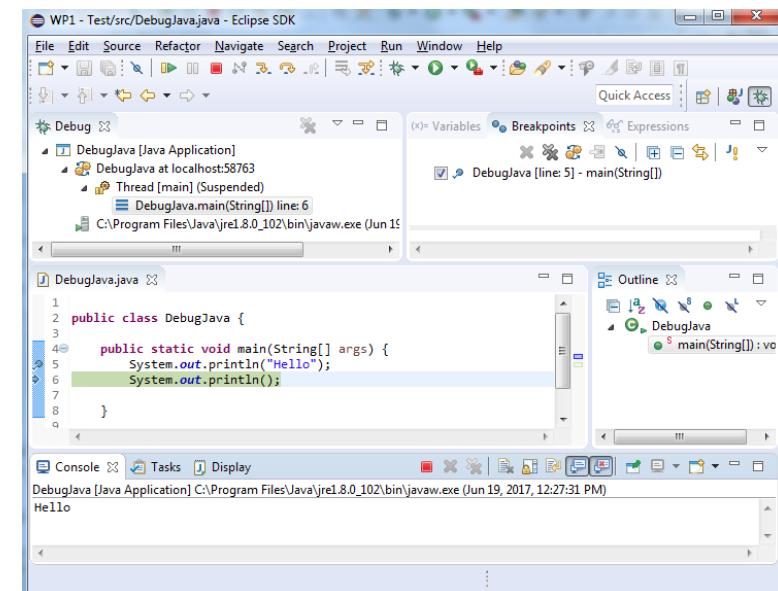


Herramientas de depuración

Debugging tool: Es un programa para depurar y probar otro programa.



Uno de los primeros “bugs” en computación era un insecto atrapado en los relays de la Harvard Mark II en 1947.



Una herramienta de depuración ejecuta un programa en un ambiente controlado en donde se puede detener y resumir la ejecución inspeccionando el estado de las variables y memoria.

GDB es la herramienta de depuración del proyecto GNU.

GNU Debugger (GDB)

Un depurador o “debugger” para varios lenguajes, entre ellos C y C++.

Permite la inspección del programa para saber que es lo que esta haciendo y como cambian las variables en tiempo de ejecución.

Errores como “*Segmentation Fault*” (“*Falla de Segmentación*”) son más fáciles de depurar usando *gdb*.

Para poder usar ***gdb***, es necesario compilar con la opción **-g**

Ejemplo:

Usualmente se compilaría *main.c* de la siguiente forma:

- `gcc -o ejecutable main.c`

Para poderlo depurar con *gdb*:

- `gcc -g -o ejecutable main.c`

La opción **-g** agrega símbolos adicionales al archivo ejecutable los cuales permiten a *gdb* conectar el ejecutable con el código fuente.

Finalmente, para depurar:

- `gdb ejecutable`

gdb contiene una consola interactiva similar a los shells *bash* y *sh*.

El *shell* o consola de *gdb* tiene conveniencias como navegación de historial con flechas y completar con tab.

Dentro de la consola de *gdb*, se puede invocar ayuda para cualquier comando:

```
(gdb) help [comando]
```

Para ejecutar el programa a depurar:

```
(gdb) run
```

Ejecución en la consola de *gdb*

Si el programa no tiene ningún problema, ejecuta normalmente dentro de *gdb*.

Si el programa contiene errores en tiempo de ejecución, *gdb* muestra información un poco más detallada sobre el error.

```
Program received signal SIGSEGV, Segmentation fault.  
0x0000000000400524 in sum_array_region (arr=0x7fffc902a270, r1=2, c1=5,  
r2=4, c2=6) at sum-array-region2.c:12
```

Usualmente, es útil detener el programa durante la ejecución e inspeccionar el estado del mismo.

Una forma de detener un programa en ejecución es definiendo *breakpoints* con el comando *break*:

- (gdb) break file1.c:6 *(Breakpoint en la línea 6 del archivo file1.c)*

El programa se detendrá en la línea 6 y permitirá el uso de otros comandos.

Se pueden definir tantos *breakpoints* como se deseen.

También se pueden definir *breakpoints* en funciones. Por ejemplo, si se tiene:

- int **my_func**(int a, char *b);

Entonces el comando:

- (gdb) break **my_func**

Detiene la ejecución al entrar a **my_func**.

Una vez detenida la ejecución, se puede inspeccionar el estado del programa.

Para continuar la ejecución y parar en el siguiente *breakpoint*:

- (gdb) continue

Para continuar la ejecución paso por paso:

- (gdb) step (Entre a las subrutinas o funciones)
 - (gdb) next (No entre a las funciones, las trata como una instrucción más)

Es común usar estos comando muchas veces seguidas, *gdb* permite ejecutar el comando anterior presionando solamente ENTER.

Para inspeccionar el estado del programa es necesario ver los valores de las variables:

- (gdb) print my_var (muestra el valor actual de la variable *my_var*)
 - (gdb) print/x my_var (muestra el valor actual de la variable *my_var* en hexadecimal)

Con el comando *watch*, se puede detener la ejecución cuando una variable específica cambia de valor.

Para detener la ejecución al cambiar el valor de *my_var*:

- (gdb) `watch my_var`

El programa se detendrá imprimiendo el valor anterior y el valor nuevo de *my_var*.

Si existen varias variables con el nombre *my_var*, GDB utiliza las reglas de alcance (*scope* de la variable) para determinar la variable a usar.

Otros comandos útiles

backtrace – produce una traza de las llamadas en la pila de la función que causó un “*segmentation fault*” (similar a la información mostrada con las excepciones de Java)

where – igual que *backtrace*, excepto que puede funcionar sin necesidad de que exista un error en tiempo de ejecución

finish – continua con la ejecución hasta que la función termine

delete – borra un *breakpoint*

info breakpoints – muestra información sobre todos los *breakpoints*

Demostración

Referencias

Tutorial:

- <https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf>

Libro guía:

- Computer Systems, Bryant y O'Hallaron. Sección 3.11