[Mini guía] Inicio rápido a GDB

Guía de inicio rápido a la depuración de errores de código escrito en C o C++ con GDB. Esta guía está pensada para ser seguida paso a paso al mismo tiempo que se ejecutan los comandos en GDB. El anexo 1 contiene algunos comandos útiles que no fueron explicados en las instrucciones principales. El anexo 2 contiene comandos extra para depurar código assembly, cuya lectura se sugiere al terminar el paso 5 del caso B.

Caso A: El programa explota

Decimos informalmente que explota cuando la ejecución termina abruptamente a causa de un error. La idea es correr el programa con GDB para que este nos ayude a encontrar el punto del código que genera el error.

1. **gdb** *nombreDelArchivoEjecutable*
2. **run** (más corto: **r**) para correr el programa.
3. **backtrace** (más corto: **bt**) para ver toda la jerarquía de llamados de funciones y así identificar la porción de código que explota.
4. **quit** (más corto: **q**) para salir de GDB y, si no logramos identificar el error, proceder al caso B.

Caso B: El programa funciona mal

Decimos que funciona mal cuando no funciona de la manera esperada (explote o no). La idea es poner puntos de interrupción (*breakpoints* en inglés) en el código y luego ir avanzando de a una línea por vez para ver cómo avanza la ejecución, al mismo tiempo que consultamos los valores de las variables con el fin último de identificar el problema.

1. Indicar al compilador de código C/C++ (GCC) la opción "-g" para que genere la información de depuración que utiliza GDB (sin esto no funcionan los puntos de interrupción). En caso de tener código assembly, también indicar al compilador NASM la opción "-g".
2. **gdb** *nombreDelArchivoEjecutable*
3. **break** *númeroDeLínea* (más corto: **b** *númeroDeLínea*) para indicar que la ejecución debe parar en la línea indicada del archivo principal. Se pueden poner muchos puntos de interrupción.
   Si se quiere un punto de interrupción en un archivo que no sea el principal (que no tenga la función main) hay que poner el nombre antes de indicar la línea, por ejemplo: break MiArchivo.h:112
4. **run** (más corto: **r**) para ejecutar el programa hasta terminar o hasta encontrar el primer punto de interrupción.
   start es similar a run pero deja el programa parado en la primera línea. Puede ser útil en caso de que el programa explote y queramos recorrerlo desde el comienzo.
5. Depurando:
   - **next** o **n** para ejecutar la línea actual y dejar la ejecución parada en la siguiente línea.
   - **step** o **s** es igual que next con la diferencia que si la línea actual es un llamado a función, step se mete adentro del cuerpo de la misma (next ejecuta todo el cuerpo de la función en un solo paso).
   - **continue** o **c** para saltar directamente al siguiente punto de interrupción (si hubiere) o al fin del programa.
   - **print** *nombreDeUnaVariable* o **p** *nombreDeUnaVariable* para ver el valor actual de la variable cuando la ejecución llega a un punto de interrupción.
   - **break** *X* o **b** *X* para ir metiendo más puntos de interrupción.
   - **watch** *condición* es un continue condicional; el programa se ejecuta sin parar mientras la condición sea falsa y queda parado cuando sea verdadera. Muy útil para ciclos, por ejemplo: watch i ¿ 60
6. **quit** (más corto: **q**) para salir de GDB.

Las instrucciones anteriores muestran algunos comandos que se pueden utilizar. Si durante una sesión de depuración de código les surge una necesidad que no fue cubierta en esta guía o se les ocurre algo que suene razonable o les gustaría poder hacer, posiblemente ya esté implementado. Utilizar un buscador web o tipear help desde la consola de GDB para consultar el manual. Para ver cómo se usa un comando, tipear **help** *nombreDeComando*.

Anexo 1: Otros comandos útiles

- **kill** (más corto: **k**) para matar la ejecución actual sin salir de GDB.
- **file** *nombreDelArchivoEjecutable* para seleccionar el programa a depurar, luego de haber hecho un kill o de haber entrado a GDB sin especificar nombre de archivo.
- **where** para ver el punto en que estamos parados durante una sesión de depuración (archivo, función, línea) por si nos perdimos.
- **break** *nombreDeFunción* para pausar la ejecución cada vez que aparezca un llamado a una función determinada.
- **break** ±*númeroDeLíneas* para indicar la línea del punto de interrupción con un desplazamiento desde la línea actual.
- **tbreak** es un punto de interrupción temporario que desaparece luego de ser alcanzado una vez. Acepta los mismos parámetros que el no temporario. La instrucción start mencionada anteriormente es equivalente a poner un tbreak en la primer línea del programa y luego ejecutar run.
- **finish** para avanzar directamente hasta el fin de la función actual. Si hicimos un step pero no queremos recorrer la función completa, en lugar de tener que meter muchos next podemos hacer finish.
- **delete** para eliminar todos los puntos de interrupción (incluye aquellos creados con break, tbreak y watch).
- **up** es similar a backtrace (explicado en el caso A) pero muestra un solo nivel por vez y va subiendo de nivel en ejecuciones sucesivas.

Anexo 2: Comandos para trabajar con Assembly

- **next instruction** (más corto: **ni**) equivalente a next para programas de assembly.
- **step instruction** (más corto: **si**) equivalente a step para programas de assembly.
- **break** *nombreDeEtiqueta* (más corto: **b** *nombreDeEtiqueta*) para pausar la ejecución cada vez que el programa llegue a la etiqueta indicada. Por ejemplo: break .ciclo
- **print** *$nombreDeRegistro* (más corto: **p** *$nombreDeRegistro*) para ver el contenido de un registro, por ejemplo: p $rax para ver el contenido del registro RAX.
- **info registers** (más corto: **ir**) para ver el contenido de los registros todos juntos (usar **info all-registers** para ver los XMMs y otros registros adicionales).
- **print printf("%s", $nombreDeRegistro)** (más corto: **p printf("%s", $nombreDeRegistro)**) para mostrar en pantalla el valor de un string con formato C (terminado en cero). Por ejemplo, si el registro RDI apunta a un string, podemos mostrarlo con **p printf("%s", $rdi)**
- ¿Cómo ver los valores de un struct de C desde código ASM? Ejemplo: Tenemos en el registro RDI un puntero a un struct persona. **print *(struct persona*) $rdi** muestra el struct y **print (*(struct persona*) $rdi).dni** muestra el DNI. También podemos ver structs anidados.