Laboratorio Sesión 04: Debugging

Objetivo

El objetivo de esta sesión de laboratorio es aprender a utilizar el depurador (debugger) de Linux: el **gdb**. Para ello deberéis corregir y depurar los errores de un programa escrito en C y ensamblador.

Documentación

La documentación de referencia para esta sesión es el manual de gdb:

- "*Debugging with GBD*".

Esta documentación puede encontrarse en la página web de la asignatura.

¿Cómo Funciona el gdb?

gdb es el depurador de GNU que proporciona numerosas utilidades para evaluar y analizar el comportamiento de programas en tiempo de ejecución. El uso de **gdb** permite estudiar y observar el comportamiento de un programa en ejecución, así como tener un cierto control respecto a su ejecución.

Antes de utilizar el depurardor puede ser útil obtener una versión en ensamblador del programa a depurar utilizando el comando Linux **objdump**.

La idea general para hacer funcionar gdb (y cualquier otro depurador) es activar un conjunto de *breakpoints* cercanos a los puntos que nos interesen en el código. Estos puntos pueden ser las primeras instrucciones de una subrutina o una determinada dirección de memoria. Cuando el programa pasa por uno de los *breakpoints*, el programa se para (antes de ejecutar la sentencia asociada al *breakpoint*) y devuelve control al usuario. Desde ese punto, podemos examinar el contenido de los registros y el contenido de la memoria. También podemos ejecutar el programa instrucción a instrucción, por grupos de instrucciones o justo hasta el siguiente *breakpoint*.

En la siguiente tabla mostramos algunos de los comandos de **gdb** para ejecutar programas, activar / desactivar *breakpoints* y algunas utilidades:

Comando	Efecto
quit	Salir de gdb.
run	Ejecutar el programa, los argumentos del programa se escribirían aquí.
kill	Parar el programa.
break sum	Activar un breakpoint al inicio de la rutina sum (también puede ser una etiqueta, o el main).
break 23	Activar un breakpoint en la linea 23 del codigo fuente (del fichero activo en ese momento).
break *0x80483c3	Activar un breakpoint en la dirección 0x80483c3.
delete 1	Eliminar el breakpoint 1.
delete	Eliminar todos los breakpoints.
stepi	Ejecutar una instrucción.
stepi 5	Ejecutar 5 instrucciones.
nexti	Idem que stepi, pero funciona entre subrutinas
continue	Ejecuta hasta el siguiente breakpoint.
finish	Ejecuta hasta que la rutina actual retorna.
info frame	Da información del bloque de activación actual
help	Nos da información general de gdb.
help *comando*	Nos da información específica del comando indicado.

En la siguiente tabla mostramos algunos de los comandos de **gdb** para examinar código y datos:

Comando	Efecto
disas	Desensambla la función actual.
disas sum	Desensambla la función sum.
disas 0x80483e8	Desensambla alrededor de la dirección especificada.
disas 0x80 0x90	Desensambla en el rango de direcciones especificado.
print /x $eax	Muestra el valor del registro especificado en hexadecimal.
print $eax	Muestra el valor del registro especificado en decimal.
print /t $eax	Muestra el valor del registro especificado en binario.
print 0x100	Muestra el número 0x100 en decimal.
print /x 456	Muestra el número 456 en hexadecimal.
print /x ($ebp +8)	Muestra el contenido de la dirección de memoria apuntada por ebp + 8 en hexadecimal.
print *(int *) 0xbfff890	Muestra el entero contenido en la dirección especificada.
print *(char *) ($ebx)	Muestra el caracter contenido en la dirección de memoria apuntada por ebx.
x/2w 0xbfff890	Examinar 8 bytes (2 longs) a partir de la dirección especificada.
x/2h ($ebp)	Examinar 4 bytes (2 words) a partir de la dirección apuntada por ebp.
x/20b sum	Examinar los primeros 20 bytes de la función sum (puede ser una etiqueta).
info registers	Muestra el valor de TODOS los registros.

Una sesión de gdb

Los pasos a seguir en una sesión con gdb serían los siguientes:

1) Compilar el programa:

```
$> gcc -gstabs+ Main.c Fusion.s -o FUSION
```

Notad que el programa puede estar compuesto de varios ficheros, algunos en C y otros en ensamblador. Si alguno de los programas tiene errores sintácticos no se generará ningún ejecutable. La opción -gstabs+ le indica al compilador que almacene información para el debugger.

2) Invocar el debugger:

```
$> gdb FUSION
(gdb)
```

3) Activar un breakpoint a la entrada del main

```
(gdb) break main
```

4) Ejecutar el programa hasta el primer breakpoint

```
(gdb) run
```

5) Aquí se pueden activar breakpoints en las funciones que queramos controlar:

```
(gdb) break Fusion
```

6) Ejecutar el programa hasta el siguiente breakpoint

```
(gdb) continue
```

7) A partir de aquí se puede examinar el contenido de los registros, ejecutar el programa instrucción a instrucción, etc.

Trabajo previo de la sesión

Como trabajo previo de la sesión hay que realizar las siguientes tareas:

1) Explica en detalle qué hacen las siguientes instrucciones. Si están sintácticamente mal escritas, indica el motivo:

```
movl $1, %eax
movl 1, %eax
movl $1, eax
movl 1, eax
```

2) Dada la siguiente secuencia de instrucciones:

```
cmpl %eax, %ebx
jge fin
```

¿en qué condiciones se efectua el salto?

3) Dada la siguiente secuencia de instrucciones:

```
cmpl %ebx, %eax
jge fin
```

¿en qué condiciones se efectua el salto?

4) Dibuja el bloque de activación de la rutina "fusion".

5) Explica con detalle para qué sirve la siguiente secuencia de instrucciones:

```
fusion: push %ebp
        movl %esp, %ebp
        pushl %ebx
        pushl %esi
        pushl %edi
```

Resultados de la sesión

Antes de acabar la sesión tenéis que haber hecho, por lo menos, lo siguiente:

- Compilar y ejecutar el programa en ensamblador que se ha utilizado como ejemplo.
- Eliminar todos los errores del programa, tanto sintácticos como semánticos. Sólo hay errores en la rutina en ensamblador. Los errores semánticos deberían eliminarse utilizando el debugger.
- La rutina Fusión mezcla dos vectores ordenados de menor a mayor para generar un vector ordenado de menor a mayor con el contenido de los dos vectores. Si hay tiempo se pueden programar alguna de estas alternativas:
  - mezclar, eliminando las repeticiones.
  - mezclar suponiendo que los vectores de entrada están ordenados de mayor a menor. Obviamente, el fichero resultado ha de quedar ordenado de mayor a menor.

Nota: Los ficheros para esta sesión los podéis encontrar en la página web de la asignatura: "`Programas.Sesion04.tar.gz`".