# Tutorial de GDB

Algoritmos y Estructuras de Datos II

# ¿Qué es gdb?

- "GNU Debugger"
- Es un depurador para varios lenguajes, incluyendo C y C++.
- Permite inspeccionar qué hace un programa en punto determinado de la ejecución.
- Errores como *segmentation faults* son más fáciles de encontrar con la ayuda de gdb.

# Compilando para depurar

El flag -g incorpora información necesaria para que gdb pueda operar:

```
$> gcc [otros flags] -g <archivos .c> -o dictionary
```

En el makefile, asegurarse de escribir el flag -g donde corresponda:

```
CFLAGS = -Wall -Werror -Wextra -pedantic -std=c99 -g
```

# ¿Cómo usar gdb?

Ejecutar gdb en una terminal:

```
$> gdb dictionary
```

Se obtiene un prompt como el siguiente:

```
(gdb)
```

También se puede cargar el ejecutable dentro de gdb:

```
(gdb) file dictionary
```

# Corriendo el programa

Para ejecutar el programa cargado, se utiliza el comando run:

```
(gdb) run
```

¿Y si hay bugs?

```
Program received signal SIGSEGV, Segmentation fault. 0x080493f0
in list_empty () at linked_list.c:26
```

Hasta ahora, no hay nada interesante.

# Breakpoints

Para insertar un punto de interrupción en el programa, se usa break:

```
(gdb) break linked_list.c : 24
Breakpoint 1 at 0x80493e6: file linked_list.c, line 24
```

Si el programa alcanza un breakpoint, gdb detiene la ejecución y nos pide otro comando. En este caso se detiene *justo antes* de ejecutar la línea 24 del archivo `linked_list.c`:

```
(gdb) run
Breakpoint 1, list_empty () at linked_list.c:24
(gdb)
```

# Otros tipos de breakpoints

La ejecución se interrumpe por cada llamada a la función indicada:

```
(gdb) break list_add
```

La ejecución se interrumpe bajo una condición dada:

```
(gdb) break linked_list.c : 24 if result == 0
```

Insertar un breakpoint algunas líneas adelante (o atrás):

```
(gdb) break +3
(gdb) break -1
```

# Otros tipos de breakpoints

La ejecución se interrumpe cada vez que la expresión dada cambia. Por ejemplo, con el siguiente comando, se indica que se debe interrumpir si cambia el valor de la variable `result`:

```
(gdb) watch result
```

La ejecución se interrumpe cada vez que la expresión dada es leída. Por ejemplo, interrumpir si se intenta leer el valor de la variable `result`:

```
(gdb) rwatch result
```

Interrumpe si se intenta leer o escribir una expresión dada:

```
(gdb) awatch result
```

# El comando print

Podemos mostrar el valor de las variables utilizando print:

```
(gdb) print list
(linked_list_t) 0x804d018
```

# El comando print

Es posible acceder a los punteros:

```
(gdb) print *list
{initial = 0x804d188, length = 1}
```

Tenemos acceso a los miembros de la estructura:

```
(gdb) print *list->initial->elem->data
{content = "mi definición", length = 13}
```

# El comando print

Es posible llamar a funciones:

```
(gdb) print list_length(list)
1
```

# Avanzando la ejecución

Ejecutar la siguiente línea de código con step:

```
(gdb) step
```

El comando next también ejecuta la siguiente línea de código:

```
(gdb) next
```

La diferencia está en las llamadas a función. El comando next considera una llamada a función como una única operación, en cambio step continúa la ejecución en la primera línea del cuerpo de la función.

# Avanzando la ejecución

Es posible indicar un número de pasos:

```
(gdb) step 4
```

Avanzar hasta llegar a un número de línea dado con until:

```
(gdb) until 29
```

# Avanzando la ejecución

Avanzar hasta el siguiente breakpoint:

```
(gdb) continue
```

Avanzar hasta finalizar la función actual:

```
(gdb) finish
```

# Revirtiendo la ejecución (gdb $\geq$ 7.0)

En algún punto podemos empezar a grabar la ejecución del programa:

```
(gdb) target record
```

Y luego podemos revertir pasos de ejecución, pero sólo podemos retroceder hasta el punto donde comenzamos a grabar:

```
(gdb) revert-step
(gdb) revert-continue
```

# Más comandos

Para ver la traza usamos where o backtrace:

```
(gdb) where
#0 index_from_string at index.c:36
#1 dict_exists at dict.c:62
#2 dict_add at dict.c:43
#3 main () at main.c:95
```

# Más comandos

Listar los breakpoints existentes:

```
(gdb) info break
Num Type Enb What
1 breakpoint y main at main.c:81
2 breakpoint y list_add at linked_list.c:68
```

Desabilitar temporalmente un breakpoint:

```
(gdb) disable breakpoints 1
```

Eliminar un breakpoint:

```
(gdb) delete 2
```

# Más comandos

Cambiar el valor de una variable:

```
(gdb) set var result = 0
```

Esto puede servir para chequear robustez del programa ante valores raros de variables.

# Y por último...

Mostrar ayuda sobre un comando de gdb:

```
(gdb) help <comando>
```

Salir de gdb:

```
(gdb) quit
```

# Para más información...

- Página oficial de gdb
- Manual de gdb
- Gdb en wikipedia