Sistemas Operativos

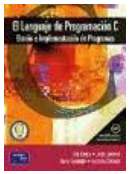
sesión 4: depuración de programas C

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

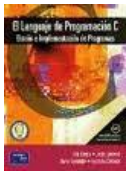
ARCOS @ UC3M
Alejandro Calderón Mateos

Contenidos



- Proceso de depuración
- Valgrind
- gdb

Contenidos



- **Proceso de depuración**
- Valgrind
- gdb

Motivación

- ¿Cómo puedo facilitarme la búsqueda de errores en el código?

```
acaldero@guernika:/infoso$ gcc -g -o e21 e21.c
acaldero@guernika:/infoso$ ./e21
Violación de segmento
…
```

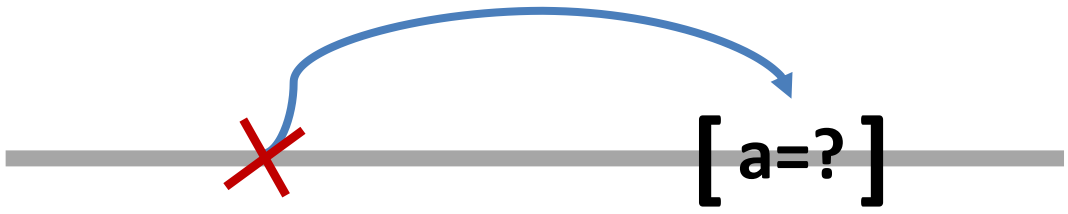
Depuración

- El proceso de depuración en general supone:

[ ] 1. Buscar en qué parte del código está el fallo.

a=? 2. Analizar el valor de las variables (estado) y ver qué valores son erróneos.

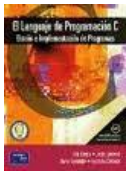
3. Buscar por qué se llega al estado erróneo.



Depuración

- Mecanismos típicos:
  - Uso de *printf* para imprimir valores y posición:
    - Ventaja: muy sencillo.
    - Desventaja: lento (recompilar+ejecutar muchas veces)
  - Uso de herramientas de depuración:
    - Ventajas: flexible.
    - Desventaja: hay que aprender su manejo.

Contenidos



- Proceso de depuración
- **Valgrind**
- gdb

valgrind

- Ejecución con valgrind (simple):

```
valgrind ./myprog arg1 arg2
```

- Ejecución típica:

```
valgrind --leak-check=full \
         --show-reachable=yes \
         ./myprog arg1 arg2
```

  - Detección de pérdidas de memoria y memoria malgastada (no liberada)

Valgrind: ejemplo

```
#include <stdlib.h>

void f ( void )
{
  int *x ;
  x = malloc(10*sizeof(int));
  x[10] = 0 ;
}

int main ( int argc,
           char *argv[] )
{
  f() ;
  return 0;
}
```

gcc –Wall –g –o e21 e21.c

./e21

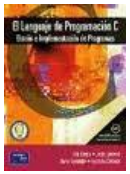
1. X[10] no reservado
2. falta free(x) ;

**valgrind **
**--leak-check=full **
**--show-reachable=yes **
./e21

Valgrind: ejemplo

```
==5303== Memcheck, a memory error detector
==5303== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==5303== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==5303== Command: ./e21
==5303==
==5303== Invalid write of size 4
==5303==    at 0x80483DF: f (e21.c:7)
==5303==    by 0x80483F1: main (e21.c:12)
==5303==  Address 0x41ae050 is 0 bytes after a block of size 40 alloc'd
==5303==    at 0x4026864: malloc (vg_replace_malloc.c:236)
==5303==    by 0x80483D5: f (e21.c:6)
==5303==    by 0x80483F1: main (e21.c:12)
==5303==
==5303== HEAP SUMMARY:
==5303==     in use at exit: 40 bytes in 1 blocks
==5303==   total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==5303==
==5303== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==5303==    at 0x4026864: malloc (vg_replace_malloc.c:236)
==5303==    by 0x80483D5: f (e21.c:6)
==5303==    by 0x80483F1: main (e21.c:12)
==5303==
==5303== LEAK SUMMARY:
==5303==    definitely lost: 40 bytes in 1 blocks
==5303==    indirectly lost: 0 bytes in 0 blocks
==5303==      possibly lost: 0 bytes in 0 blocks
==5303==    still reachable: 0 bytes in 0 blocks
==5303==         suppressed: 0 bytes in 0 blocks
==5303==
==5303== For counts of detected and suppressed errors, rerun with: -v
==5303== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 11 from 6)
```

Contenidos



- Proceso de depuración
- Valgrind
- **gdb**

gdb

- Inicio con gdb:
  ```
  gdb ./myprog
  ```
- Ejecución del programa:
  ```
  (gdb) run
  ```
- Finalizar la depuración:
  ```
  (gdb) quit
  ```

http://bulma.net/body.phtml?nIdNoticia=1805

gdb

- Argumentos (antes de ejecutar):

```
(gdb) set args [args. de myprog]
(gdb) show args
```

- Entorno (antes de ejecutar):

```
(gdb) set environment var[=valor]
(gdb) show environment [var]
```

gdb: parar la ejecución

- Un punto de ruptura (*breakpoint*) para el programa cuando se alcanza una localización:

```
(gdb) break [localización]
(gdb) info break
(gdb) del [identificador de break]
```

- La localización puede ser:
  - **Línea**
  - **NombreFichero:Línea**
  - **NombreDeFunción**
  - **NombreFichero:nombreFunción**
  - **Etc.**

gdb: parar la ejecución

- Un punto espía (*watchpoint*) para cuando el valor de una expresión cambia (en ámbito):

```
(gdb) watch expr
(gdb) info watch
(gdb) del [identificador de watchpoint]
```

- La expresión puede ser:
  - **Variable**
  - **Variable==valor**
  - **Etc.**

gdb: listar

- Un listado (*list*) permite visualizar el código fuente:
  `(gdb) list [posición]`

- La posición puede ser:
  - **NúmeroDeLínea**
  - **NombreDeFunción**
  - **<nada para continuar mostrando código>**
  - **Etc.**

gdb: examinar valores

- Se puede imprimir (*print*) el valor de una variable o expresión en este momento:

```
(gdb) print expr
```

- Se puede mostrar (*display*) el valor de una variable o expresión cada vez que pare la ejecución:

```
(gdb) display expr
```

- La expresión puede ser:
  - **Variable**
  - **Variable=valor**
  - **Etc.**

gdb: pila de llamadas

- Se puede imprimir (*bt*) la pila de llamadas:

```
(gdb) bt
```

- Se puede ir al contexto de la llamada anterior (*up*) y luego volver (*down*):

```
(gdb) up
(gdb) down
```

gdb: continuar la ejecución

- Continuar la ejecución hasta siguiente punto de ruptura (o fin del programa):

```
(gdb) continue
```

- Ejecutar la siguiente línea de código (entra dentro de funciones):

```
(gdb) step
```

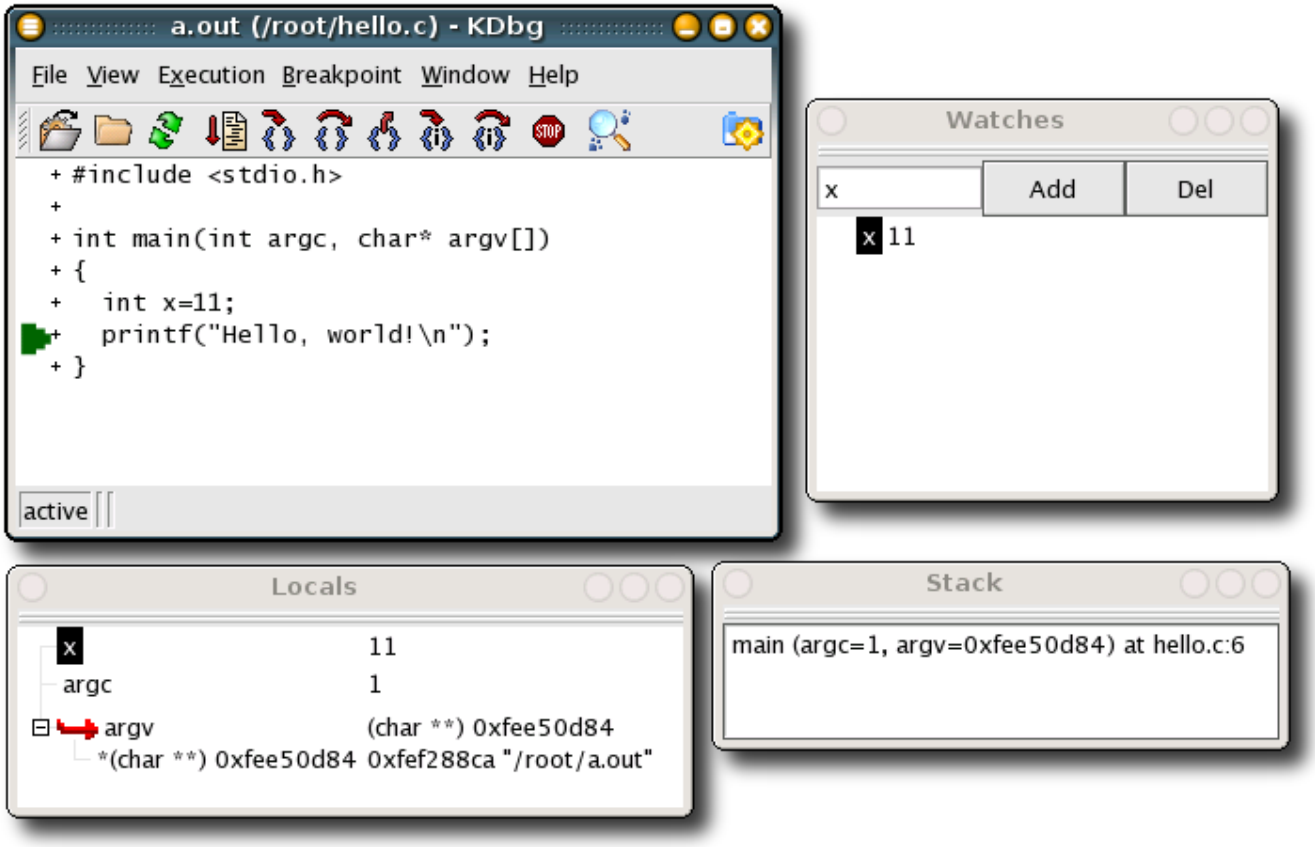
- Ejecutar la siguiente línea de código (**no** entra dentro de funciones):

```
(gdb) next
```

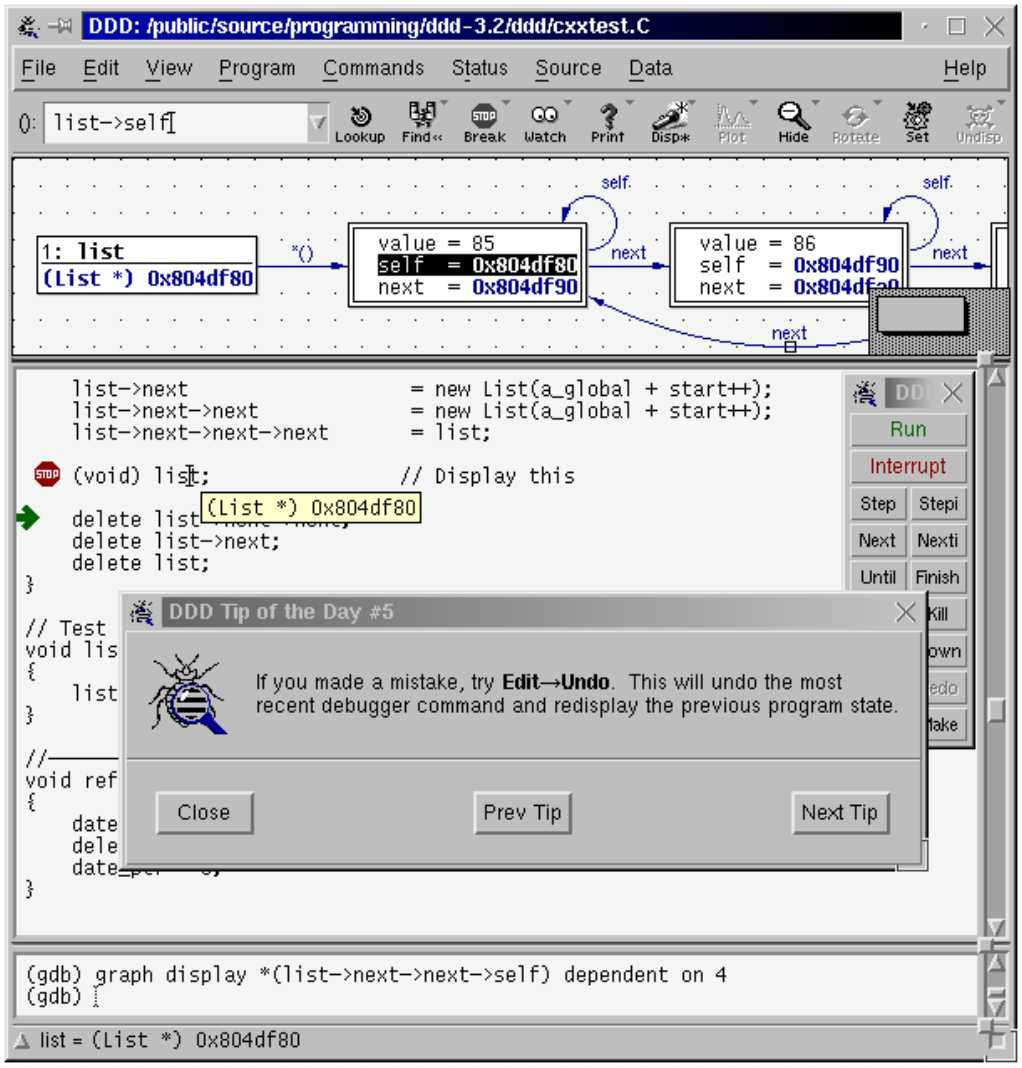
- Ejecutar hasta que termine la función actual:

```
(gdb) finish
```

kdbg


a.out (/root/hello.c) - KDbg
File View Execution Breakpoint Window Help
+ #include <stdio.h>
+
+ int main(int argc, char* argv[])
+ {
+ int x=11;
+ printf("Hello, world!\n");
+ }
active
Watches
x
Add
Del
x 11
Locals
x 11
argc 1
argv (char **) 0xfee50d84
*(char **) 0xfee50d84 0xfef288ca "/root/a.out"
Stack
main (argc=1, argv=0xfee50d84) at hello.c:6

ddd


DDD: /public/source/programming/ddd-3.2/ddd/cxxtest.C
File Edit View Program Commands Status Source Data Help
list->self
Lookup Find« Break Watch Print Disp* Plot Hide Rotate Set Undisp
1: list
(List *) 0x804df80
value = 85
self = 0x804df80
next = 0x804df90
value = 86
self = 0x804df90
next
list->next = new List(a_global + start++);
list->next->next = new List(a_global + start++);
list->next->next->next = list;
(void) list; // Display this
(List *) 0x804df80
delete list->next;
delete list;
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
DDD Tip of the Day #5
If you made a mistake, try Edit→Undo. This will undo the most recent debugger command and redisplay the previous program state.
Close
Prev Tip
Next Tip
(gdb) graph display *(list->next->next->self) dependent on 4
(gdb)
list = (List *) 0x804df80

Sistemas Operativos

sesión 4: depuración de programas C

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

ARCOS @ UC3M
Alejandro Calderón Mateos