

TALLER 5 – PROGRAMACIÓN DE SISTEMAS

ERRORES ENCONTRADOS

1.- Retorno de arreglo como variable local sin memoria inicializada

```
zero@zero-VirtualBox: ~/Descargas/taller5/taller5/src
Archivo Editar Ver Buscar Terminal Ayuda

zero@zero-VirtualBox:~/Descargas/taller5/taller5/src$ gdb a.out
GNU gdb (Ubuntu 8.1.0-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
Para las instrucciones de informe de errores, vea:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Leyendo símbolos desde a.out...hecho.
(gdb) r 5
Starting program: /home/zero/Descargas/taller5/taller5/src/a.out 5

Program received signal SIGSEGV, Segmentation fault.
0x0000555555554931 in mostrar_int (a=0x0) at taller5_bad.c:53
53      printf("%d\n", *(z+i));
(gdb) |
```

```
(gdb) r 5
Starting program: /home/zero/Descargas/taller5/taller5/src/a.out 5

Program received signal SIGSEGV, Segmentation fault.
0x0000555555554931 in mostrar_int (a=0x0) at taller5_bad.c:53
53      printf("%d\n", *(z+i));
(gdb) p z
$1 = 0x0
(gdb) p *z
No se puede acceder a la memoria en la dirección 0x0
(gdb) p *a
No se puede acceder a la memoria en la dirección 0x0
(gdb) p &z
$2 = (char **) 0x7fffffffdf38
(gdb) p &a
$3 = (int **) 0x7fffffffdf28
(gdb) |
```

```
(gdb) n
63      buf[i] = mul*i;
(gdb) n
62      for(i = 0; i < cant; i++){
(gdb) n
66      return &buf[0];
(gdb) p &buf
$1 = (int (*)[5]) 0x7fffffffdee0
(gdb) p &buf[0]
$2 = (int *) 0x7fffffffdee0
(gdb) p buf
$3 = {0, 2, 4, 6, 8}
(gdb) n
67      }
(gdb) n
main (argc=2, argv=0x7fffffffef078) at taller5_bad.c:102
102      est *res = NULL;
(gdb) p buf
$4 = 0x0
(gdb) |
```

El primer error se presenta al ejecutar la línea 53 donde nos retorna un segmentation fault que es parte del algoritmo de la función *mostrar_int()* en donde se pretende mostrar por pantalla cada contenido de la dirección que apunta el puntero tipo char Z, sin embargo se trata de una dirección de un bloque de memoria que no ha sido determinado por nosotros. Esta función recibe un arreglo de enteros, y retrocediendo en los pasos de la ejecución nos damos cuenta que este arreglo proviene de la función *fn1()* donde se realizan correctamente las operaciones para la formación de los elementos enteros, no obstante el arreglo de enteros *buf[]* que es una variable local no ha sido inicializado y no se le ha asignado un bloque de memoria. El presente problema se ha solucionado cambiando la declaración del arreglo *buf[]* por la inicialización de un puntero a entero **buf* asignándole con *malloc* un tamaño total de (4 X número de elementos(cant)) bytes.

```
Leyendo símbolos desde a.out...hecho.
(gdb) break 106
Punto de interrupción 1 at 0xb68: file taller5_bad.c, line 106.
(gdb) r 5
Starting program: /home/zero/Descargas/taller5/taller5/src/a.out 5
0
2
```

2.- Cantidad de Iteraciones menor a la esperada

```
Breakpoint 1, mostrar_int (a=0x555555757260) at taller5_bad.c:51
51      char *z = (char *)a;
(gdb) n
52      for(i = 0; i < sizeof(z)/sizeof(int); i++){
(gdb) p sizeof(z)
$1 = 8
(gdb) p sizeof(int)
$2 = 4
(gdb)
```

Una vez que se hizo posible la impresión de los enteros la cantidad de elementos mostradas por `mostrar_int()` no era la correcta que en este caso se trata de 5. Al momento de la iteración `for` se tuvo que modificar la condición de que la variable `i < sizeof(Z)/sizeof(int)` donde resultaría `i < 2` ya que el tamaño del puntero a `char Z` es 8 y el tipo `int` son 4 bytes. Simplemente se puede cambiar esto por la condición de que `i < cantidad de elementos`, es decir `i < cant` añadiendo esta variable `cantidad` como un nuevo argumento que debe recibir la función.

```
(gdb) r 5
Starting program: /home/zero/Descargas/taller5/taller5/src/a.out 5
0
2
4
6
8
Breakpoint 1, main (argc=2, argv=0x7fffffff078) at taller5_bad.c:106
106      int *arr2 = mas(cant);
(gdb)
```

3.- Retorno de arreglo como variable local sin memoria inicializada 2

```
Breakpoint 1, main (argc=2, argv=0x7fffffff078) at taller5_bad.c:106
106      int *arr2 = mas(cant);
(gdb) s
mas (cant=5) at taller5_bad.c:22
22      int *mas(int cant){
(gdb) n
24          int arr[cant];
(gdb) n
25          int *b = (int *)arr;
(gdb) p arr
$1 = {-8080, 32767, 3, 5, 0}
(gdb)
```

Nuevamente surge un `segmentation fault` en la línea 53 por el mismo motivo de un arreglo como variable local y sin bloques de memoria inicializados pero esta vez presente en la función `mas()`. Si se tratan de verificar los elementos de este arreglo se arroja una advertencia de que se trata una dirección a la que no se puede acceder. Esto se resuelve de la misma manera cambiando la declaración del arreglo `arr[]` por la inicialización de un puntero a entero `*arr` asignándole con `malloc` un tamaño total de (4 X número de elementos(`cant`)) bytes.

```
zero@zero-VirtualBox: ~/Descargas/taller5/taller5/src
Archivo Editar Ver Buscar Terminal Ayuda
2
4
5
3
Breakpoint 1, main (argc=2, argv=0x7fffffff078) at taller5_bad.c:107
107     mostrar_est(res);
(gdb) s
mostrar_est (a=0x7fffffffdf20) at taller5_bad.c:40
40     int i = 0;
(gdb) p res
El símbolo «res» no está en el contexto actual.
(gdb) p a
$1 = (est *) 0x7fffffffdf20
(gdb) p *a
$2 = {c = 1433760352, arr = 0x7fffffffdf20, cant_arr = 1433761664,
      mensaje = 0x555555757780 ""}
(gdb) n
41     for(i = 0; i < sizeof(a)/sizeof(est); i++){
(gdb) p sizeof(a)
$3 = 8
(gdb) p sizeof(est)
$4 = 32
(gdb)
```

4.- Cero iteraciones

```
(gdb) break 41
Punto de interrupción 1 at 0x8cd: file taller5_bad.c, line 41.
(gdb) r 5
Starting program: /home/zero/Descargas/taller5/taller5/src/a.out 5
0
2
4
6
8
Breakpoint 1, mostrar_est (cant=5, a=0x7fffffffdf20) at taller5_bad.c:41
41     for(i = 0; i < cant; i++){
(gdb)
```

```
Leyendo símbolos desde a.out...hecho.
(gdb) r 5
Starting program: /home/zero/Descargas/taller5/taller5/src/a.out 5
0
2
4
6
8
10
20
30
40
[Inferior 1 (process 2115) exited normally]
(gdb)
```

Dado que se han resuelto los anteriores problemas, por pantalla se mostraron los primeros y últimos elementos que se supone que debían mostrarse y el programa terminaba omitiéndose de esta manera los procesos centrales donde se trabaja con la estructura estTDA. Esto sucede específicamente en la función `mostrar_est()` donde la condición de la iteración $i < \text{sizeof}(z)/\text{sizeof}(\text{est})$ donde resulta $i < 8/32$ nunca se cumplirá si i empieza en 0 y avanza de uno en uno. Para arreglar esto se puede reemplazar dicha condición por algo más común: $i < \text{cantidad de elementos (cant)}$ siendo `cant` un nuevo argumento que puede recibir la función.

```
Breakpoint 1, main (argc=2, argv=0x7fffffff078) at taller5_bad.c:107
107     mostrar_est(res);
(gdb) p arr2
$1 = (int *) 0x555555757780
(gdb) p *arr2
$2 = 0
(gdb) p *(arr2+1)
$3 = 10
(gdb) p *(arr2+2)
$4 = 20
(gdb) p *(arr2+3)
$5 = 30
(gdb) p *(arr2+4)
$6 = 40
(gdb)
```

5.- Cambio De Dirección A La Que Apunta Un Puntero

```
Breakpoint 1, mostrar_est (cant=5, a=0x7fffffffdf20) at taller5_bad.c:41
41      for(i = 0; i < cant; i++){
(gdb) c
Continuando.

Program received signal SIGSEGV, Segmentation fault.
__strlen_avx2 () at ../sysdeps/x86_64/multiarch/strlen-avx2.S:62
62      ../sysdeps/x86_64/multiarch/strlen-avx2.S: No existe el archivo o el directorio.
(gdb)
```

```
zero@zero-VirtualBox: ~/Descargas/taller5/taller5/src
Archivo Editar Ver Buscar Terminal Ayuda
(gdb) n
30      b[i] = i * 10;
(gdb) n
29      for(i = 0; i < cant; i++){
(gdb) n
30      b[i] = i * 10;
(gdb) n
29      for(i = 0; i < cant; i++){
(gdb) n
30      b[i] = i * 10;
(gdb) n
29      for(i = 0; i < cant; i++){
(gdb) n
33      return arr;
(gdb) n
37      }
(gdb) n
main (argc=2, argv=0x7fffffffef078) at taller5_bad.c:107
107      mostrar_est(res);
(gdb) p arr2
$2 = (int *) 0x0
(gdb) p *arr2
No se puede acceder a la memoria en la dirección 0x0
(gdb)
```

Solucionado lo anterior se proceden a realizar las operaciones posteriores, sin embargo cuando se intenta mostrar la información almacenada por cada estudiante surge un nuevo error. Como la función `mostrar_est()` recibe un puntero de tipo `estTDA` que es generado por la función `fn2()` que recibe la cantidad de elementos y un puntero doble `est`, si se analiza esta función pretende crear un arreglo de estudiantes y asignar esto al puntero que se recibe como argumento, es decir que busca cambiar la dirección hacia donde apunta en este caso `**res`. En primer lugar se debe dar como argumento la dirección de memoria del puntero que sería `&res`.

Hay que verificar bien el tamaño del bloque de memoria que se desea crear, en este caso se debe hacer `malloc(sizeof(est)*cant)` y en la línea 86 al reedireccionar `res` se le asigna simplemente `data`:

`*res = data`, siendo este último un puntero esto arreglo de estudiantes.

```
leyendo símbolos desde a.out...hecho.
(gdb) break 70
Punto de interrupción 1 at 0xa4c: file taller5_bad.c, line 70.
(gdb) r 5
Starting program: /home/zero/Descargas/taller5/taller5/src/a.out 5

Breakpoint 1, fn2 (cant=5, res=0x7fffffffdf70) at taller5_bad.c:70
70      est *data = malloc(sizeof(est)*cant);
(gdb) n
72      if(data == NULL){
(gdb) Quit
(gdb)
```

```
Breakpoint 1, main (argc=2, argv=0x7fffffffef078) at taller5_bad.c:107
107      mostrar_est(cant, res);
(gdb) p res
$1 = (est *) 0x555555757280
(gdb) p *res
$2 = {c = 0, arr = 0x5555557572b0, cant_arr = 0,
      mensaje = 0x555555554c18 "Par"}
(gdb) p *(res+1)
$3 = {c = 1, arr = 0x5555557572d0, cant_arr = 1,
      mensaje = 0x555555554c1c "impar"}
(gdb) p *(res+2)
$4 = {c = 2, arr = 0x5555557572f0, cant_arr = 2,
      mensaje = 0x555555554c18 "Par"}
(gdb) p *(res+3)
$5 = {c = 3, arr = 0x555555757310, cant_arr = 3,
      mensaje = 0x555555554c1c "impar"}
(gdb) p *(res+4)
$6 = {c = 4, arr = 0x555555757330, cant_arr = 4,
      mensaje = 0x555555554c18 "Par"}
```

6.- Cantidad de elementos por estudiante

```
Archivo Editar Ver Buscar Terminal Ayuda
(gdb) r 5
Starting program: /home/zero/Descargas/taller5/taller5/src/a.out 5
0
2
4
6
8

Est c: 0, mensaje: Par
0

Est c: 1, mensaje: impar
0

Est c: 2, mensaje: Par
0
3

Est c: 3, mensaje: impar
0
3
6

Est c: 4, mensaje: Par
0
3
6
9
0
10
20
30
40

[Inferior 1 (process 2267) exited normally]
(gdb)
```

Finalmente, al mostrar cada estudiante nos damos cuenta que la cantidad de elementos que se deben mostrar es igual a la cantidad de elementos existentes, aunque suene lógico resulta ser la cantidad presente en la estructura estTDA como atributo cant_array. Al momento de llamar a mostrar_int() en la línea 43 dentro de la función mostrar_est() se debe enviar como argumento adicionalmente a (a+i)->cant_array.

```
8
Est c: 0, mensaje: Par
0
0
0
0
0

Est c: 1, mensaje: impar
0
0
0
0
0

Est c: 2, mensaje: Par
0
3
0
0
0

Est c: 3, mensaje: impar
0
3
6
0
0

Est c: 4, mensaje: Par
0
3
6
9
0
0
10
20
30
40
zero@zero-VirtualBox:~/Descargas/taller5/taller5/bin$
```

Con esto, el programa funciona de tal manera que cumple con todas las expectativas mencionadas en la premisa del presente trabajo.