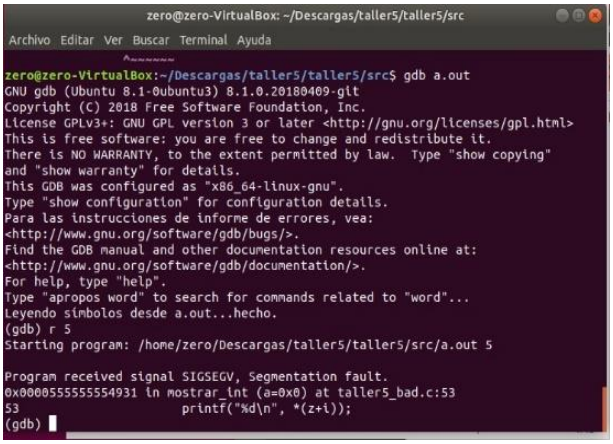
TALLER 5 – PROGRAMACIÓN DE SISTEMAS

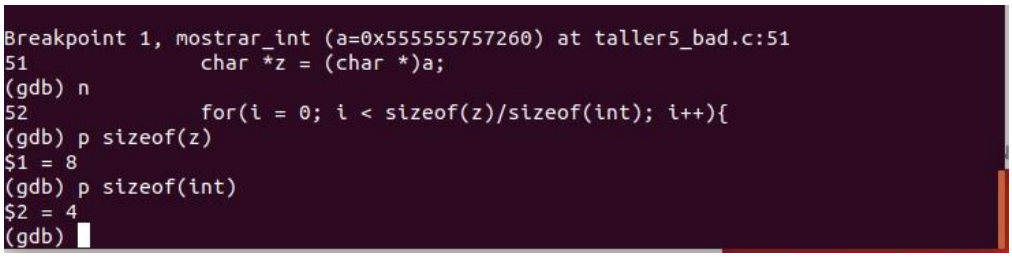
ERRORES ENCONTRADOS

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



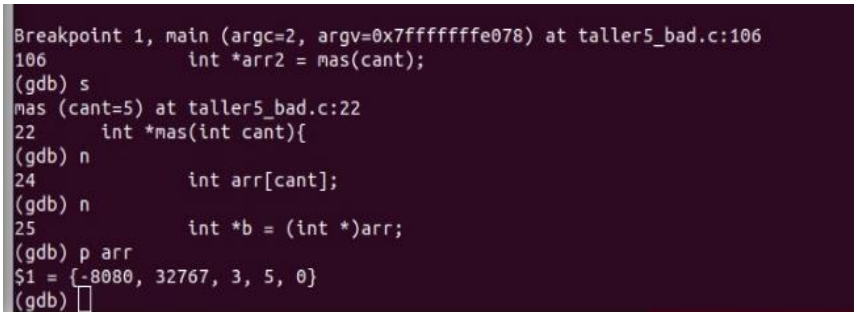
El primer error se presenta al ejecutar la línea 53 donde nos retorna un segmentation fault que es parte del algoritmo de la función *mostrar_int()* en donde se pretende mostrar por pantalla cada contenido de la dirección que apunta el puntero tipo char *Z*, sin embargo se trata de una dirección de un 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.

2.- Cantidad de Iteraciones menor a la esperada

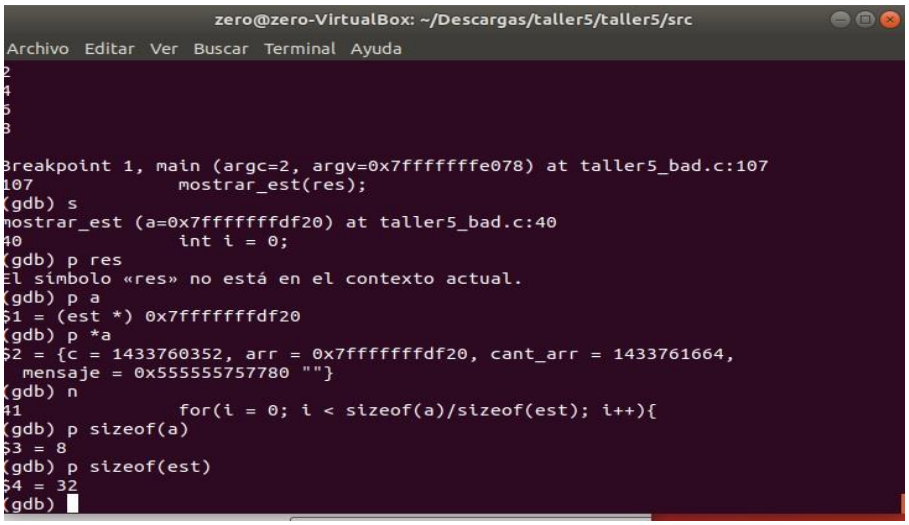


Una vez que se hizo posible la impresión de los enteros la cantidad de elementos mostradas por mostrar_int() no era la correcta que en este caso se trata de 5. Al momento de la iteración for se tuvo que modificar la condición de que la variable i < sizeof(Z)/sizeof(int) donde resultaría i<2 ya que el 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.

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

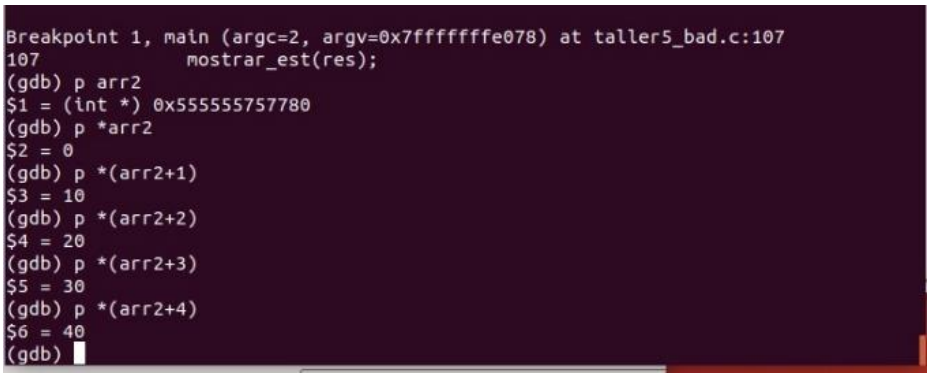


Nuevamente surge un segmentation fault en la línea 53 por el mismo motivo de un arreglo como variable local y sin bloques de memoria inicializados pero esta vez presente en la función mas(). Si se tratan de verificar los elementos de este arreglo se arroja una advertencia de que se trata una dirección 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.



4.- Cero iteraciones

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<sizeof(z)/sizeof(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<cantidad de elementos (cant) siendo cant un nuevo argumento que puede recibir la función.



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 dir
ectorio.
(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=0x7fffffffe078) 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)
```

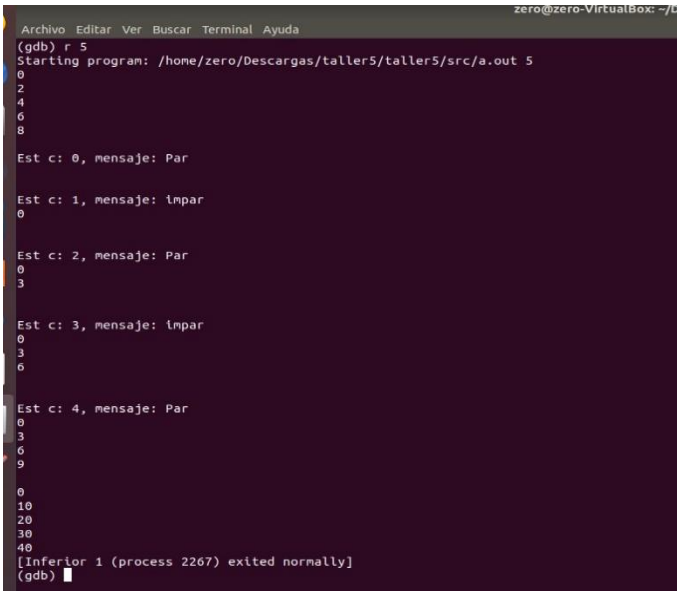
Solucionado lo anterior se proceden a realizar las operaciones posteriores, sin embargo cuando se intenta mostrar la información almacenada por cada estudiante surge un nuevo error. Como la funcion mostrar_est() recibe un puntero de tipo estTDA que es generado por la función fn2() que recibe la cantidad 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.

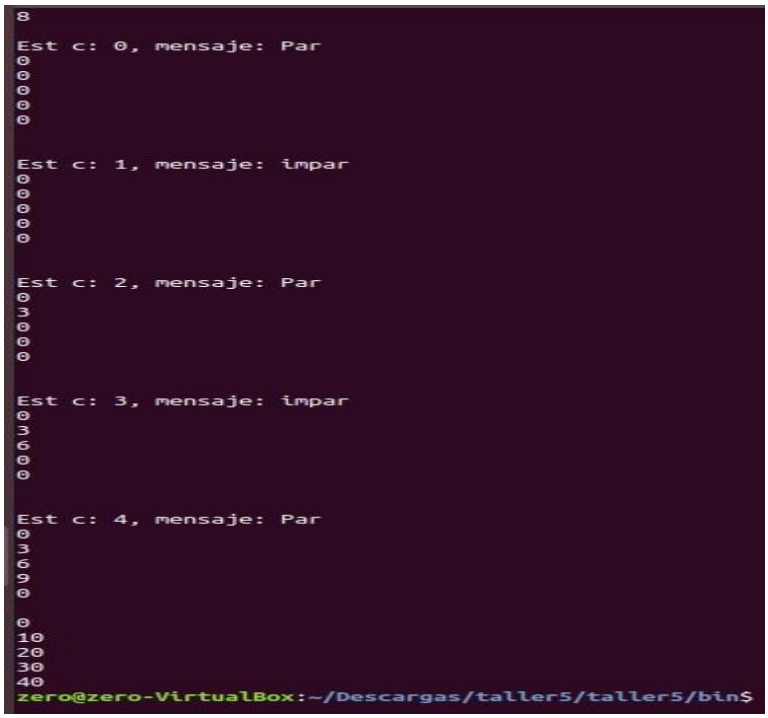
```
Leyendo símbolos desde a.out...hecho.
(gdb) break 70
Punto de interrupción 1 at 0xa4c: file taller5_bad.c, line 70.
(gdb) r 5
Starting program: /home/zero/Descargas/taller5/taller5/src/a.out 5

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) q
```

```
Breakpoint 1, main (argc=2, argv=0x7fffffffe078) 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



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.



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