

# Sistemas Operativos I Informe Problema 5

13 de mayo de 2020

**Autores y NIUB:**

Maria Isabel González Sánchez 20221913

Oriol Saguillo González 20150502

**Grupo:** C00

## Índice

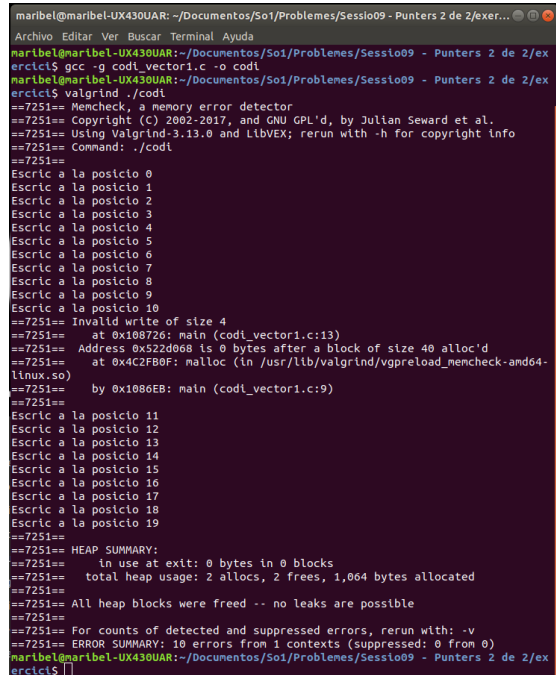
1. <code>codi_vector1</code>	3
2. <code>codi_vector2</code>	3
3. <code>codi_vector3</code>	4
4. <code>codi_vector4</code>	4
5. <code>codi_vector5</code>	5

## 1. codi\_vector1

Cuando en el bucle *for* se intenta escribir en la posición 10 del vector “a”, como es una posición más allá de las 10 asignadas (la posición 11) con la memoria dinámica, te devuelve que no puedes escribir ningún *int* más, por ello el *size* de 4 *bytes*.

Esto se debe a que con el *malloc* solo reservamos 40 *bytes* de memoria y ya no quedan *bytes* disponibles, especificándote “*Address 0x522d068 is 0 bytes after a block of size 40 alloc'd at 0x4C2FB0F: malloc*”. Por lo tanto, hemos excedido los límites de nuestro bloque y entramos en el *heap*.

Después intenta escribir fuera de nuestra zona reservada y al final, como el *for* imprime 20 veces en total y no funciona bien desde *i* = 10, el sumario de errores provocados por falta de espacio reservado mediante *malloc* será 10: “*ERROR SUMMARY: 10 errors from 1 contexts (suppressed: 0 from 0)*”.

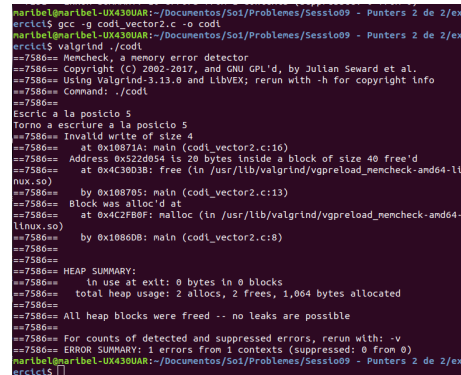


```
maribel@maribel-UX430UAR: ~/Documentos/SoI/Problemas/Sesio09 - Punters 2 de 2/exer...
Archivo Editar Ver Buscar Terminal Ayuda
maribel@maribel-UX430UAR:~/Documentos/SoI/Problemas/Sesio09 - Punters 2 de 2/ex
$ gcc -g codi_vector1.c -o codi
maribel@maribel-UX430UAR:~/Documentos/SoI/Problemas/Sesio09 - Punters 2 de 2/ex
$ valgrind ./codi
==7251== Memcheck, a memory error detector
==7251== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7251== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7251== Command: ./codi
Escrib a la posiclo 0
Escrib a la posiclo 1
Escrib a la posiclo 2
Escrib a la posiclo 3
Escrib a la posiclo 4
Escrib a la posiclo 5
Escrib a la posiclo 6
Escrib a la posiclo 7
Escrib a la posiclo 8
Escrib a la posiclo 9
Escrib a la posiclo 10
==7251== Invalid write of size 4
==7251== at 0x108726: main (codi_vector1.c:13)
==7251== Address 0x522d068 is 0 bytes after a block of size 40 alloc'd
==7251== at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-and64-
(linux.so)
==7251== by 0x1086EB: main (codi_vector1.c:9)
Escrib a la posiclo 11
Escrib a la posiclo 12
Escrib a la posiclo 13
Escrib a la posiclo 14
Escrib a la posiclo 15
Escrib a la posiclo 16
Escrib a la posiclo 17
Escrib a la posiclo 18
Escrib a la posiclo 19
==7251==
==7251== HEAP SUMMARY:
==7251== in use at exit: 0 bytes in 0 blocks
==7251== total heap usage: 2 allocs, 2 frees, 1,064 bytes allocated
==7251==
==7251== All heap blocks were freed -- no leaks are possible
==7251==
==7251== For counts of detected and suppressed errors, rerun with: -v
==7251== ERROR SUMMARY: 10 errors from 1 contexts (suppressed: 0 from 0)
maribel@maribel-UX430UAR:~/Documentos/SoI/Problemas/Sesio09 - Punters 2 de 2/ex
$
```

## 2. codi\_vector2

En el código 2, reservas 40 *bytes* de memoria dinámica mediante *malloc* y escribes el *int* 10 en la posición 5 del vector “a”. Luego, liberas dicha memoria con *free* e intentas volver a escribir en un vector ya liberado. Ahí reside el fallo del código.

Esto se puede ver en lo que nos devuelve el *valgrind*: “*Invalid write of size 4 at 0x10871A: main (codi\_vector2.c:16) Address 0x522d054 is 20 bytes inside a block of size 40 free'd at 0x4C30D3B: free*”. Cuando dice que es 20 *bytes* dentro de un bloque de 40 se refiere a la posición 5 del vector (4 *bytes* \* 5ª posición).



```
maribel@maribel-UX430UAR:~/Documentos/SoI/Problemas/Sesio09 - Punters 2 de 2/ex
$ gcc -g codi_vector2.c -o codi
maribel@maribel-UX430UAR:~/Documentos/SoI/Problemas/Sesio09 - Punters 2 de 2/ex
$ valgrind ./codi
==7586== Memcheck, a memory error detector
==7586== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7586== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7586== Command: ./codi
Escrib a la posiclo 5
Torno a escriure a la posiclo 5
==7586== Invalid write of size 4
==7586== at 0x10871A: main (codi_vector2.c:16)
==7586== Address 0x522d054 is 20 bytes inside a block of size 40 free'd
==7586== at 0x4C30D3B: free (in /usr/lib/valgrind/vgpreload_memcheck-and64-ll
(linux.so)
==7586== by 0x108705: main (codi_vector2.c:13)
==7586== Block was alloc'd at
==7586== at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-and64-
(linux.so)
==7586== by 0x10860B: main (codi_vector2.c:8)
==7586==
==7586== HEAP SUMMARY:
==7586== in use at exit: 0 bytes in 0 blocks
==7586== total heap usage: 2 allocs, 2 frees, 1,064 bytes allocated
==7586==
==7586== All heap blocks were freed -- no leaks are possible
==7586==
==7586== For counts of detected and suppressed errors, rerun with: -v
==7586== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
maribel@maribel-UX430UAR:~/Documentos/SoI/Problemas/Sesio09 - Punters 2 de 2/ex
$
```

### 3. codi\_vector3

```
maribel@maribel-UX430UAR: ~/Documentos/Sol/Problemas/Sesio09 - Punters 2 de 2/exer...
Archivo Editar Ver Buscar Terminal Ayuda
maribel@maribel-UX430UAR:~/Documentos/Sol/Problemas/Sesio09 - Punters 2 de 2/exer...
ercic1$ gcc -g codi_vector3.c -o codi
maribel@maribel-UX430UAR:~/Documentos/Sol/Problemas/Sesio09 - Punters 2 de 2/exer...
ercic1$ valgrind ./codi
==7636== Memcheck, a memory error detector
==7636== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7636== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7636== Command: ./codi
==7636==
==7636== Conditional jump or move depends on uninitialised value(s)
==7636== at 0x4E988DA: vfprintf (vfprintf.c:1642)
==7636== by 0x4EA0F25: printf (printf.c:33)
==7636== by 0x10870C: main (codi_vector3.c:10)
==7636==
==7636== Use of uninitialised value of size 8
==7636== at 0x4E9486B: _ltoa_word (_ltoa.c:179)
==7636== by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==7636== by 0x4EA0F25: printf (printf.c:33)
==7636== by 0x10870C: main (codi_vector3.c:10)
==7636==
==7636== Conditional jump or move depends on uninitialised value(s)
==7636== at 0x4E94875: _ltoa_word (_ltoa.c:179)
==7636== by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==7636== by 0x4EA0F25: printf (printf.c:33)
==7636== by 0x10870C: main (codi_vector3.c:10)
==7636==
==7636== Conditional jump or move depends on uninitialised value(s)
==7636== at 0x4E98814: vfprintf (vfprintf.c:1642)
==7636== by 0x4EA0F25: printf (printf.c:33)
==7636== by 0x10870C: main (codi_vector3.c:10)
==7636==
==7636== Conditional jump or move depends on uninitialised value(s)
==7636== at 0x4E9884C: vfprintf (vfprintf.c:1642)
==7636== by 0x4EA0F25: printf (printf.c:33)
==7636== by 0x10870C: main (codi_vector3.c:10)
==7636==
==7636== Valor d'a[5]: 0
==7636==
==7636== HEAP SUMMARY:
==7636== in use at exit: 0 bytes in 0 blocks
==7636== total heap usage: 2 allocs, 2 frees, 1,064 bytes allocated
==7636==
==7636== All heap blocks were freed -- no leaks are possible
==7636==
==7636== For counts of detected and suppressed errors, rerun with: -v
==7636== Use --track-origins=yes to see where uninitialised values come from
==7636== ERROR SUMMARY: 5 errors from 5 contexts (suppressed: 0 from 0)
maribel@maribel-UX430UAR:~/Documentos/Sol/Problemas/Sesio09 - Punters 2 de 2/exer...
ercic1$
```

Este código crea un vector dinámico, reservando 40 *bytes* de memoria y avanza así:  $a[i] = *(a[0]+i)$ . Sin embargo, al usar *malloc*, las distintas posiciones del vector no se inicializan a 0, como con *calloc*.

Entonces cuando intenta llegar a la posición 5, hace un salto incondicional debido a que no hay ningún valor inicializado: “*Conditional jump or move depends on uninitialised value(s) at 0x4E988DA: vfprintf (vfprintf.c:1642) by 0x4EA0F25: printf (printf.c:33) by 0x10870C: main (codi\_vector3.c:10)*”.

De ahí que los errores sean 5, ya que hasta llegar a la posición que queremos pasa por las posiciones no inicializadas 0, 1, 2, 3 y 4: “*ERROR SUMMARY: 5 errors from 5 contexts (suppressed: 0 from 0)*”.

### 4. codi\_vector4

El código 4 ejecuta sin problemas, pero *valgrind* notifica que ha habido una pérdida definitiva de un bloque de memoria dinámica de 40 *bytes*: “*HEAP SUMMARY: in use at exit: 40 bytes in 1 blocks*”.

Esto se debe a que no se ha liberado dicha memoria reservada mediante un *free* al acabar el programa. Es decir, no hemos regresado la memoria dinámica que reservamos usando *malloc*.

```
maribel@maribel-UX430UAR:~/Documentos/Sol/Problemas/Sesio09 - Punters 2 de 2/exer...
ercic1$ gcc -g codi_vector4.c -o codi
maribel@maribel-UX430UAR:~/Documentos/Sol/Problemas/Sesio09 - Punters 2 de 2/exer...
ercic1$ valgrind ./codi
==7671== Memcheck, a memory error detector
==7671== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7671== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7671== Command: ./codi
==7671==
==7671== Valor d'a[5]: 10
==7671==
==7671== HEAP SUMMARY:
==7671== in use at exit: 40 bytes in 1 blocks
==7671== total heap usage: 2 allocs, 1 frees, 1,064 bytes allocated
==7671==
==7671== LEAK SUMMARY:
==7671== definitely lost: 40 bytes in 1 blocks
==7671== indirectly lost: 0 bytes in 0 blocks
==7671== possibly lost: 0 bytes in 0 blocks
==7671== still reachable: 0 bytes in 0 blocks
==7671== suppressed: 0 bytes in 0 blocks
==7671== Rerun with --leak-check=full to see details of leaked memory
==7671==
==7671== For counts of detected and suppressed errors, rerun with: -v
==7671== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
maribel@maribel-UX430UAR:~/Documentos/Sol/Problemas/Sesio09 - Punters 2 de 2/exer...
ercic1$
```

## 5. codi\_vector5

En este último código, ha diferencia de los 4 anteriores, trabajamos con memoria estática. Esto quiere decir que, al no reservar memoria dinámica, no tendremos que hacer ningún *free* en el programa y que el lenguaje C no realiza ninguna comprobación de si estamos utilizando espacio fuera del que tenemos asignado (C no se queja). Esto conlleva sus riesgos.

Usando *valgrind* no se muestra ningún error aunque en verdad los estemos cometiendo. Un ejemplo claro es este código. Primero, cogemos un vector de tamaño 10, pero asignamos 2 *ints* a las posiciones 100 y 1000 respectivamente (fuera de nuestro espacio asignado). Estos se imprimirán bien, pero porque esas posiciones sí existen en la memoria estática, teniendo un contenido posiblemente aleatorio. Esto se puede ver si realizamos el siguiente trozo de código:

```
for(i = 0; i < 101; i++) {  
    printf("a[%d]=%d", i, a[i]);  
}
```

Se imprimirán valores como por ejemplo:

a[98]= -16776362

a[99]= 31

a[100]= 1234

Trabajar con memoria estática será peligroso si escogemos una posición fuera de nuestro espacio y en ella ya haya implementado código, ya que este se sobrescribirá, pudiendo generar un fallo masivo del programa.

```
maribel@maribel-UX430UAR:~/Documentos/So1/Problemes/Sessio09 - Punters 2 de 2/exercici$ gcc -g codi_vector5.c -o codi
maribel@maribel-UX430UAR:~/Documentos/So1/Problemes/Sessio09 - Punters 2 de 2/exercici$ valgrind ./codi
==7720== Memcheck, a memory error detector
==7720== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7720== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7720== Command: ./codi
==7720==
Faig assignacions
a[100] = 1234
a[1000] = 4321
Surto del main
==7720==
==7720== HEAP SUMMARY:
==7720==    in use at exit: 0 bytes in 0 blocks
==7720==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==7720==
==7720== All heap blocks were freed -- no leaks are possible
==7720==
==7720== For counts of detected and suppressed errors, rerun with: -v
==7720== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
maribel@maribel-UX430UAR:~/Documentos/So1/Problemes/Sessio09 - Punters 2 de 2/exercici$
```