UNIVERSITAT DE BARCELONA

Teorico-pràctic 10: Valgrind

Noah Márquez Vara
Alejandro Guzman Requena

19 maig 2022

1 INTRODUCCIÓ

1 Objectius de la sessió

En aquesta última sessió teorico-pràctica entregable realitzarem diverses proves amb codis proporcionats pel professorat per a comprovar, amb ajuda de programari, diversos errors d'assignació i accés a memòria que trobem.

Com s'ha explicat a classe el llenguatge C no realitza, en temps d'execució, comprovacions sobre si l'accés que es fa és vàlid. Té l'avantatge que les aplicacions en C acostumen a ser més ràpides que les equivalents en altres llenguatges. Hi ha l'inconvenient que és més difícil rastrejar els possibles problemes d'accés a memòria que tingui la nostra aplicació. Per tant la tasca que se'ns demana és rastrejar aquests problemes mitjançant l'aplicació *Valgrind*; aquesta aplicació implementa la seva pròpia versió de *malloc* cosa que permet detectar els problemes d'accés a memòria i vàries coses més.

2 PROVES REALITZADES

1 codi_vector1.c

```
==3346== ERROR SUMMARY: 10 errors from 1 contexts (suppressed: 0 from 0)
==3346==
==3346== 10 errors in context 1 of 1:
==3346== Invalid write of size 4
==3346==    at 0x4005E1: main (codi_vector1.c:13)
==3346==  Address 0x5213068 is 0 bytes after a block of size 40 alloc'd
==3346==    at 0x4C312EF: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==3346==    by 0x4005A8: main (codi_vector1.c:9)
```

Figura 2.1: *Error summary codi_vector1.c*

Invalid write significa que el nostre nostre programa intenta escriure dades en una zona de memòria on no hauria de fer-ho.

Però *Valgrind* ens diu molt més que això. Primer ens indica la mida de les dades escrites, que és de *4 bytes*. A més, ens indica la línia *0x4005E1: main(codi_vector1.c:13* on s'ha produït l'error.

La línia 13 correspon a *a[i] = i* de dins del bucle on escrivim a una certa posició.

A la línia *Address 0x5213068 is 0 bytes after a block of size 40 alloc'd,* també ens indica que l'adreça no vàlida es troba just després d'un bloc de *40 bytes* assignat. Això vol dir que s'ha assignat una zona de memòria de mida *40 bytes* (per tant, probablement 40 caràcters), però hem intentat escriure un *byte* nº 41.

2 codi_vector2.c

Com en el codi anterior, obtenim un error d'*Invalid write*, que com hem comentat vol dir que el programa intenta escriure dades en una zona de memòria on no ho hauria de fer (línia 16: *a[5] = 10*).

El següent error (*Address 0x5213054 is 20 bytes inside a block of size 40 free'd*) ens indica que estem intentant accedir a un bloc de *20 bytes* dins d'un àrea que s'ha alliberat (utilitzant la funció *free* a la línia 13). Definitivament, això és un problema i pot provocar que el nostre codi es bloquegi.

```
==3338== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
==3338==
==3338== 1 errors in context 1 of 1:
==3338== Invalid write of size 4
==3338==    at 0x4005E3: main (codi_vector2.c:16)
==3338==  Address 0x5213054 is 20 bytes inside a block of size 40 free'd
==3338==    at 0x4C3251B: free (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==3338==    by 0x4005D0: main (codi_vector2.c:13)
==3338==  Block was alloc'd at
==3338==    at 0x4C312EF: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==3338==    by 0x4005A8: main (codi_vector2.c:8)
```

Figura 2.2: *Error summary codi_vector2.c*

3 codi_vector3.c

```
==3379== ERROR SUMMARY: 5 errors from 5 contexts (suppressed: 0 from 0)
==3379==
==3379== 1 errors in context 1 of 5:
==3379== Conditional jump or move depends on uninitialised value(s)
==3379==    at 0x4EA8244: __vfprintf_internal (in /lib64/libc-2.31.so)
==3379==    by 0x4E93CC7: printf (in /lib64/libc-2.31.so)
==3379==    by 0x4005C7: main (codi_vector3.c:10)
==3379==
==3379==
==3379== 1 errors in context 2 of 5:
==3379== Conditional jump or move depends on uninitialised value(s)
==3379==    at 0x4EA719F: __vfprintf_internal (in /lib64/libc-2.31.so)
==3379==    by 0x4E93CC7: printf (in /lib64/libc-2.31.so)
==3379==    by 0x4005C7: main (codi_vector3.c:10)
==3379==
==3379==
==3379== 1 errors in context 3 of 5:
==3379== Conditional jump or move depends on uninitialised value(s)
==3379==    at 0x4E8E035: _itoa_word (in /lib64/libc-2.31.so)
==3379==    by 0x4EA70E7: __vfprintf_internal (in /lib64/libc-2.31.so)
==3379==    by 0x4E93CC7: printf (in /lib64/libc-2.31.so)
==3379==    by 0x4005C7: main (codi_vector3.c:10)
==3379==
==3379==
==3379== 1 errors in context 4 of 5:
==3379== Use of uninitialised value of size 8
==3379==    at 0x4E8E02B: _itoa_word (in /lib64/libc-2.31.so)
==3379==    by 0x4EA70E7: __vfprintf_internal (in /lib64/libc-2.31.so)
==3379==    by 0x4E93CC7: printf (in /lib64/libc-2.31.so)
==3379==    by 0x4005C7: main (codi_vector3.c:10)
==3379==
==3379==
==3379== 1 errors in context 5 of 5:
==3379== Conditional jump or move depends on uninitialised value(s)
==3379==    at 0x4EA7DFA: __vfprintf_internal (in /lib64/libc-2.31.so)
==3379==    by 0x4E93CC7: printf (in /lib64/libc-2.31.so)
==3379==    by 0x4005C7: main (codi_vector3.c:10)
```

Figura 2.3: *Error summary codi_vector3.c*

L'error de *Conditional jump or move depends on uninitialised value(s)* es produeix si ens oblidem d'inicialitzar les variables abans d'utilitzar-les o accedir-hi.

A la imatge no s'ha fet servir, però es podria utilitzar el *flag '–track-origins=yes'* per veure d'on prové el valor no inicialitzat. Si l'executem ens diu que prové de la línia 8 (*a = malloc(10 * sizeof(int));*.

Posteriorment a la línia 10 intentem utilitzar el valor que no hem inicialitzat (*Use of uninitialised value of size 8*).

4 codi_vector4.c

```
==4469== HEAP SUMMARY:
==4469==     in use at exit: 40 bytes in 1 blocks
==4469==   total heap usage: 2 allocs, 1 frees, 1,064 bytes allocated
==4469==
==4469== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==4469==    at 0x4C312EF: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==4469==    by 0x400558: main (codi_vector4.c:8)
==4469==
==4469== LEAK SUMMARY:
==4469==    definitely lost: 40 bytes in 1 blocks
==4469==    indirectly lost: 0 bytes in 0 blocks
==4469==      possibly lost: 0 bytes in 0 blocks
==4469==    still reachable: 0 bytes in 0 blocks
==4469==         suppressed: 0 bytes in 0 blocks
==4469==
==4469== For lists of detected and suppressed errors, rerun with: -s
==4469== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Figura 2.4: *Heap summary codi_vector4.c*

En aquest codi el problema amb el que es troba el sistema no és d'accés a memòria o assignació a una regió que no es pot accedir. En aquest cas s'inicialitza un vector dinàmic amb una memòria determinada (40 bytes) i el missatge més significatiu que mostra el programari *Valgrind* a l'executar el codi és *definitely lost: 40 bytes in 1 blocks*. Amb aquest missatge es refereix a que, després de l'execució del codi i utilitzar el vector dinàmic prèviament inicialitzat, no s'allibera la memòria que s'ha assignat a aquest punter. Això es necessari per a que altres processos (després de l'execució d'aquest) puguin utilitzar la memòria que aquest ha utilitzat i que, en la seva finalització, ha deixat de fer servir.

5 codi_vector5.c

```
==4489== HEAP SUMMARY:
==4489==     in use at exit: 0 bytes in 0 blocks
==4489==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==4489==
==4489== All heap blocks were freed -- no leaks are possible
==4489==
==4489== For lists of detected and suppressed errors, rerun with: -s
==4489== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figura 2.5: *Heap summary codi_vector5.c*

En l'últim codi s'inicialitza un vector estàtic de 10 unitats i s'accedeix mitjançant ell a posicions de memòria que es troben fora d'aquest vector. Tot i que és una acció permesa, pot comportar accedir a un element no existent, ja que el vector només s'utilitza com una referència dintre de la memòria per accedir a qualsevol lloc d'aquesta. En aquest cas però, s'accedeix a posicions llunyanes a les assignades pel vector mitjançant aquest mateix, però aquestes posicions existeixen i per tant no es produeix cap error, com mostra l'output de l'execució mitjançant el programari *Valgrind*. Igualment, una conseqüència de realitzar aquest accés a memòria, es que estàs sobreescrivint posicions situades fora de l'array, llavors d'alguna manera estàs sobreescrivint memòria que durant l'execució del programa no saps si és important o no.

6 Conclusions

Com a conclusions finals podem extreure que l'accés a memòria es pot realitzar de manera molt sofisticada i no es tan senzill com pensàvem que era fins ara. Igualment amb alguns mètodes, com ja hem vist, poden comportar problemes que inclús facin "petar" el programa i causar problemes en el programari, així que s'ha de controlar i programar prèvia i correctament l'accés a memòria que es realitzarà.