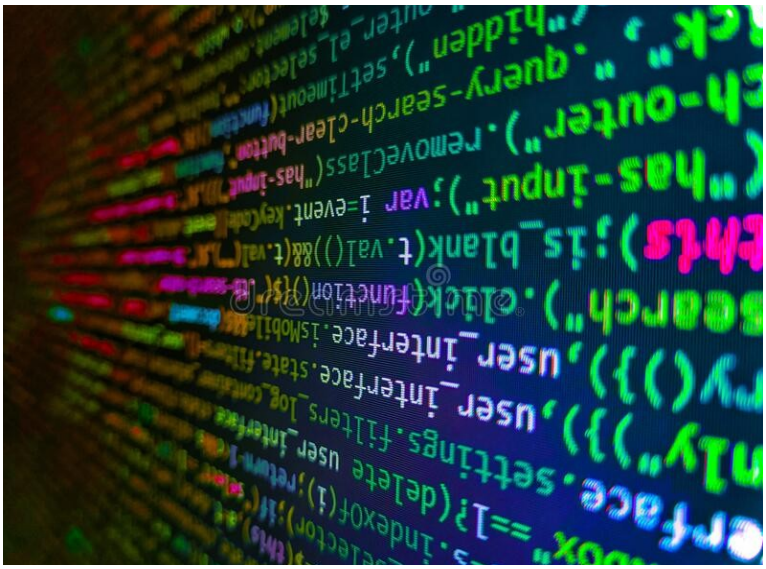
Sistemas Operativos

Teórico-Práctica 10

Semestre de Primavera 2023



Jose Maria Fernández Paredes
Víctor Sort Rubio

Jose Maria Fernández
Víctor Sort

ÍNDICE

1. Introducción 3
2. Ejercicios 3
    2.1 Ejercicio 1 3
    2.2 Ejercicio 2 3
    2.3 Ejercicio 3 3
    2.4 Ejercicio 4 3
    2.5 Ejercicio 5..... 3
3. Conclusiones finales 4
4. Anexos 5
    4.1 Salida Ejercicio 1 5
    4.2 Salida Ejercicio 2 5
    4.3 Salida Ejercicio 3 5
    4.4 Salida Ejercicio 4 5
    4.5 Salida Ejercicio 5 5

Jose Maria Fernández
Víctor Sort

1. INTRODUCCIÓN

En esta sesión de teórico-práctica se tiene como objetivo ver si la aplicación Valgrind, una herramienta de Linux, es capaz de encontrar los problemas relacionados con el acceso a memoria que existen en 5 códigos dados. Intuitivamente, esta aplicación implementa su propia versión de malloc lo que permite detectar los problemas de acceso a memoria, entre otras cosas.

2. EJERCICIOS

Puntualizar que inicialmente hemos compilado las aplicaciones en modo debugger i luego las hemos pasado como argumento al ejecutar la aplicación Valgrind, de la siguiente manera:

```
gcc -g codi_vectorx.c -o codi_x
```

```
valgrind ./codi_x
```

Debido a la longitud de las salidas, estas se encuentran en los anexos.

2.1 Ejercicio 1

El código codi_vector_1 crea un vector dinámico con malloc “a” reservando espacio para 10 enteros, y a continuación escribe en “a[i]”, siendo “i” un entero de 0 a 19. Luego, hace un free del vector.

Al ejecutar-lo con Valgrind, nos dice que al intentar escribir a la posición “a[10]”, que en nuestro caso se corresponde con la posición “0x5213068”, estamos 0 bytes después del vector creado y es un error. Al final, nos dice que hemos hecho 10 errores iguales (escribiendo entre las posiciones 10 a la 20).

Podemos sacar de conclusión que no hay que acceder a una posición de memoria “dentro” de un vector si no se ha reservado espacio para ello, pues no estaremos dentro de él, i no sabemos a la posición a la que estamos accediendo, pudiendo tocar datos delicados.

2.2 Ejercicio 2

El código codi_vector_2 crea un vector dinámico con malloc “a” reservando espacio para 10 enteros, y a continuación escribe en “a[5]”. Luego, hace un free del vector y vuelve a intentar escribir en la misma posición.

Al ejecutar-lo con Valgrind, nos dice que al intentar escribir por segundo vez a la posición “a[5]”, que en nuestro caso se corresponde con la posición “0x5213054”, estamos dentro de un vector cuya memoria ya ha sido eliminada al haber hecho un free, y es por lo tanto un error. Al final, nos dice que hemos cometido únicamente este error.

Podemos sacar de conclusión que no hay que acceder a una posición de memoria de un vector si ya hemos liberado previamente la memoria con free, pues ya podría haber sido ocupada por otros datos sensibles.

Jose Maria Fernández
Víctor Sort

2.3 Ejercicio 3

El código codi_vector_3 crea un vector dinámico con malloc “a” reservando espacio para 10 enteros, y a continuación lee el valor en “a[5]”. Luego, hace un free del vector.

Al ejecutar-lo con Valgrind, nos dice que hemos cometido 5 errores, aunque todos se refieren a lo mismo. Al no haber inicializado la posición “a[5]”, aunque el resultado mostrado nos devuelve 0. Esto pasa pues al acceder a la posición de “a[5]”, esta aún no se encuentra en la memoria física.

Podemos sacar de conclusión que siempre hay que inicializar una posición de un vector dinámico antes de acceder a ella.

2.4 Ejercicio 4

El código codi_vector_4 crea un vector dinámico con malloc “a” reservando espacio para 10 enteros, y a continuación escribe y luego lee el valor en “a[5]”.

Al ejecutar-lo con Valgrind, no nos dice que hayamos cometido ningún error. Aún así, a diferencia de en los otros códigos, en este nos muestra el siguiente mensaje:

```
==3264== LEAK SUMMARY:
==3264==    definitely lost: 40 bytes in 1 blocks
```

Esto nos dice que hemos perdido toda la información que teníamos en el vector de 40 bytes, pues no tenemos manera de saber en que posición de memoria se encuentran los datos ahora.

Podemos sacar de conclusión que siempre hay que liberar la memoria con un free antes de acabar la ejecución de una aplicación.

2.5 Ejercicio 5

El código codi_vector_5, que es el mismo que el código vector_estatic ya visto en clase de teórico-práctica, crea un vector estático “a” de 10 posiciones e intenta escribir y leer en las posiciones “a[100]” y “a[1000]”.

Aunque ya sabemos que esto no es una buena práctica, pues estamos escribiendo en posiciones de memoria cuya información desconocemos y puede ser importante, al ejecutar-lo con Valgrind no detecta que se cometa ningún error, a diferencia de en el ejercicio 1, dónde haciamos el mismo error pero el vector era dinámico en vez de estático.

3. CONCLUSIONES FINALES

La realización de esta teórico-práctica nos ha ayudado a entender bastantes conceptos sobre las memorias virtuales y físicas, así como los vectores dinámicos y estáticos. Además, hemos visto unos cuantos errores típicos que se pueden cometer a la hora de programar.

Además, hemos trabajado muy bien en equipo y creemos que hemos realizado una buena práctica, así que estamos satisfechos con nuestros resultados.

Jose Maria Fernández
Víctor Sort

4. ANEXOS

4.1 Salida Ejercicio 1

```
==2562== Memcheck, a memory error detector
==2562== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2562== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==2562== Command: ./codi_1
==2562==
Escric a la posicio 0
Escric a la posicio 1
Escric a la posicio 2
Escric a la posicio 3
Escric a la posicio 4
Escric a la posicio 5
Escric a la posicio 6
Escric a la posicio 7
Escric a la posicio 8
Escric a la posicio 9
Escric a la posicio 10
==2562== Invalid write of size 4
==2562==    at 0x4005E1: main (codi_vector1.c:13)
==2562==  Address 0x5213068 is 0 bytes after a block of size 40 alloc'd
==2562==      at 0x4C312EF: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-
amd64-linux.so)
==2562==    by 0x4005A8: main (codi_vector1.c:9)
==2562==
Escric a la posicio 11
Escric a la posicio 12
Escric a la posicio 13
Escric a la posicio 14
Escric a la posicio 15
Escric a la posicio 16
Escric a la posicio 17
Escric a la posicio 18
Escric a la posicio 19
==2562==
==2562== HEAP SUMMARY:
```

```
==2562==     in use at exit: 0 bytes in 0 blocks
==2562==   total heap usage: 2 allocs, 2 frees, 1,064 bytes allocated
==2562==
==2562== All heap blocks were freed -- no leaks are possible
==2562==
==2562== For lists of detected and suppressed errors, rerun with: -s
==2562== ERROR SUMMARY: 10 errors from 1 contexts (suppressed: 0 from 0)
```

4.2 Salida Ejercicio 2

```
==3245== Memcheck, a memory error detector
==3245== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3245== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==3245== Command: ./codi_2
==3245==
Escric a la posicio 5
Torno a escriure a la posicio 5
==3245== Invalid write of size 4
==3245==    at 0x4005E3: main (codi_vector2.c:16)
==3245==  Address 0x5213054 is 20 bytes inside a block of size 40 free'd
==3245==    at 0x4C3251B: free (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-
linux.so)
==3245==    by 0x4005D0: main (codi_vector2.c:13)
==3245==  Block was alloc'd at
==3245==      at 0x4C312EF: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-
amd64-linux.so)
==3245==    by 0x4005A8: main (codi_vector2.c:8)
==3245==
==3245==
==3245== HEAP SUMMARY:
==3245==     in use at exit: 0 bytes in 0 blocks
==3245==   total heap usage: 2 allocs, 2 frees, 1,064 bytes allocated
==3245==
==3245== All heap blocks were freed -- no leaks are possible
==3245==
==3245== For lists of detected and suppressed errors, rerun with: -s
==3245== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Jose Maria Fernández
Víctor Sort

4.3 Salida Ejercicio 3

```
==3255== Memcheck, a memory error detector
==3255== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3255== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==3255== Command: ./codi_3
==3255==
==3255== Conditional jump or move depends on uninitialised value(s)
==3255==    at 0x4EA7DFA: __vfprintf_internal (in /lib64/libc-2.31.so)
==3255==    by 0x4E93CC7: printf (in /lib64/libc-2.31.so)
==3255==    by 0x4005C7: main (codi_vector3.c:10)
==3255==
==3255== Use of uninitialised value of size 8
==3255==    at 0x4E8E02B: _itoa_word (in /lib64/libc-2.31.so)
==3255==    by 0x4EA70E7: __vfprintf_internal (in /lib64/libc-2.31.so)
==3255==    by 0x4E93CC7: printf (in /lib64/libc-2.31.so)
==3255==    by 0x4005C7: main (codi_vector3.c:10)
==3255==
==3255== Conditional jump or move depends on uninitialised value(s)
==3255==    at 0x4E8E035: _itoa_word (in /lib64/libc-2.31.so)
==3255==    by 0x4EA70E7: __vfprintf_internal (in /lib64/libc-2.31.so)
==3255==    by 0x4E93CC7: printf (in /lib64/libc-2.31.so)
==3255==    by 0x4005C7: main (codi_vector3.c:10)
==3255==
==3255== Conditional jump or move depends on uninitialised value(s)
==3255==    at 0x4EA719F: __vfprintf_internal (in /lib64/libc-2.31.so)
==3255==    by 0x4E93CC7: printf (in /lib64/libc-2.31.so)
==3255==    by 0x4005C7: main (codi_vector3.c:10)
==3255==
==3255== Conditional jump or move depends on uninitialised value(s)
==3255==    at 0x4EA8244: __vfprintf_internal (in /lib64/libc-2.31.so)
==3255==    by 0x4E93CC7: printf (in /lib64/libc-2.31.so)
==3255==    by 0x4005C7: main (codi_vector3.c:10)
==3255==
Valor d'a[5]: 0
```

```
==3255==
==3255== HEAP SUMMARY:
==3255==     in use at exit: 0 bytes in 0 blocks
==3255==   total heap usage: 2 allocs, 2 frees, 1,064 bytes allocated
==3255==
==3255== All heap blocks were freed -- no leaks are possible
==3255==
==3255== Use --track-origins=yes to see where uninitialised values come from
==3255== For lists of detected and suppressed errors, rerun with: -s
==3255== ERROR SUMMARY: 5 errors from 5 contexts (suppressed: 0 from 0)
```

4.4 Salida Ejercicio 4

```
==3264== Memcheck, a memory error detector
==3264== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3264== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==3264== Command: ./codi_4
==3264==
Valor d'a[5]: 10
==3264==
==3264== HEAP SUMMARY:
==3264==     in use at exit: 40 bytes in 1 blocks
==3264==   total heap usage: 2 allocs, 1 frees, 1,064 bytes allocated
==3264==
==3264== LEAK SUMMARY:
==3264==    definitely lost: 40 bytes in 1 blocks
==3264==    indirectly lost: 0 bytes in 0 blocks
==3264==      possibly lost: 0 bytes in 0 blocks
==3264==    still reachable: 0 bytes in 0 blocks
==3264==         suppressed: 0 bytes in 0 blocks
==3264== Rerun with --leak-check=full to see details of leaked memory
==3264==
==3264== For lists of detected and suppressed errors, rerun with: -s
==3264== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4.5 Salida Ejercicio 5

```
==3274== Memcheck, a memory error detector
==3274== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3274== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==3274== Command: ./codi_5
==3274==
Faig assignacions
a[100] = 1234
a[1000] = 4321
Surto del main
==3274==
==3274== HEAP SUMMARY:
==3274==     in use at exit: 0 bytes in 0 blocks
==3274==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==3274==
==3274== All heap blocks were freed -- no leaks are possible
==3274==
==3274== For lists of detected and suppressed errors, rerun with: -s
==3274== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```