Homework 1: Valgrind Usage

Nombre: Ing. Luis Carlos Alvarez Mata
Carne: 200902144
Nombre: Ing. Verny Moreles Soto
Carne: 201116212

Las fallas de caché y las pérdidas de memoria son un problema común en el desarrollo de software, apenas se notan cuando se trabaja en computadoras personales, sin embargo, cuando se ejecuta un programa con fugas en el sistema incorporado, las cosas generalmente salen mal. Los sistemas embebidos comunes tienen una cantidad limitada de memoria y el uso correcto es crítico para evitar el bloqueo del sistema en los primeros 5 minutos de operaciones. Con esta tarea, el estudiante comenzará a experimentar con herramientas comunes de análisis de memoria y comprenderá cómo rastrearlas efectivamente dentro de un código en ejecución.

1. Resultados

1.1. Caso 1

Como se puede observar en código case1.c se está alocando un espacio de memoria equivalente a 10 enteros. Lo que el error nos está diciendo es que se trata de escribir en una zona de memoria donde no debería. En este caso se está tratando de alocar 4 Bytes extra de lo esperado en la definici'on de malloc, y son 4 Bytes ya que una variable de tipo entero equivale a 4 Bytes de memoria.

Código 1: case1.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv){
  int i;
  int *a = malloc(sizeof(int) * 10);
  if (!a) return -1; /*malloc failed*/
  for (i = 0; i < 11; i++){
    a[i] = i;
  }
  free(a);
  return 0;
}
```

Nos dice además, en cual línea del código está sucediendo el error, lo cual pasaen la linea 9. Como se observa esto es un ciclo que se va a repetir 11 veces, es decir se van a alocar 11 enteros en un espacio de memoria que se había creado únicamente con 10 espacios disponible y nos indica que el bloque inicial es de 40 Bytes (4*10), mientras que lo que se necesitaba era un bloque de 44 Bytes.

Código 2: Salida valgrind de case1.c

```
==8955== Memcheck, a memory error detector
==8955== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8955== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8955== Command: ./case1
==8955==
```

```
==8955== Invalid write of size 4
==8955==   at 0x1086D5: main (case1.c:9)
==8955== Address 0x522d068 is 0 bytes after a block of size 40 alloc'd
==8955==   at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8955==   by 0x1086A2: main (case1.c:6)
==8955==
==8955==
==8955== HEAP SUMMARY:
==8955==    in use at exit: 0 bytes in 0 blocks
==8955==  total heap usage: 1 allocs, 1 frees, 40 bytes allocated
==8955==
==8955== All heap blocks were freed -- no leaks are possible
==8955==
==8955== For counts of detected and suppressed errors, rerun with: -v
==8955== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

1.2. Caso 2

Como se observa en el código case2.c no se realiza ninguna inicialización de memoria para ninguna de las variables utilizadas en el código. Esto es lo que está generando los errores de jump y de uninitialised. Se deben inicializar los espacios de memoria para la variable a e i. Es por eso quese observa un error de asignación de 8 Bytes, ya que ambas son enteros.

Código 3: case2.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv){
  int i;
  int a[10];
  for (i = 0; i < 9; i++)
    a[i] = i;

  for (i = 0; i < 10; i++){
    printf("%d ", a[i]);
  }
  printf("\n");
  return 0;
}
```

Código 4: Salida valgrind de case2.c

```
==8956== Memcheck, a memory error detector
==8956== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8956== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8956== Command: ./case2
==8956==
==8956== Conditional jump or move depends on uninitialised value(s)
==8956==   at 0x4E988DA: vfprintf (vfprintf.c:1642)
==8956==   by 0x4EA0F25: printf (printf.c:33)
==8956==   by 0x10875B: main (case2.c:11)
==8956==
==8956== Use of uninitialised value of size 8
```

```
==8956==    at 0x4E9486B: _itoa_word (_itoa.c:179)
==8956==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==8956==    by 0x4EA0F25: printf (printf.c:33)
==8956==    by 0x10875B: main (case2.c:11)
==8956==
==8956== Conditional jump or move depends on uninitialised value(s)
==8956==    at 0x4E94875: _itoa_word (_itoa.c:179)
==8956==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==8956==    by 0x4EA0F25: printf (printf.c:33)
==8956==    by 0x10875B: main (case2.c:11)
==8956==
==8956== Conditional jump or move depends on uninitialised value(s)
==8956==    at 0x4E98014: vfprintf (vfprintf.c:1642)
==8956==    by 0x4EA0F25: printf (printf.c:33)
==8956==    by 0x10875B: main (case2.c:11)
==8956==
==8956== Conditional jump or move depends on uninitialised value(s)
==8956==    at 0x4E98B4C: vfprintf (vfprintf.c:1642)
==8956==    by 0x4EA0F25: printf (printf.c:33)
==8956==    by 0x10875B: main (case2.c:11)
==8956==
0 1 2 3 4 5 6 7 8 31
==8956==
==8956== HEAP SUMMARY:
==8956==     in use at exit: 0 bytes in 0 blocks
==8956==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==8956==
==8956== All heap blocks were freed -- no leaks are possible
==8956==
==8956== For counts of detected and suppressed errors, rerun with: -v
==8956== Use --track-origins=yes to see where uninitialised values come from
==8956== ERROR SUMMARY: 7 errors from 5 contexts (suppressed: 0 from 0)
```

1.3. Case 3

Como se observa en el código fuente case3.c se estan realizando 10 asignaciones de memoria 10 veces del tamaño de un entero, es decir 40000 Bytes. Al final del código únicamente se libera un bloque de memoria de la variable a, como cada bloque es de 400 Bytes, Valgrind indica que existen aun 3600 Bytes de memoria que se perdieron ya que nunca fueron liberados.

Código 5: case3.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv){
  int i;
  int *a;

  for (i=0; i < 10; i++){
    a = malloc(sizeof(int) * 100);
  }
  free(a);
  return 0;
}
```

Código 6: Salida valgrind de case3.c

```
==8957== Memcheck, a memory error detector
==8957== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8957== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8957== Command: ./case3
==8957==
==8957==
==8957== HEAP SUMMARY:
==8957==    in use at exit: 3,600 bytes in 9 blocks
==8957==  total heap usage: 10 allocs, 1 frees, 4,000 bytes allocated
==8957==
==8957== LEAK SUMMARY:
==8957==   definitely lost: 3,600 bytes in 9 blocks
==8957==   indirectly lost: 0 bytes in 0 blocks
==8957==     possibly lost: 0 bytes in 0 blocks
==8957==   still reachable: 0 bytes in 0 blocks
==8957==        suppressed: 0 bytes in 0 blocks
==8957== Rerun with --leak-check=full to see details of leaked memory
==8957==
==8957== For counts of detected and suppressed errors, rerun with: -v
==8957== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Referencias

[1]"Debug: Understanding Valgrind's messages — EPITECH 2022 - Technical Documentation 1.3.38 documentation." http://docs.epitech2022.eu/en/latest/valgrind.html (accessed Jul. 19, 2020).

[2]"Valgrind - Ubuntu Wiki." https://wiki.ubuntu.com/Valgrind// (accessed Jul. 19, 2020).