

## Tarea 1: Valgrind Usage and Trace Memory Leaks

---

Nombre: Ing. Andrés Gómez Jiménez  
Nombre: Ing. Randy Céspedes Deliyore

Carne: 200935203  
Carne: 201054417

---

## 1 Results using Valgrind

A branch named *Homework\_1* in the repository named *grupo1-HPEC\_2020* that could be find here: [https://github.com/rscd27p/group1\\_HPCE\\_2020/tree/Homework\\_1](https://github.com/rscd27p/group1_HPCE_2020/tree/Homework_1).

The *Valgrind* program was tested with the following source codes: *case1*, *case2*, *case3*.

*Case1* code:

```
#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;
}
```

*Case2* code:

```
#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;
}
```

*Case3* code:

```
#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;
}
```

The result below was obtained after running *Valgrind* in *case1*:

The valgring output reports the next error:

```
==19756== Invalid write of size 4
==19756==    at 0x1086D5: main (case1.c:9)
==19756==    Address 0x522d068 is 0 bytes after a block of size 40 alloc'd
```

From the source we extract the next lines:

```
int *a = malloc(sizeof(int) * 10);
for (i = 0; i < 11; i++){
    a[i] = i;
}
```

As we can see, the malloc operation is reserving memory to save 10 integer values. However, the for loop will actually iterate 11 times (from 0 to 10), which means that in the last iteration the program will try to save data to a location that was not reserved by the malloc. This is why we get the error "Invalid write of size 4", which is basically indicating that the last attempt to save an integer (4 bytes) is trying to access memory that was not reserved.

The result below was obtained after running *Valgrind* in *case2*:

The valgring output reports the next information:

```
==19828== Conditional jump or move depends on uninitialised value(s)
==19828==    at 0x4E9896A: fprintf (fprintf.c:1642)
==19828==    by 0x4EA0FA5: printf (printf.c:33)
==19828==    by 0x10875B: main (case2.c:11)
==19828==
==19828== Use of uninitialised value of size 8
==19828==    at 0x4E948FB: _itoa_word (_itoa.c:179)
==19828==    by 0x4E97F9D: fprintf (fprintf.c:1642)
==19828==    by 0x4EA0FA5: printf (printf.c:33)
==19828==    by 0x10875B: main (case2.c:11)
```

From the source we extract the next lines:

```
for (i = 0; i < 9; i++)  
    a[i] = i;  
  
for (i = 0; i < 10; i++){  
    printf("%d ", a[i]);  
}
```

Where we can observe that 9 elements in the array (0 to 8) were initialized to a known value. However, the last element (index 9) does not have a known value. The error is reported because the print will try to access a value that has not been initialized.

The result below was obtained after running *Valgrind* in *case3*:

The valgring output reports the next information:

```
==19861== HEAP SUMMARY:  
==19861==      in use at exit: 3,600 bytes in 9 blocks  
==19861==    total heap usage: 10 allocs, 1 frees, 4,000 bytes allocated  
==19861==  
==19861== LEAK SUMMARY:  
==19861==    definitely lost: 3,600 bytes in 9 blocks
```

From the source we extract the next lines:

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

Each iteration of the for loop will reserve 400 bytes (100 \* 4 bytes). After 10 iterations, this will have reserved in total 4000 bytes. However, at the end only the last reserved block is freed. This means that 3600 bytes are lost and not properly freed as reported by valgrind.

## 2 LD\_PRELOAD to trace memory leaks

The following code is created to test the *LD<sub>P</sub>RELOAD* function to trace memory leaks:

```
/*  
=====   
Name      : memcheck.c  
Author    : agomez and rcespedes  
Version   : 1.0.0  
Description : General purpose:  
Course: MP-6171 High Performance Embedded Systems  
Tecnologico de Costa Rica (www.tec.ac.cr)  
Input:  
Output:  
=====   
*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <dlfcn.h>
```

```
#include <unistd.h>
#include <sys/wait.h>
#include "config.h"

// Declare variables

char *p = NULL;

// Constant for help on usage
static char usage[] = "Usage: %s [-p ./PROGRAM] [-h][-a]\n"
    "-a displays the information of the author of the program.\n"
    "-h displays the usage message to let the user know how to execute\n"
    "the application.\n"
    "-p PROGRAM specifies the path to the program binary that will be\n"
    "analyzed.\n";

// Use flags below to tell if the required arguments were provided
int p_flag = 0;

int main(int argc, char** argv){
    int c;
    while ((c = getopt (argc, argv, "p:ha")) != -1)
        switch (c){
            case 'p':
                p_flag = 1;
                p = optarg;
                break;
            case 'h':
                fprintf(stderr, usage, argv[0]);
                exit(1);
                break;
            case 'a':
                printf("Authors: agomez and rcespedes\n");
                exit(1);
                break;
            case ':':
                break;
            case '?':
                if (optopt == 'p')
                    fprintf (stderr, "Option -%c requires an argument.\n", optopt);
                else
                    fprintf (stderr, "Unknown option '-%c'.\n", optopt);
                return 1;
            default:
                abort();
        }
}

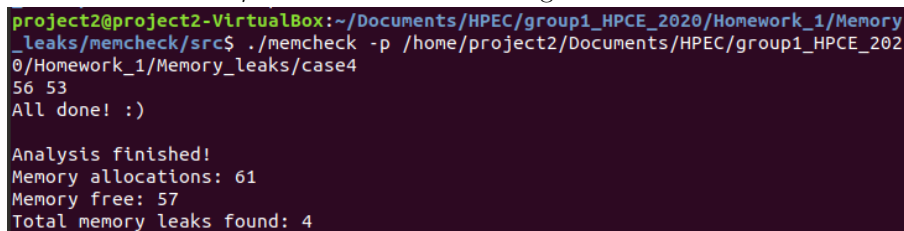
// Load the program to analyze with the LD_PRELOAD variable set to our
// custom libmemcheck.so

// case4 path: /home/project2/Documents/HPEC/group1_HPCE-2020/Homework_1/
```

```
Memory_leaks/case4

if (p_flag = 1){
    char *const program_path = p;
    // Enable line below for troubleshooting
    //printf ("program path = %s\n", p);
    char *const args [] = {program_path, NULL};
    char *const envs [] = {"LD_PRELOAD=./lib/libmemcheck.so", NULL};
    execve(program_path, args, envs);
    printf ("Program succesfully executed\n");
}
return 0;
}
```

This code was tested with the *case4* code and the result in figure 1 was obtained.



```
project2@project2-VirtualBox: ~/Documents/HPEC/group1_HPCE_2020/Homework_1/Memory_leaks/memcheck/src$ ./memcheck -p /home/project2/Documents/HPEC/group1_HPCE_2020/Homework_1/Memory_leaks/case4
56 53
All done! :)

Analysis finished!
Memory allocations: 61
Memory free: 57
Total memory leaks found: 4
```

Figure 1: Memcheck code results on case4 code

To distribute the *memcheck* program this tutorial [1] was followed. To install the memcheck package is necessary to download the following and uncompress the tar file *memcheck-1.0.tar.gz*. Then execute the run the configuration file with the command:

```
./configure
```

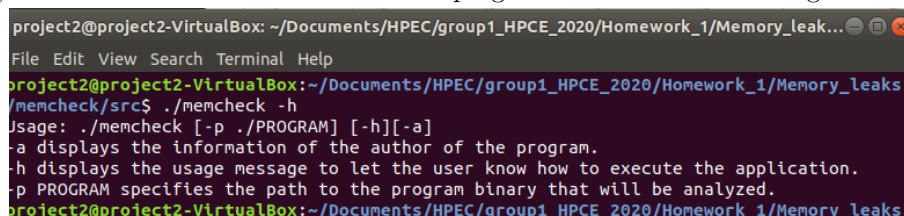
Then use the following command to install the package. **Note:** this may require administrative privileges.

```
make install
```

Once the program is installed it can be found in the *src* folder with the name *memcheck*. It can be run using the command:

```
./memcheck -h
```

This will give you the instruction son how to use the program as it can be seen in figure 2.



```
project2@project2-VirtualBox: ~/Documents/HPEC/group1_HPCE_2020/Homework_1/Memory_leaks/memcheck/src$ ./memcheck -h
Usage: ./memcheck [-p ./PROGRAM] [-h][-a]
-a displays the information of the author of the program.
-h displays the usage message to let the user know how to execute the application.
-p PROGRAM specifies the path to the program binary that will be analyzed.
```

Figure 2: Memcheck code results on case4 code

## References

- [1] I. Free Software Foundation, *Gnu automake*, Online; accessed 18-July-2020, 2020. [Online]. Available: [https://www.gnu.org/software/automake/manual/html\\_node/index.html#Top](https://www.gnu.org/software/automake/manual/html_node/index.html#Top).