

# CS450 Operating Systems (Fall 21)

## Programming Assignment 3

### Part1

Memory leaks and tools to find them

Hassan Alamri & Timur Gaimakov

[halamri@hawk.iit.edu](mailto:halamri@hawk.iit.edu)

A20473047

[tgaimakov@hawk.iit.edu](mailto:tgaimakov@hawk.iit.edu)

A20415319

### A program allocates memory:

We made a simple C program called memLeak.c to allocate the memory using malloc() without freeing it before exiting.

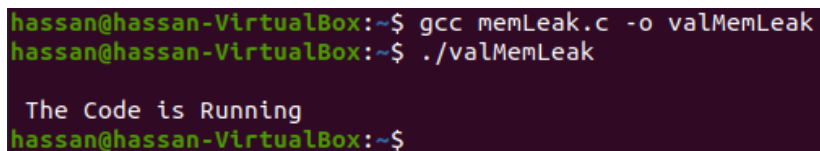
```
#include <stdio.h>
#include <stdlib.h>

int main () {
    char* string = malloc(8 * sizeof(char));
    // We will not free the memory, so there is a leak
    printf ("\n The Code is Running \n");
    return 0;
}
```

### There are some questions need to be addressed:

#### What happens when this program runs?

Once the code is running, it will run fine, and it will print “The Code is Running” but there is a memory leak that we already done it through this code. Since the leaked memory is automatically released, when an application is terminated but if we run it on a server application for long period of time, it will for sure affects our memory. So as programmers, we need to avoid this memory leak; and here using valgrind, we will find a memory leak and we will try to fix it “as a suggestion”.



```
hassan@hassan-VirtualBox:~$ gcc memLeak.c -o valMemLeak
hassan@hassan-VirtualBox:~$ ./valMemLeak

The Code is Running
hassan@hassan-VirtualBox:~$
```

#### Can you use gdb to find any problems with it?

Basically, nothing comes up after running the code in the GDB debugger which yielded that there are no visible errors. We can use GDB to trace such errors like segmentation fault or infinite loop to fix them.

#### How about valgrind (with the command: valgrind --leak-check=yes null)?

Now we are going to use valgrind which is a programming tool for memory debugging, memory leak detection, and profiling. So far, there are several commands for valgrind that would help us tracking and understanding memory leak which are --leak-check=full, --show-leak-kinds=all, and --track-origins=yes. So, we might combine them together for full one command and save the output into external txt file which is: valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes --verbose --log-file=valgrind-out.txt ./filename, or we can use the one that is in the document: (valgrind --leak-check=yes/full null) that searches for memory leaks when the client program finishes. As we see, if set to full or yes, each individual leak will be shown in detail and/or counted as an error, as specified by the options --show-leak-kinds and --errors-for-leak-kinds.

```
hassan@hassan-VirtualBox:~$ gcc memLeak.c -o valMemLeak
hassan@hassan-VirtualBox:~$ valgrind --leak-check=full --log-file=valgrind-out.txt ./valMemLeak

The Code is Running
hassan@hassan-VirtualBox:~$
```

Now, we will look at the report which is already created after this command just if you type `ls` command or just write `cat valgrind-out.txt`:

```
hassan@hassan-VirtualBox:~$ cat valgrind-out.txt
==50013== Memcheck, a memory error detector
==50013== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==50013== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==50013== Command: ./valMemLeak
==50013== Parent PID: 49967
==50013==
==50013==
==50013== HEAP SUMMARY:
==50013==   in use at exit: 8 bytes in 1 blocks
==50013==   total heap usage: 2 allocs, 1 frees, 1,032 bytes allocated
==50013==
==50013== 8 bytes in 1 blocks are definitely lost in loss record 1 of 1
==50013==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==50013==    by 0x10917E: main (in /home/hassan/valMemLeak)
==50013==
==50013== LEAK SUMMARY:
==50013==   definitely lost: 8 bytes in 1 blocks
==50013==   indirectly lost: 0 bytes in 0 blocks
==50013==   possibly lost: 0 bytes in 0 blocks
==50013==   still reachable: 0 bytes in 0 blocks
==50013==   suppressed: 0 bytes in 0 blocks
==50013==
==50013== For lists of detected and suppressed errors, rerun with: -s
==50013== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
hassan@hassan-VirtualBox:~$
```

We notice from the output that using the Valgrind leak-check command on the program it would reveal the memory leaks, specifying the number of bytes leaked, and how many blocks were left unfreed. It provides the address, and the associated line of code where each memory block was allocated, allowing for easy debugging.

## Test cases for valgrind

### Explain why you choose them and the expected results

**We made three test cases programs.** Each program has a test case namely “testcase” function that demonstrate different types of failures, and its correction function namely “testSoln” function to avoid memory leak and misuse of malloc. When we run the test case function in every program, it runs fine as if there are no errors, though, there is terrible memory leaks and errors injections to a system. Therefore, we choose these kinds of test cases because they mostly happened with beginners’ developers which is out of their expectations. Meanwhile, with each test case, we made the test case solution’s function which shows the correct way of using malloc and freeing the memory before exiting the main function.

**The first test case program is (memleak1.c).** we made a function that from first glimpse, we may not notice that there are some radical errors since developers think they got the output would be satisfied, instead we made errors such as misuse of malloc or freeing a memory by assigning invalid values in the array called heights array inside the test case function.

```
==8975== Invalid write of size 4
==8975==    at 0x1091D0: testCase (in /home/hassan/valMemLeak1)
==8975==    by 0x1092BE: main (in /home/hassan/valMemLeak1)
==8975==    Address 0x4a55040 is 0 bytes inside a block of size 3 alloc'd
==8975==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
```

So, we are trying to get access into the memory to acquiring some space for our values, but it gives invalid access because the malloc() takes byte values, and developer thinks that passing constant integer value to malloc would be enough. Therefore, we made “testSoln” function to handle this problem which is a correct version of the test case code including the free() function.

```
==8834== HEAP SUMMARY:
==8834==    in use at exit: 0 bytes in 0 blocks
==8834==    total heap usage: 2 allocs, 2 frees, 1,036 bytes allocated
==8834==
==8834== All heap blocks were freed -- no leaks are possible
```

**The second test case program is (memleak2.c).** As an extension from the first test case, we made a simple test case that tests reading from the memory after freeing it. We thought that there would be no errors since we handled the malloc function correctly and we freed the memory. Instead, what was

```
==9663== Invalid read of size 4
==9663==    at 0x109222: testCase (in /home/hassan/valMemLeak2)
==9663==    by 0x10925B: main (in /home/hassan/valMemLeak2)
==9663==    Address 0x4a55044 is 4 bytes inside a block of size 12 free'd
==9663==    at 0x483CA3F: free (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==9663==    by 0x109219: testCase (in /home/hassan/valMemLeak2)
==9663==    by 0x10925B: main (in /home/hassan/valMemLeak2)
==9663==    Block was alloc'd at
==9663==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==9663==    by 0x1091AC: testCase (in /home/hassan/valMemLeak2)
==9663==    by 0x10925B: main (in /home/hassan/valMemLeak2)
==9663==
==9663==
==9663== HEAP SUMMARY:
==9663==    in use at exit: 0 bytes in 0 blocks
==9663==    total heap usage: 2 allocs, 2 frees, 1,036 bytes allocated
==9663==
==9663== All heap blocks were freed -- no leaks are possible
==9663==
==9663== For lists of detected and suppressed errors, rerun with: -s
==9663== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

shown through the valgrain that there is an error which is invalid reading from the memory because we do not have access to the memory after freeing it.

**The third test case program is (memLeak3.c).** It is the same as second test case instead we assign a new value to the heights array after free the memory. It shows that there are some errors and the major one is invalid writing to the memory because we already free it. So, we cannot go to the same memory address because it might be in use or reserved with different applications.

```
==12437== Invalid write of size 4
==12437==    at 0x10924A: testCase (in /home/hassan/valMemLeak3)
==12437==    by 0x109355: main (in /home/hassan/valMemLeak3)
==12437== Address 0x4a55040 is 0 bytes inside a block of size 12 free'd
==12437==    at 0x483CA3F: free (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==12437==    by 0x109239: testCase (in /home/hassan/valMemLeak3)
==12437==    by 0x109355: main (in /home/hassan/valMemLeak3)
==12437== Block was alloc'd at
==12437==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==12437==    by 0x1091CC: testCase (in /home/hassan/valMemLeak3)
==12437==    by 0x109355: main (in /home/hassan/valMemLeak3)
```