Minimal Valgrind Tutorial

By Alexandra Stefan

Last updated: 1/21/2024

This document is dedicated to using Valgrind. It assumes you know how to use one of omega/VM/Ubuntu (to transfer files, compile code, run commands).

Valgrind is already installed on both omega and the VM from the cse13xx page.

To install it on Ubuntu run: `sudo apt install valgrind`

Below is a simple tutorial. For more information go to the official Valgrind page. You can start from the Valgrind Quick Start Guide. For more information see also FAQ and the User Manual accessible from the same page.

Run code with Valgrind

Copy the memory_errors.c file on omega or the VM and go to the location with the file.

1. compile with the `-g` flag so that Valgrind will give the line number where the error was found:

```
gcc -g memory_errors.c
```

2. run with flag: `--leak-check=full`

2a) Run with user input:

```
valgrind --leak-check=full ./a.out
```

2b) Run with file redirection (create a file named data.txt that only has a number between 0 and 3 in it and is in the same folder as the memory_errors.c file):

```
valgrind --leak-check=full ./a.out < data.txt
```

Note that even if you have an executable that was not produced with debugging information (i.e. was compiled without the –g flag) you can still run Valgrind, but it will not show line numbers for errors. Simply run it with: `valgrind --leak-check=yes ./myprog`

Sample Valgrind reports

The report you get will have a DIFFERENT number than mine to the left of the lines (e.g. instead of the ==18931== below). That is fine. That number is irrelevant.

Sample GOOD Valgrind report

No errors reported by Valgrind, see: `0 errors from 0 contexts`.

It is ok to have the `(suppressed: 4 from 4)` message.

```
==18931== HEAP SUMMARY:
==18931==     in use at exit: 0 bytes in 0 blocks
==18931==   total heap usage: 6 allocs, 6 frees, 96 bytes allocated
==18931==
==18931== All heap blocks were freed -- no leaks are possible
==18931==
==18931== For counts of detected and suppressed errors, rerun with: -v
==18931== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
```

Sample BAD Valgrind report 1

Report from running memory_errors.c with input 1 (calls function `pointer_error()`):

```
This program runs one of the 3 tests below at a time:
 0 - no error.
 1 - pointer error (invalid memory access)
 2 - memory leak
 3 - conditional jump depends on unitialized value
Enter your test choice (0-3): 1
==2031== Use of uninitialised value of size 8
==2031==    at 0x10899E: pointer_error (memory_errors.c:70)
==2031==    by 0x1088A6: main (memory_errors.c:33)
==2031==
==2031==
==2031== Process terminating with default action of signal 11 (SIGSEGV)
==2031==  Bad permissions for mapped region at address 0x1086F0
==2031==    at 0x10899E: pointer_error (memory_errors.c:70)
==2031==    by 0x1088A6: main (memory_errors.c:33)
==2031==
==2031== HEAP SUMMARY:
==2031==     in use at exit: 0 bytes in 0 blocks
==2031==   total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated
==2031==
==2031== All heap blocks were freed -- no leaks are possible
==2031==
==2031== Use --track-origins=yes to see where uninitialised values come
from
==2031== For lists of detected and suppressed errors, rerun with: -s
==2031== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Segmentation fault (core dumped)
```

Commented [SA1]: The uninitialized value is the pointer value (size of a pointer is 8 bytes)

Commented [SA2]: Bad memory access.

Commented [SA3]: See function name and line number that generated the error.

Commented [SA4]: Even though there are no leaks, there is still a memory related error (accessing a memory location that was not ours).

Commented [SA5]: This indicates that this program has an error.

Sample BAD Valgrind report 2

Report from running memory_errors.c with input 2 (calls function `mem_leak_error()`):

```
Started...
```

```
This program runs one of the 3 tests below at a time:
 0 - no error.
 1 - pointer error (invalid memory access)
 2 - memory leak
 3 - conditional jump depends on unitialized value
Enter your test choice (0-3): 2
Finished...
==2043==
==2043== HEAP SUMMARY:
==2043==     in use at exit: 40 bytes in 1 blocks
==2043==   total heap usage: 3 allocs, 2 frees, 2,088 bytes allocated
==2043==
==2043== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2043==    at 0x4C2FECB: malloc (vg_replace_malloc.c:307)
==2043==    by 0x1089D8: mem_leak_error (memory_errors.c:78)
==2043==    by 0x1088BA: main (memory_errors.c:36)
==2043==
==2043== LEAK SUMMARY:
==2043==    definitely lost: 40 bytes in 1 blocks
==2043==    indirectly lost: 0 bytes in 0 blocks
==2043==      possibly lost: 0 bytes in 0 blocks
==2043==    still reachable: 0 bytes in 0 blocks
==2043==         suppressed: 0 bytes in 0 blocks
==2043==
==2043== For lists of detected and suppressed errors, rerun with: -s
==2043== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Commented [SA6]: Function and line number of the line that allocated the memory that was not freed.

Commented [SA7]: Line that caused the error:
double *arr = malloc(5 * sizeof(double));
size of double is 8 bytes
The 40 bytes lost are from 5 x 8 bytes (we allocated space for 5 doubles).

Sample BAD Valgrind report 3

Report from running memory_errors.c with input 3 (calls function `cond_jump_error()`):

```
Started...
This program runs one of the 3 tests below at a time:
 0 - no error.
 1 - pointer error (invalid memory access)
 2 - memory leak
 3 - conditional jump depends on unitialized value
Enter your test choice (0-3): 3
==2047== Conditional jump or move depends on uninitialised value(s)
==2047==    at 0x1089F0: cond_jump_error (memory_errors.c:85)
==2047==    by 0x1088CE: main (memory_errors.c:39)
==2047==
n is 0
Finished...
==2047==
==2047== HEAP SUMMARY:
==2047==     in use at exit: 0 bytes in 0 blocks
==2047==   total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated
==2047==
==2047== All heap blocks were freed -- no leaks are possible
==2047==
```

Commented [SA8]: Some condition (from an if/while/for instruction) depends on a value that was not initialized.

Commented [SA9]: Line 85 has code:
if (n==0) {
where n was not initialized.

```
==2047== Use --track-origins=yes to see where uninitialised values come
from
==2047== For lists of detected and suppressed errors, rerun with: -s
==2047== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Note the suggested flag: "`Use --track-origins=yes to see where uninitialised values come from`"
If you rerun with it:

```
valgrind --leak-check=full --track-origins=yes  ./a.out
```

It will show that the unitialized value is on the stack:

```
…
==2050== Conditional jump or move depends on uninitialised value(s)
==2050==    at 0x1089F0: cond_jump_error (memory_errors.c:85)
==2050==    by 0x1088CE: main (memory_errors.c:39)
==2050==  Uninitialised value was created by a stack allocation
==2050==    at 0x1089E4: cond_jump_error (memory_errors.c:83)
…
```

Sample error message 4: invalid write

Sample Valgrind report of invalid memory access. The program attempts to write past the allocated space. It only checks this for dynamically allocated data (on the heap), bt not for arrays allocated on the stack.

```
==9814==   Invalid write of size 1
==9814==   at 0x804841E: main (example2.c:6)
==9814==  Address 0x1BA3607A is 0 bytes after a block of size 10 alloc'd
```

data.txt (link)

```
1
```

Practice:

What causes this program to have a memory leak: fileread_test.c, fileread_data.txt?

fileread_data.txt (link)

```
a       un, uno, una[Article]
aardvark        cerdo hormiguero
ab      prefijo que indica separacio/n
aback   hacia atras
abacterial      abacteriano, sin bacterias
abacus  a/baco
```

fileread_test.c (link)

```
/*
 When you run this program with Valgrind, you will find a memory leak.
 What is the cause of this leak?
  */

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

void read_file(char* fname);

int main(int argc, char** argv) {

    read_file("fileread_data.txt");

    return (EXIT_SUCCESS);
}


void read_file(char* fname) {
    FILE * fp = fopen(fname, "r");
    if (fp == NULL) {
        return;
    }

       size_t len = 0;
    char * buffer = (char*) malloc(1001 * sizeof (char));
    int count = 0;
    size_t read;

    while ((read = getline(&buffer, &len, fp)) != -1) {
        printf("buffer = %s\n", buffer);
    }

    fclose(fp);
    free(buffer);
}
```