Part 0: Setting up Valgrind

```
dei@dei-server:~/docker/xv6/labs/3$ valgrind ls -l
==1398011== Memcheck, a memory error detector
==1398011== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1398011== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==1398011== Command: ls -l
==1398011==
total 16088
drwxr-xr-x 25 dei dei     4096 Mar 30 14:28 valgrind-3.20.0
-rw-rw-r--  1 dei dei 16469274 Mar 30 14:14 valgrind-3.20.0.tar.bz2
==1398011==
==1398011== HEAP SUMMARY:
==1398011==     in use at exit: 20,376 bytes in 9 blocks
==1398011==   total heap usage: 254 allocs, 245 frees, 105,830 bytes allocated
==1398011==
==1398011== LEAK SUMMARY:
==1398011==    definitely lost: 0 bytes in 0 blocks
==1398011==    indirectly lost: 0 bytes in 0 blocks
==1398011==      possibly lost: 0 bytes in 0 blocks
==1398011==    still reachable: 20,376 bytes in 9 blocks
==1398011==         suppressed: 0 bytes in 0 blocks
==1398011== Rerun with --leak-check=full to see details of leaked memory
==1398011==
==1398011== For lists of detected and suppressed errors, rerun with: -s
==1398011== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Part 1a: Creating memory leak

Sample program that allocates memory using malloc() but forgets to free it before exiting in C:

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

int main() {
    int* ptr = (int*) malloc(sizeof(int));
    *ptr = 10;
    printf("%d\n", *ptr);
    return 0;
}
```

This program allocates memory for an integer using malloc() and assigns it a value of 10. Iit does not free the allocated memory before exiting. This can lead to memory leaks in larger programs.

GDB gives this output running the above first implementation:

```
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) run ./memLeaker
Starting program:  ./memLeaker
No executable file specified.
Use the "file" or "exec-file" command.
(gdb) file ./memLeaker
Reading symbols from ./memLeaker...
(No debugging symbols found in ./memLeaker)
(gdb) run
Starting program: /home/dei/docker/xv6/labs/3/memLeaker ./memLeaker
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
10
[Inferior 1 (process 1408572) exited normally]
(gdb)
```

As can be seen, the process exited normally with no issues…

I altered the program to give me more output and hopefully crash or give errors:

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

int main() {
    int* ptr = (int*) malloc(sizeof(int));
    *ptr = 100;
    int count=1000000;
    while(count>0){
        printf("%d\n", *ptr);
        count--;
    }
    return 0;
}
```

GDB gives this output running this:

```
100
100
[Inferior 1 (process 1410334) exited normally]
(gdb)
```

Once again no errors observed in terms of memory leaks.

I altered the program one last time as another test case for GDB. This can be observed below. I realized having a loop will not change anything in terms of memory and my initial hypothesis of it triggering memory issues if run consistently was wrong:

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

int main() {
    int* ptr = (int*) malloc(sizeof(int));
    *ptr = 10;
    printf("%d\n", *ptr);
    free(ptr);
    int* ptr2 = (int*) malloc(sizeof(int));
    *ptr2 = 20;
    printf("%d\n", *ptr2);
    return 0;
}
```

This time I free the initial pointer and create a new memory block that I do not free. I use the -g flag when compiling and set a breakpoint in gdb like so:

```
dei@dei-server:~/docker/xv6/labs/3$ gcc -g memLeaker.c
dei@dei-server:~/docker/xv6/labs/3$ gdb ./a.out
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./a.out...
(gdb) break 10
Breakpoint 1 at 0x11e3: file memLeaker.c, line 10.
(gdb) break 12
Breakpoint 2 at 0x1209: file memLeaker.c, line 12.
(gdb) run
Starting program: /home/dei/docker/xv6/labs/3/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
10

Breakpoint 1, main () at memLeaker.c:10
10          *ptr2 = 20;
```

As you can see I ran and stopped at the breakpoint 1. I then checked memory:

```
(gdb) info proc mappings
process 1418302
Mapped address spaces:

          Start Addr           End Addr       Size     Offset  Perms  objfile
      0x555555554000     0x555555555000     0x1000        0x0  r--p   /home/dei/docker/xv6/labs/3/a.out
      0x555555555000     0x555555556000     0x1000     0x1000  r-xp   /home/dei/docker/xv6/labs/3/a.out
      0x555555556000     0x555555557000     0x1000     0x2000  r--p   /home/dei/docker/xv6/labs/3/a.out
      0x555555557000     0x555555558000     0x1000     0x2000  r--p   /home/dei/docker/xv6/labs/3/a.out
      0x555555558000     0x555555559000     0x1000     0x3000  rw-p   /home/dei/docker/xv6/labs/3/a.out
      0x555555559000     0x55555557a000    0x21000        0x0  rw-p   [heap]
      0x7ffff7d88000     0x7ffff7d8b000     0x3000        0x0  rw-p
      0x7ffff7d8b000     0x7ffff7db3000    0x28000        0x0  r--p   /usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7db3000     0x7ffff7f48000   0x195000    0x28000  r-xp   /usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7f48000     0x7ffff7fa0000    0x58000   0x1bd000  r--p   /usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7fa0000     0x7ffff7fa4000     0x4000   0x214000  r--p   /usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7fa4000     0x7ffff7fa6000     0x2000   0x218000  rw-p   /usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7fa6000     0x7ffff7fb3000     0xd000        0x0  rw-p
      0x7ffff7fbb000     0x7ffff7fbd000     0x2000        0x0  rw-p
      0x7ffff7fbd000     0x7ffff7fc1000     0x4000        0x0  r--p   [vvar]
      0x7ffff7fc1000     0x7ffff7fc3000     0x2000        0x0  r-xp   [vdso]
      0x7ffff7fc3000     0x7ffff7fc5000     0x2000        0x0  r--p   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
      0x7ffff7fc5000     0x7ffff7fef000    0x2a000     0x2000  r-xp   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
      0x7ffff7fef000     0x7ffff7ffa000     0xb000    0x2c000  r--p   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
      0x7ffff7ffb000     0x7ffff7ffd000     0x2000    0x37000  r--p   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
      0x7ffff7ffd000     0x7ffff7fff000     0x2000    0x39000  rw-p   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
      0x7ffffffde000     0x7ffffffff000    0x21000        0x0  rw-p   [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0  --xp   [vsyscall]
```

Based on these addresses we can see there are 5 bytes being allocated for my program each with 5 0x1 addresses apart.

Next I will test this program on valgrind and see what results we can observe:

```
dei@dei-server:~/docker/xv6/labs/3$ valgrind ./a.out
==1422256== Memcheck, a memory error detector
==1422256== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1422256== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==1422256== Command: ./a.out
==1422256==
10
20
==1422256==
==1422256== HEAP SUMMARY:
==1422256==     in use at exit: 4 bytes in 1 blocks
==1422256==   total heap usage: 3 allocs, 2 frees, 1,032 bytes allocated
==1422256==
==1422256== LEAK SUMMARY:
==1422256==    definitely lost: 4 bytes in 1 blocks
==1422256==    indirectly lost: 0 bytes in 0 blocks
==1422256==      possibly lost: 0 bytes in 0 blocks
==1422256==    still reachable: 0 bytes in 0 blocks
==1422256==         suppressed: 0 bytes in 0 blocks
==1422256== Rerun with --leak-check=full to see details of leaked memory
==1422256==
==1422256== For lists of detected and suppressed errors, rerun with: -s
==1422256== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
dei@dei-server:~/docker/xv6/labs/3$
```

I will rerun with –leak-check=full:

```
dei@dei-server:~/docker/xv6/labs/3$ valgrind --leak-check=full ./a.out
==1425085== Memcheck, a memory error detector
==1425085== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1425085== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==1425085== Command: ./a.out
==1425085==
10
20
==1425085==
==1425085== HEAP SUMMARY:
==1425085==     in use at exit: 4 bytes in 1 blocks
==1425085==   total heap usage: 3 allocs, 2 frees, 1,032 bytes allocated
==1425085==
==1425085== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1425085==    at 0x484884F: malloc (in /usr/local/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==1425085==    by 0x1091DE: main (memLeaker.c:9)
==1425085==
==1425085== LEAK SUMMARY:
==1425085==    definitely lost: 4 bytes in 1 blocks
==1425085==    indirectly lost: 0 bytes in 0 blocks
==1425085==      possibly lost: 0 bytes in 0 blocks
==1425085==    still reachable: 0 bytes in 0 blocks
==1425085==         suppressed: 0 bytes in 0 blocks
==1425085==
==1425085== For lists of detected and suppressed errors, rerun with: -s
==1425085== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

As can be seen valgrind actually saw our 4 bytes* (not 5 as previously thought) and we can see that they are lost. I will test our previous example as well that can be seen below:

Input:

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

int main() {
    int* ptr = (int*) malloc(sizeof(int));
    *ptr = 100;
    int count=100;
    while(count>0){
        printf("%d\n", *ptr);
        count--;
    }
    return 0;
}
```

Output:

```
100
100
100
100
100
100
100
100
100
100
100
==1427195==
==1427195== HEAP SUMMARY:
==1427195==     in use at exit: 4 bytes in 1 blocks
==1427195==   total heap usage: 2 allocs, 1 frees, 1,028 bytes allocated
==1427195==
==1427195== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1427195==    at 0x484884F: malloc (in /usr/local/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==1427195==    by 0x10917E: main (memLeaker.c:6)
==1427195==
==1427195== LEAK SUMMARY:
==1427195==    definitely lost: 4 bytes in 1 blocks
==1427195==    indirectly lost: 0 bytes in 0 blocks
==1427195==      possibly lost: 0 bytes in 0 blocks
==1427195==    still reachable: 0 bytes in 0 blocks
==1427195==         suppressed: 0 bytes in 0 blocks
==1427195==
==1427195== For lists of detected and suppressed errors, rerun with: -s
==1427195== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

As hypothesized the while loop did not affect our memory blocks and thus no real other issues arose. Same result as previously observed.

Overall we observe that GDB does not find any issues with our code in terms of memory. It displays the allocated addresses and memory block but it does not detect any issues. On the flip side VALGRIND does detect issues with our code and the unreleased malloc call is an issue here.. as expected. The –leak-check=full actually showcased our error whereas running regular valgrind showed no error just lost memory. We will move on to making more valgrind examples now.

Part 1b: Creating Other Valgrind Examples

I will provide 2 other examples that I can think of that valgrind would trigger memory errors for. I will also summarize the previous example I used:

1) Forgetting to free memory after malloc() (used previously)

```c
int main() {
    int *ptr = malloc(sizeof(int));
    return 0;
}
```

2) Overwriting a pointer to allocated memory with a new value without freeing the original memory.

```c
int main() {
    int *ptr = malloc(sizeof(int));
    ptr = malloc(sizeof(int));
    free(ptr);
    return 0;
}
```

3) Allocating memory in a loop without freeing it (what I was trying in part 1a).

```c
int main() {
    int *ptr;
    for (int i = 0; i < 10; i++) {
        ptr = malloc(sizeof(int));
    }
    free(ptr);
    return 0;
}
```

The respective output from valgrind for each can be observed below:

1) Forgetting to free memory after malloc() (used previously)

```
dei@dei-server:~/docker/xv6/labs/3$ valgrind --leak-check=full ./memLeaker_1
==1437195== Memcheck, a memory error detector
==1437195== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1437195== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==1437195== Command: ./memLeaker_1
==1437195==
==1437195==
==1437195== HEAP SUMMARY:
==1437195==     in use at exit: 4 bytes in 1 blocks
==1437195==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==1437195==
==1437195== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1437195==    at 0x484884F: malloc (in /usr/local/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==1437195==    by 0x10915E: main (memLeaker.c:4)
==1437195==
==1437195== LEAK SUMMARY:
==1437195==    definitely lost: 4 bytes in 1 blocks
==1437195==    indirectly lost: 0 bytes in 0 blocks
==1437195==      possibly lost: 0 bytes in 0 blocks
==1437195==    still reachable: 0 bytes in 0 blocks
==1437195==         suppressed: 0 bytes in 0 blocks
==1437195==
==1437195== For lists of detected and suppressed errors, rerun with: -s
==1437195== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

2) Overwriting a pointer to allocated memory with a new value without freeing the original memory.

```
dei@dei-server:~/docker/xv6/labs/3$ valgrind --leak-check=full ./memLeaker_2
==1441748== Memcheck, a memory error detector
==1441748== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1441748== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==1441748== Command: ./memLeaker_2
==1441748==
==1441748==
==1441748== HEAP SUMMARY:
==1441748==     in use at exit: 4 bytes in 1 blocks
==1441748==   total heap usage: 2 allocs, 1 frees, 8 bytes allocated
==1441748==
==1441748== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1441748==    at 0x484884F: malloc (in /usr/local/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==1441748==    by 0x10917E: main (memLeaker.c:4)
==1441748==
==1441748== LEAK SUMMARY:
==1441748==    definitely lost: 4 bytes in 1 blocks
==1441748==    indirectly lost: 0 bytes in 0 blocks
==1441748==      possibly lost: 0 bytes in 0 blocks
==1441748==    still reachable: 0 bytes in 0 blocks
==1441748==         suppressed: 0 bytes in 0 blocks
==1441748==
==1441748== For lists of detected and suppressed errors, rerun with: -s
==1441748== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

3) Allocating memory in a loop without freeing it (my initial thoughts perfected).

```
dei@dei-server:~/docker/xv6/labs/3$ valgrind --leak-check=full ./memLeaker_3
==1442229== Memcheck, a memory error detector
==1442229== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1442229== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==1442229== Command: ./memLeaker_3
==1442229==
==1442229==
==1442229== HEAP SUMMARY:
==1442229==     in use at exit: 36 bytes in 9 blocks
==1442229==   total heap usage: 10 allocs, 1 frees, 40 bytes allocated
==1442229==
==1442229== 36 bytes in 9 blocks are definitely lost in loss record 1 of 1
==1442229==    at 0x484884F: malloc (in /usr/local/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==1442229==    by 0x109187: main (memLeaker.c:6)
==1442229==
==1442229== LEAK SUMMARY:
==1442229==    definitely lost: 36 bytes in 9 blocks
==1442229==    indirectly lost: 0 bytes in 0 blocks
==1442229==      possibly lost: 0 bytes in 0 blocks
==1442229==    still reachable: 0 bytes in 0 blocks
==1442229==         suppressed: 0 bytes in 0 blocks
==1442229==
==1442229== For lists of detected and suppressed errors, rerun with: -s
==1442229== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Using a for loop I was able to get what I was initially trying to do. Multiple memory leaks with each iteration. I believe we can remove the free(ptr) to get another block of data lost which I will attempt below:

Input:

```c
int main() {
    int *ptr;
    for (int i = 0; i < 10; i++) {
        ptr = malloc(sizeof(int));
    }
    return 0;
}
```

Output:

```
dei@dei-server:~/docker/xv6/labs/3$ valgrind --leak-check=full ./memLeaker_3-1
==1443576== Memcheck, a memory error detector
==1443576== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1443576== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==1443576== Command: ./memLeaker_3-1
==1443576==
==1443576==
==1443576== HEAP SUMMARY:
==1443576==     in use at exit: 40 bytes in 10 blocks
==1443576==   total heap usage: 10 allocs, 0 frees, 40 bytes allocated
==1443576==
==1443576== 40 bytes in 10 blocks are definitely lost in loss record 1 of 1
==1443576==    at 0x484884F: malloc (in /usr/local/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==1443576==    by 0x109167: main (memLeaker.c:6)
==1443576==
==1443576== LEAK SUMMARY:
==1443576==    definitely lost: 40 bytes in 10 blocks
==1443576==    indirectly lost: 0 bytes in 0 blocks
==1443576==      possibly lost: 0 bytes in 0 blocks
==1443576==    still reachable: 0 bytes in 0 blocks
==1443576==         suppressed: 0 bytes in 0 blocks
==1443576==
==1443576== For lists of detected and suppressed errors, rerun with: -s
==1443576== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

As can be seen, success. This Gives us 3, possibly 4 examples for memory leaks in valgrind.