

Programming Assignment 3 Part A

Binghan Geng
A20482350
bgeng1@hawk.iit.edu

Yu Li
A20496405
yli385@hawk.iit.edu

1. Project background

This part is to learn how to use the debugging tools gdb and valgrind to find memory leaks and other insidious memory problems.

2. Test scope

2.1 Involving platforms

Ubuntu 18.04.4 LTS + valgrind-3.13.0

3. Test Case

3.1 Test Cast 1

This test case allocates memory using malloc() and free it before exiting.

```
// mem_leak.c
#include <stdlib.h>

void test_memory_leak(int size) {
    malloc(size);
}

void no_leak() {
    void *p = malloc(1024 * 1024);
    free(p);
}

int main(int argc, char* argv[]) {
    no_leak();
}
```

```

    test_memory_leak(1024);

    return 0;
}

```

3.2 Test Cast 2

This test case allocates memory using malloc() but forgets to free it before exiting.

```

// mem_leak2.c
#include <stdlib.h>

void test_memory_leak() {
    void *p = malloc(4096);

    p = malloc(1024); free(p);
}

int main(int argc, char* argv[]) {
    test_memory_leak();

    return 0;
}

```

3.3 Test execution by GDB

Step 1: compile, and enable GDB debugging.

```

cs450@ubuntu:~/PA3_part1$ gcc mem_leak.c -g -o mem_leak 1
cs450@ubuntu:~/PA3_part1$ gdb ./mem_leak 2
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 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:
<http://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 ./mem_leak...done.
(gdb) 

```

Step 2: set breakpoint at line 9.

(gdb) b 9

```

C mem_leak.c x
C mem_leak.c > ...
1 // mem_leak.c
2 #include <stdlib.h>
3
4 void test_memory_leak(int size) {
5     malloc(size);
6 }
7
8 void no_leak() {
9     void *p = malloc(1024 * 1024);
10    free(p);
11 }
12
13 int main(int argc, char* argv[]) {
14    no_leak();
15    test_memory_leak(1024);
16
17    return 0;

```

```
(gdb) b 9
Breakpoint 1 at 0x6ad: file mem_leak.c, line 9.
(gdb) r
Starting program: /home/cs450/PA3_part1/mem_leak

Breakpoint 1, no_leak () at mem_leak.c:9
9      void *p = malloc(1024 * 1024);
(gdb) |
```

Step 3: executing line 9.

Before executing to line 9, the total in use bytes (i.e. the memory being used by the process) is 0. After executing line 9, the total in use bytes is 1053264, at which point `free`. 1053264 is similar to the space allocated using `malloc` $1024 * 1024 = 1048576$ (part of the space is used for 1053264 is similar to the space allocated using `malloc`, $1024 * 1024 = 1048576$ (part of the space is used to maintain the memory allocation data structure)).

The screenshot shows a code editor with the source code of `mem_leak.c`. Line 9, `void *p = malloc(1024 * 1024);`, is highlighted with a red box. Below the code editor is a terminal window showing the output of `malloc_stats()` before and after line 9 is executed. The output shows that the 'in use bytes' increase from 0 to 1053264 after line 9 is executed.

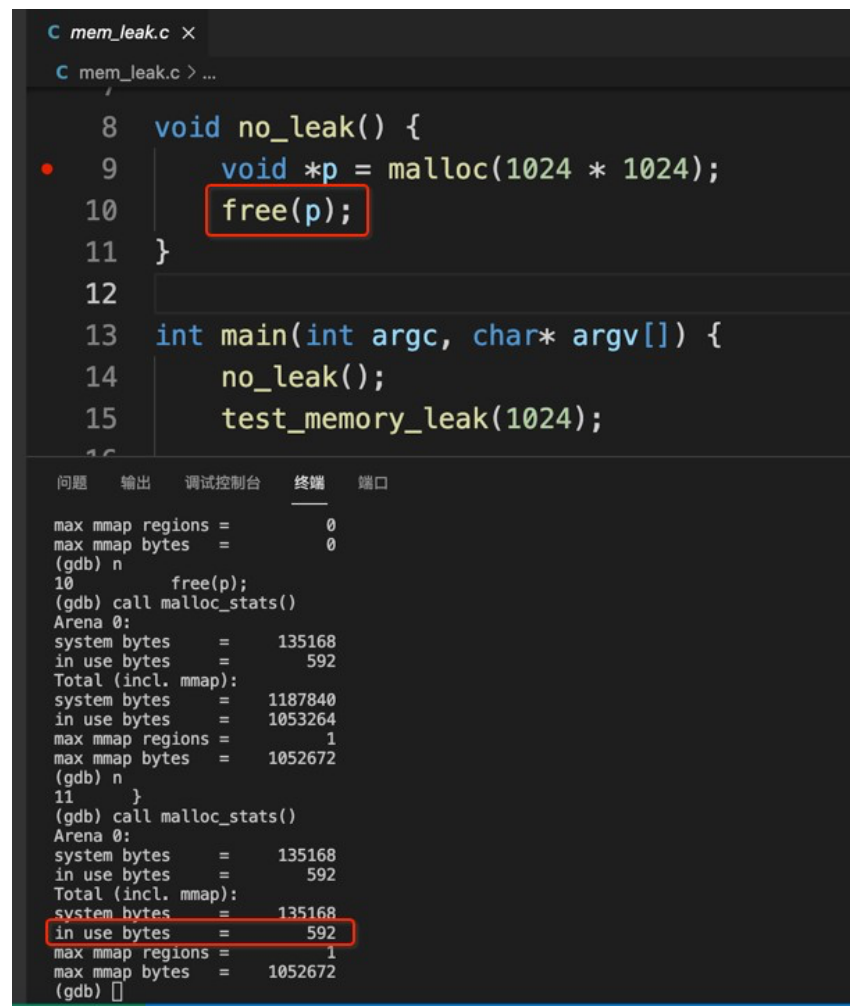
```
C mem_leak.c x
C mem_leak.c > ...

8 void no_leak() {
9     void *p = malloc(1024 * 1024);
10    free(p);
11 }
12
13 int main(int argc, char* argv[]) {
14     no_leak();
15     test_memory_leak(1024);
16
17     return 0;
}

问题 输出 调试控制台 终端 端口
Breakpoint 1, no_leak () at mem_leak.c:9
9      void *p = malloc(1024 * 1024);
(gdb) call malloc_stats()
Arena 0:
system bytes      =      0
in use bytes      =      0
Total (incl. mmap):
system bytes      =      0
in use bytes      =      0
max mmap regions  =      0
max mmap bytes    =      0
(gdb) n
10     free(p);
(gdb) call malloc_stats()
Arena 0:
system bytes      =    135168
in use bytes      =      592
Total (incl. mmap):
system bytes      =   1187840
in use bytes      =   1053264
max mmap regions  =      1
max mmap bytes    =   1052672
```

Step 4: executing the free() statement.

After executing the free() statement, you can see that the in use bytes are reduced to 592. malloc() frees the space obtained.



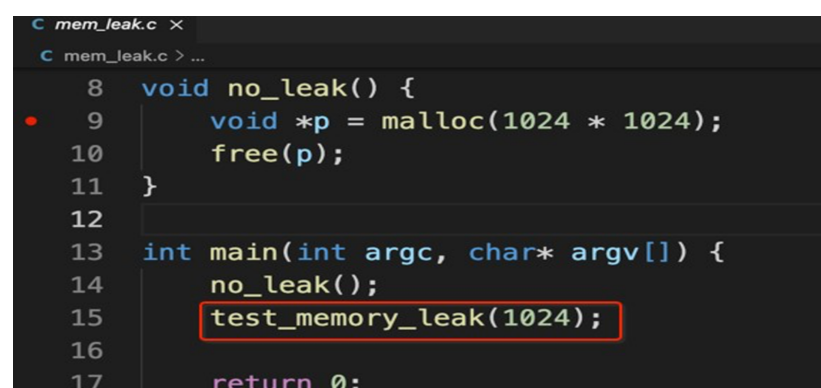
```
mem_leak.c x
mem_leak.c > ...

8 void no_leak() {
9     void *p = malloc(1024 * 1024);
10    free(p);
11 }
12
13 int main(int argc, char* argv[]) {
14     no_leak();
15     test_memory_leak(1024);
16 }

问题 输出 调试控制台 终端 端口
max mmap regions = 0
max mmap bytes = 0
(gdb) n
10 free(p);
(gdb) call malloc_stats()
Arena 0:
system bytes = 135168
in use bytes = 592
Total (incl. mmap):
system bytes = 1187840
in use bytes = 1053264
max mmap regions = 1
max mmap bytes = 1052672
(gdb) n
11 }
(gdb) call malloc_stats()
Arena 0:
system bytes = 135168
in use bytes = 592
Total (incl. mmap):
system bytes = 135168
in use bytes = 592
max mmap regions = 1
max mmap bytes = 1052672
(gdb) □
```

Step 5:

Next, execute the test_memory_leak() function, we can observe that before the execution of the function, the in use bytes is 592, after the execution is 1632, a total of 1040bytes of space is still in use and not freed, there is a memory leak.



```
mem_leak.c x
mem_leak.c > ...

8 void no_leak() {
9     void *p = malloc(1024 * 1024);
10    free(p);
11 }
12
13 int main(int argc, char* argv[]) {
14     no_leak();
15     test_memory_leak(1024);
16 }
17 return 0;
```

```
问题 输出 调试控制台 终端 端口
main (argc=1, argv=0x7fffffff038) at mem_leak.c:15
15      test_memory_leak(1024);
(gdb) call malloc_stats()
Arena 0:
system bytes      =    135168
in use bytes      =     592
Total (incl. mmap):
system bytes      =    135168
in use bytes      =     592
max mmap regions =      1
max mmap bytes    =   1052672
(gdb) n
17      return 0;
(gdb) call malloc_stats()
Arena 0:
system bytes      =    135168
in use bytes      =    1632
Total (incl. mmap):
system bytes      =    135168
in use bytes      =    1632
max mmap regions =      1
max mmap bytes    =   1052672
(gdb) █
```

Therefore, with the help of the gdb tool, it is possible to detect memory leaks by observing the change in the value of in use bytes before and after function execution.

4. Test execution by valgrind

As for mem_leak.c, execute by:

```
valgrind --leak-check=yes ./mem_leak
```

```
cs450@ubuntu:~/PA3_part1$ valgrind --leak-check=yes ./mem_leak
==8357== Memcheck, a memory error detector
==8357== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8357== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8357== Command: ./mem_leak
==8357==
==8357== HEAP SUMMARY:
==8357==   in use at exit: 1,024 bytes in 1 blocks
==8357==   total heap usage: 2 allocs, 1 frees, 1,049,600 bytes allocated
==8357==
==8357== 1,024 bytes in 1 blocks are definitely lost in loss record 1 of 1
==8357==   at 0x4C31B0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8357==   by 0x1086A1: test_memory_leak (mem_leak.c:5)
==8357==   by 0x1086EC: main (mem_leak.c:15)
==8357==
==8357== LEAK SUMMARY:
==8357==   definitely lost: 1,024 bytes in 1 blocks
==8357==   indirectly lost: 0 bytes in 0 blocks
==8357==   possibly lost: 0 bytes in 0 blocks
==8357==   still reachable: 0 bytes in 0 blocks
==8357==   suppressed: 0 bytes in 0 blocks
==8357==
==8357== For counts of detected and suppressed errors, rerun with: -v
==8357== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

execute by:

```
valgrind --xtree-leak=yes ./mem_leak
```

```
cs450@ubuntu:~/PA3_part1$ valgrind --xtree-leak=yes ./mem_leak
==5474== Memcheck, a memory error detector
==5474== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5474== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5474== Command: ./mem_leak
==5474==
==5474==
==5474== HEAP SUMMARY:
==5474==   in use at exit: 1,024 bytes in 1 blocks
==5474==   total heap usage: 2 allocs, 1 frees, 1,049,600 bytes allocated
==5474==
==5474== xtree leak report: /home/cs450/PA3_part1/xtleak.kcg.5474
==5474== LEAK SUMMARY:
==5474==   definitely lost: 1,024 bytes in 1 blocks
==5474==   indirectly lost: 0 bytes in 0 blocks
==5474==   possibly lost: 0 bytes in 0 blocks
==5474==   still reachable: 0 bytes in 0 blocks
==5474==   suppressed: 0 bytes in 0 blocks
==5474==
==5474== For counts of detected and suppressed errors, rerun with: -v
==5474== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

execute by:

```
valgrind --show-reachable=yes ./mem_leak
```

```
cs450@ubuntu:~/PA3_part1$ valgrind --show-reachable=yes ./mem_leak
==5522== Memcheck, a memory error detector
==5522== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5522== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5522== Command: ./mem_leak
==5522==
==5522==
==5522== HEAP SUMMARY:
==5522==   in use at exit: 1,024 bytes in 1 blocks
==5522==   total heap usage: 2 allocs, 1 frees, 1,049,600 bytes allocated
==5522==
==5522== LEAK SUMMARY:
==5522==   definitely lost: 1,024 bytes in 1 blocks
==5522==   indirectly lost: 0 bytes in 0 blocks
==5522==   possibly lost: 0 bytes in 0 blocks
==5522==   still reachable: 0 bytes in 0 blocks
==5522==   suppressed: 0 bytes in 0 blocks
==5522== Rerun with --leak-check=full to see details of leaked memory
==5522==
==5522== For counts of detected and suppressed errors, rerun with: -v
==5522== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

As for mem_leak2.c, The rationale for this test case is that the design of a program that points a pointer to a new space and loses ownership of the originally allocated space is one of the common, and often insidious, causes of memory leaks.

The expected result is that the initial allocation of 4096 bytes is not reclaimed.


```
valgrind --leak-check=yes ./mem_leak2
```

```
cs450@ubuntu:~/PA3_part1$ valgrind --leak-check=yes ./mem_leak2
==5574== Memcheck, a memory error detector
==5574== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5574== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5574== Command: ./mem_leak2
==5574==
==5574== HEAP SUMMARY:
==5574==   in use at exit: 4,096 bytes in 1 blocks
==5574==   total heap usage: 2 allocs, 1 frees, 5,120 bytes allocated
==5574==
==5574== 4,096 bytes in 1 blocks are definitely lost in loss record 1 of 1
==5574==   at 0x4C31B0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==5574==   by 0x10869B: test_memory_leak (mem_leak2.c:5)
==5574==   by 0x1086D5: main (mem_leak2.c:11)
==5574==
==5574== LEAK SUMMARY:
==5574==   definitely lost: 4,096 bytes in 1 blocks
==5574==   indirectly lost: 0 bytes in 0 blocks
==5574==   possibly lost: 0 bytes in 0 blocks
==5574==   still reachable: 0 bytes in 0 blocks
==5574==   suppressed: 0 bytes in 0 blocks
==5574==
==5574== For counts of detected and suppressed errors, rerun with: -v
==5574== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

```
valgrind --xtree-leak=yes ./mem_leak2
```

```
cs450@ubuntu:~/PA3_part1$ valgrind --xtree-leak=yes ./mem_leak2
==5596== Memcheck, a memory error detector
==5596== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5596== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5596== Command: ./mem_leak2
==5596==
==5596== HEAP SUMMARY:
==5596==   in use at exit: 4,096 bytes in 1 blocks
==5596==   total heap usage: 2 allocs, 1 frees, 5,120 bytes allocated
==5596==
==5596== xtree leak report: /home/cs450/PA3_part1/xtleak.kcg.5596
==5596== LEAK SUMMARY:
==5596==   definitely lost: 4,096 bytes in 1 blocks
==5596==   indirectly lost: 0 bytes in 0 blocks
==5596==   possibly lost: 0 bytes in 0 blocks
==5596==   still reachable: 0 bytes in 0 blocks
==5596==   suppressed: 0 bytes in 0 blocks
==5596==
==5596== For counts of detected and suppressed errors, rerun with: -v
==5596== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

```
valgrind --show-reachable=yes ./mem_leak2
```

```
cs450@ubuntu:~/PA3_part1$ valgrind --show-reachable=yes ./mem_leak2
==5641== Memcheck, a memory error detector
==5641== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5641== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5641== Command: ./mem_leak2
==5641==
==5641==
==5641== HEAP SUMMARY:
==5641==   in use at exit: 4,096 bytes in 1 blocks
==5641==   total heap usage: 2 allocs, 1 frees, 5,120 bytes allocated
==5641==
==5641== LEAK SUMMARY:
==5641==    definitely lost: 4,096 bytes in 1 blocks
==5641==    indirectly lost: 0 bytes in 0 blocks
==5641==    possibly lost: 0 bytes in 0 blocks
==5641==    still reachable: 0 bytes in 0 blocks
==5641==    suppressed: 0 bytes in 0 blocks
==5641== Rerun with --leak-check=full to see details of leaked memory
==5641==
==5641== For counts of detected and suppressed errors, rerun with: -v
==5641== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```