HW#3
Advanced Operating Systems, Spring 2023

MengXian,Du (CBB108047)
Department of Computer Science and Information Engineering
National Pingtung University

1. First, write a simple program called null.c that creates a pointer to an integer, sets it to NULL, and then tries to dereference it. Compile this into an executable called null. What happens when you run this program?

   Solution: Please refer to List 1 (null.c):

   Listing 1: null.c

```c
/*
First, write a simple program called null.c that creates a pointer
to an integer, sets it to NULL, and then tries to dereference it.
Compile this into an executable called null. What happens when you
run this program?
*/

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

int main()
{
    int *p = NULL;
    printf("Start\n");
    printf("The address of p is %p\n", p);
    printf("The value of p is %d\n", *p);
    printf("End\n");
    return 0;
}
```

   Its execution results are as follows:

```
Start
Segmentation fault (core dumped)
```

   As we can see the program is terminated with a segmentation fault.

   So we know that if a variable doesn't allocated memory space, it will cause a segmentation fault when it is dereferenced.

   But if we need to reference it. It's ok to reference it.

2. Next, compile this program with symbol information included (with the -g flag). Doing so let's put more information into the executable, enabling the debugger to access more useful information about variable names and the like. Run the program under the debugger by typing gdb null and then, once gdb is running, typing run. What does gdb show you?

   Solution:

   Its execution results are as follows:

```
(gdb) run
Starting program: /home/it/Desktop/Advanced_Operation_Systen/hw3/null
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Start
The address of p is (nil)
```

```
7
8 Program received signal SIGSEGV, Segmentation fault.
9 0x00005555555551ab in main () at null.c:16
10 16          printf("The value of p is %d\n", *p);
```

The gdb shows that the program is terminated with a segmentation fault.

3. Finally, use the valgrind tool on this program. We'll use the memcheck tool that is a part of valgrind to analyze what happens. Run this by typing in the following: valgrind –leak-check=yes null. What happens when you run this? Can you interpret the output from the tool?

Solution:

Its execution results are as follows:

```
1 ==20372== Memcheck, a memory error detector
2 ==20372== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
3 ==20372== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
4 ==20372== Command: ./null
5 ==20372==
6 Start
7 The address of p is (nil)
8 ==20372== Invalid read of size 4
9 ==20372==    at 0x1091AB: main (null.c:16)
10 ==20372==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
11 ==20372==
12 ==20372==
13 ==20372== Process terminating with default action of signal 11 (SIGSEGV)
14 ==20372==  Access not within mapped region at address 0x0
15 ==20372==    at 0x1091AB: main (null.c:16)
16 ==20372==  If you believe this happened as a result of a stack
17 ==20372==  overflow in your program's main thread (unlikely but
18 ==20372==  possible), you can try to increase the size of the
19 ==20372==  main thread stack using the --main-stacksize= flag.
20 ==20372==  The main thread stack size used in this run was 8388608.
21 ==20372==
22 ==20372== HEAP SUMMARY:
23 ==20372==     in use at exit: 1,024 bytes in 1 blocks
24 ==20372==   total heap usage: 1 allocs, 0 frees, 1,024 bytes allocated
25 ==20372==
26 ==20372== LEAK SUMMARY:
27 ==20372==    definitely lost: 0 bytes in 0 blocks
28 ==20372==    indirectly lost: 0 bytes in 0 blocks
29 ==20372==      possibly lost: 0 bytes in 0 blocks
30 ==20372==    still reachable: 1,024 bytes in 1 blocks
31 ==20372==         suppressed: 0 bytes in 0 blocks
32 ==20372== Reachable blocks (those to which a pointer was found) are not shown.
33 ==20372== To see them, rerun with: --leak-check=full --show-leak-kinds=all
34 ==20372==
35 ==20372== For lists of detected and suppressed errors, rerun with: -s
36 ==20372== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
37 Segmentation fault (core dumped)
```

When I run 'null' with valgrind, it shows that it can't read the memory that it's size is 4.

Address 0x0 is not stack'd, malloc'd or (recently) free'd means that the address 0x0 is not allocated and

recently freed. Also the program is terminated with a SIGSEGV signal which means a process executes a invalid memory reference.

And we can see at line 23 shows that the program has a memory block in use at exit.

More see :https://stackoverflow.com/questions/3840582/still-reachable-leak-detected-by-val

4. Write a simple program that allocates memory using malloc() but forgets to free it before exiting. What happens when this program runs? Can you use gdb to find any problems with it? How about valgrind (again with the –leak-check=yes flag)?

   Solution: Please refer to List 1 (q4.c):

   Listing 2: q4.c

```c
/*
First, write a simple program called null.c that creates a pointer
to an integer, sets it to NULL, and then tries to dereference it.
Compile this into an executable called null. What happens when you
run this program?
*/

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

int main()
{
    int *p = malloc(sizeof(int));
    printf("Start\n");
    printf("The address of p is %p\n", p);
    printf("The value of p is %d\n", *p);
    printf("End\n");
    return 0;
}
```

   Its execution results are as follows:

```
Start
The address of p is 0x55f62934c2a0
The value of p is 0
End
```

   The result use gdb is as follows:

```
(gdb) run
Starting program: /home/it/Desktop/Advanced_Operation_Systen/hw3/q4
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Start
The address of p is 0x5555555592a0
The value of p is 0
End
[Inferior 1 (process 22758) exited normally]
```

   The result use valgrind is as follows:

```
==22981== Memcheck, a memory error detector
==22981== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==22981== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==22981== Command: ./q4
==22981==
Start
The address of p is 0x4a96040
==22981== Conditional jump or move depends on uninitialised value(s)
==22981==    at 0x48E1B56: __vfprintf_internal (vfprintf-internal.c:1516)
==22981==    by 0x48CB81E: printf (printf.c:33)
==22981==    by 0x1091E8: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
==22981== Use of uninitialised value of size 8
==22981==    at 0x48C533B: _itoa_word (_itoa.c:177)
==22981==    by 0x48E0B3D: __vfprintf_internal (vfprintf-internal.c:1516)
==22981==    by 0x48CB81E: printf (printf.c:33)
==22981==    by 0x1091E8: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
==22981== Conditional jump or move depends on uninitialised value(s)
==22981==    at 0x48C534C: _itoa_word (_itoa.c:177)
==22981==    by 0x48E0B3D: __vfprintf_internal (vfprintf-internal.c:1516)
==22981==    by 0x48CB81E: printf (printf.c:33)
==22981==    by 0x1091E8: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
==22981== Conditional jump or move depends on uninitialised value(s)
==22981==    at 0x48E1643: __vfprintf_internal (vfprintf-internal.c:1516)
==22981==    by 0x48CB81E: printf (printf.c:33)
==22981==    by 0x1091E8: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
==22981== Conditional jump or move depends on uninitialised value(s)
==22981==    at 0x48E0C85: __vfprintf_internal (vfprintf-internal.c:1516)
==22981==    by 0x48CB81E: printf (printf.c:33)
==22981==    by 0x1091E8: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
The value of p is 0
End
==22981==
==22981== HEAP SUMMARY:
==22981==     in use at exit: 4 bytes in 1 blocks
==22981==   total heap usage: 2 allocs, 1 frees, 1,028 bytes allocated
==22981==
==22981== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==22981==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-
    linux.so)
==22981==    by 0x10919E: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q4)
==22981==
==22981== LEAK SUMMARY:
==22981==    definitely lost: 4 bytes in 1 blocks
==22981==    indirectly lost: 0 bytes in 0 blocks
==22981==      possibly lost: 0 bytes in 0 blocks
==22981==    still reachable: 0 bytes in 0 blocks
==22981==         suppressed: 0 bytes in 0 blocks
==22981==
==22981== Use --track-origins=yes to see where uninitialised values come from
==22981== For lists of detected and suppressed errors, rerun with: -s
==22981== ERROR SUMMARY: 6 errors from 6 contexts (suppressed: 0 from 0)

    
```

1. What happens when this program runs?

Ans : The program executes normally seems not big deal.

2. Can you use gdb to find any problems with it?

Ans : No, gdb shows the program is terminated normally.

3. How about valgrind (again with the –leak-check=yes flag)?

Ans : Yes, valgrind shows that the program has a memory leak.

5. Write a program that creates an array of integers called data of size 100 using malloc; then, set data[100] to zero. What happens when you run this program? What happens when you run this program using valgrind? Is the program correct?

Solution: Please refer to List 1 (q5.c):

Listing 3: q5.c

```c
/*
Write a program that creates an array of integers called data
of size 100 using malloc; then, set data[100] to zero. What happens
when you run this program? What happens when you run this
program using valgrind? Is the program correct?
*/


#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *data = (int *)malloc(100 * sizeof(int));
    data[100] = 0;
    printf("data[100] = %d\n", data[100]);
    return 0;
}
```

Its execution results are as follows:

```
data[100] = 0
```

Valgrind's execution results are as follows:

```
==24012== Memcheck, a memory error detector
==24012== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==24012== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==24012== Command: ./q5
==24012==
==24012== Invalid write of size 4
==24012==    at 0x10918D: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q5)
==24012==  Address 0x4a961d0 is 0 bytes after a block of size 400 alloc'd
==24012==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-
    linux.so)
==24012==    by 0x10917E: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q5)
==24012==
==24012== Invalid read of size 4
```

```
==24012==    at 0x10919D: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q5)
==24012==  Address 0x4a961d0 is 0 bytes after a block of size 400 alloc'd
==24012==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-
    linux.so)
==24012==    by 0x10917E: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q5)
==24012==
data[100] = 0
==24012==
==24012== HEAP SUMMARY:
==24012==     in use at exit: 400 bytes in 1 blocks
==24012==   total heap usage: 2 allocs, 1 frees, 1,424 bytes allocated
==24012==
==24012== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==24012==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-
    linux.so)
==24012==    by 0x10917E: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q5)
==24012==
==24012== LEAK SUMMARY:
==24012==    definitely lost: 400 bytes in 1 blocks
==24012==    indirectly lost: 0 bytes in 0 blocks
==24012==      possibly lost: 0 bytes in 0 blocks
==24012==    still reachable: 0 bytes in 0 blocks
==24012==         suppressed: 0 bytes in 0 blocks
==24012==
==24012== For lists of detected and suppressed errors, rerun with: -s
==24012== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
        
```

1. What happens when you run this program?

Ans : The program executes normally seems good no big problems.

2. What happens when you run this program using valgrind?

Ans : Valgrind shows that the program has a memory leak and invalid read and write.

3. Is the program correct?

Ans : No, the program is not correct. Becaues it use the memory that is not allocated.

6. Create a program that allocates an array of integers (as above), frees them, and then tries to print the value of one of the elements of the array. Does the program run? What happens when you use valgrind on it?

Solution: Please refer to List 1 (q6.c):

Listing 4: q6.c

```c
/*
Create a program that allocates an array of integers (as above), frees
them, and then tries to print the value of one of the elements of
the array. Does the program run? What happens when you use
valgrind on it?
*/


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

```
11 int main()
12 {
13     int *data = (int *)malloc(100 * sizeof(int));
14     free(data);
15     printf("data[0] = %d\n", data[0]);
16     return 0;
17 }
```

Its execution results are as follows:

```
1 data[0] = 1570806496
```

Valgrind's execution results are as follows:

```
1 ==24375== Command: ./q6
2 ==24375==
3 ==24375== Invalid read of size 4
4 ==24375==    at 0x1091B3: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q6)
5 ==24375==  Address 0x4a96040 is 0 bytes inside a block of size 400 free'd
6 ==24375==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-
      linux.so)
7 ==24375==    by 0x1091AE: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q6)
8 ==24375==  Block was alloc'd at
9 ==24375==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-
      linux.so)
10 ==24375==    by 0x10919E: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q6)
11 ==24375==
12 data[0] = 0
13 ==24375==
14 ==24375== HEAP SUMMARY:
15 ==24375==     in use at exit: 0 bytes in 0 blocks
16 ==24375==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
17 ==24375==
18 ==24375== All heap blocks were freed -- no leaks are possible
19 ==24375==
20 ==24375== For lists of detected and suppressed errors, rerun with: -s
21 ==24375== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

1. Does the program run?

Ans : Yes, The program can be run but the value is not controllable and predictable.

2. What happens when you use valgrind on it?

Ans : Valgrind shows that the program has no memory leak but there have a memory block has invalid read and it's size is 4 bytes.

7. Now pass a funny value to free (e.g., a pointer in the middle of the array you allocated above). What happens? Do you need tools to find this type of problem?

Solution: Please refer to List 1 (q7.c):

Listing 5: q7.c

```
1 /*
2 Now pass a funny value to free (e.g., a pointer in the middle of the
```

```
3 array you allocated above). What happens? Do you need tools to
4 find this type of problem?
5 */
6
7 #include <stdio.h>
8 #include <stdlib.h>
9 int main()
10 {
11     int *data = (int *)malloc(3 * sizeof(int));
12     data[0] = 1;
13
14     free(data + 1);
15     printf("data[0] = %d\n", data[0]);
16     return 0;
17 }
```

Its execution results are as follows:

```
1 free(): invalid pointer
2 Aborted (core dumped)
```

Valgrind's execution results are as follows:

```
1 ==25997== Command: ./q7
2 ==25997==
3 ==25997== Invalid free() / delete / delete[] / realloc()
4 ==25997==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-
      linux.so)
5 ==25997==    by 0x1091BC: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q7)
6 ==25997==  Address 0x4a96044 is 4 bytes inside a block of size 12 alloc'd
7 ==25997==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-
      linux.so)
8 ==25997==    by 0x10919E: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q7)
9 ==25997==
10 data[0] = 1
11 ==25997==
12 ==25997== HEAP SUMMARY:
13 ==25997==     in use at exit: 12 bytes in 1 blocks
14 ==25997==   total heap usage: 2 allocs, 2 frees, 1,036 bytes allocated
15 ==25997==
16 ==25997== 12 bytes in 1 blocks are definitely lost in loss record 1 of 1
17 ==25997==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-
      linux.so)
18 ==25997==    by 0x10919E: main (in /home/it/Desktop/Advanced_Operation_Systen/hw3/q7)
19 ==25997==
20 ==25997== LEAK SUMMARY:
21 ==25997==    definitely lost: 12 bytes in 1 blocks
22 ==25997==    indirectly lost: 0 bytes in 0 blocks
23 ==25997==      possibly lost: 0 bytes in 0 blocks
24 ==25997==    still reachable: 0 bytes in 0 blocks
25 ==25997==         suppressed: 0 bytes in 0 blocks
26 ==25997==
27 ==25997== For lists of detected and suppressed errors, rerun with: -s
28 ==25997== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
29
```

1. What happens?

Ans : The program is terminated with a SIGABRT signal which means a process aborts.

2. Do you need tools to find this type of problem?

Ans : Yes ,I use valgrind to find this type of problem. We can see the output of valgrind that we do a invalid free() operation. It seems we can't free a memory block that in the middle of a series of memory that allocated.