第二次作业

1.首先，编写一个名为 `null.c` 的简单程序，它创建一个指向整数的指针，将其设置为NULL，然后尝试对其进行释放内存操作。把它编译成一个名为 `null` 的可执行文件。当你运行这个程序时会发生什么?

`null.c`的代码如下：

```
#include<stdlib.h>
int main()
{
    int * p = NULL;
    free(p);
    return 0;
}
```

使用`gcc -o null null.c`编译，使用`./null`运行可执行文件

无任何输出或者错误提示

2.接下来，编译该程序，其中包含符号信息（使用-g 标志）。这样做可以将本多信息放入可执行文件中，使调试器可以但问有关变量名称等的本多有用信息。通过输入 `gdb null`，在调试器下运行该程序，然后，一旦 gdb 运行，输入 `run`。gdb 显示什么信息?

执行结果：

```
(gdb) run
Starting program: /home/szh/os2/null
[Inferior 1 (process 3073) exited normally]
```

3.编写一个使用 `malloc()` 来分配内存的简单程序，但在退出之前忘记释放它。这个程序运行时会发生什么？你可以用 gdb 来查找它的任何问题吗？用 `valgrind` 呢（再次使用 `--leak-check=yes` 标志）？

该题代码为：

```c
#include<stdlib.h>
int main()
{
    int * p = (int *)malloc(sizeof(int));
    return 0;
}
```

gdb执行结果

```
(gdb) run
Starting program: /home/szh/os2/14.4
[Inferior 1 (process 3090) exited normally]
```

看不出问题。
`valgrind`执行结果：

```
szh@ubuntu:~/os2$ sudo valgrind --leak-check=yes ./14.4
==14427== Memcheck, a memory error detector
==14427== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==14427== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==14427== Command: ./14.4
==14427==
==14427==
==14427== HEAP SUMMARY:
==14427==     in use at exit: 4 bytes in 1 blocks
==14427==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==14427==
==14427== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==14427==    at 0x483C855: malloc (vg_replace_malloc.c:393)
==14427==    by 0x10915E: main (14.4.c:4)
==14427==
==14427== LEAK SUMMARY:
==14427==    definitely lost: 4 bytes in 1 blocks
==14427==    indirectly lost: 0 bytes in 0 blocks
==14427==      possibly lost: 0 bytes in 0 blocks
==14427==    still reachable: 0 bytes in 0 blocks
==14427==         suppressed: 0 bytes in 0 blocks
==14427==
==14427== For lists of detected and suppressed errors, rerun with: -s
==14427== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

检测出了内存泄漏问题

4.编写一个程序，使用 `malloc` 创建一个名为 `data`、大小为 100 的整数数组。然后，将`data[100]`设置为 0。当你运行这个程序时会发生什么？当你使用 `valgrind` 运行这个程序时会发生什么？程序是否正确？

代码为：

```c
#include<stdlib.h>
int main()
{
    int * data = (int *)malloc(100*sizeof(int));
    data[100] = 0;
    free(data);
    return 0;
}
```

gdb执行结果：

```
(gdb) run
Starting program: /home/szh/os2/14.5
[Inferior 1 (process 14461) exited normally]
```

看不出问题
`valgrind`执行结果：

```
szh@ubuntu:~/os2$ sudo valgrind --leak-check=yes ./14.5
==14468== Memcheck, a memory error detector
==14468== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==14468== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==14468== Command: ./14.5
==14468==
==14468== Invalid write of size 4
==14468==    at 0x10918D: main (14.5.c:5)
==14468==  Address 0x4a531d0 is 0 bytes after a block of size 400 alloc'd
==14468==    at 0x483C855: malloc (vg_replace_malloc.c:393)
==14468==    by 0x10917E: main (14.5.c:4)
==14468==
==14468==
==14468== HEAP SUMMARY:
==14468==     in use at exit: 0 bytes in 0 blocks
==14468==   total heap usage: 1 allocs, 1 frees, 400 bytes allocated
==14468==
==14468== All heap blocks were freed -- no leaks are possible
==14468==
==14468== For lists of detected and suppressed errors, rerun with: -s
==14468== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

检测到无效写入错误

5.创建一个分配整数数组的程序（如上所述），释放它们，然后尝试打印数组中某个元素的值。程序会运行吗？当你使用 `valgrind` 时会发生什么？

代码为

```c
#include<stdlib.h>
#include<stdio.h>
int main()
{
    int * p = (int *)malloc(100*sizeof(int));
    p[1] = 10;
    p[90] = 20;
    free(p);
    printf("%d %d\n", p[1],p[90]);
    return 0;
}
```

执行结果：

```
szh@ubuntu:~/os2$ ./14.6
0 20
```

在调用`free`函数释放内存之后，已经释放的内存不再属于程序所占用的内存区域，访问这些内存的结果是未定义的。这意味着，程序可能会输出0和20，也可能会输出一些随机的值，甚至可能会导致程序崩溃。

`valgrind`执行结果：

```
szh@ubuntu:~/os2$ sudo valgrind --leak-check=yes ./14.6
==14525== Memcheck, a memory error detector
==14525== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==14525== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==14525== Command: ./14.6
==14525==
==14525== Invalid read of size 4
==14525==    at 0x1091D7: main (14.6.c:9)
==14525==  Address 0x4a531a8 is 360 bytes inside a block of size 400 free'd
==14525==    at 0x483F0C3: free (vg_replace_malloc.c:884)
==14525==    by 0x1091CC: main (14.6.c:8)
==14525==  Block was alloc'd at
==14525==    at 0x483C855: malloc (vg_replace_malloc.c:393)
==14525==    by 0x10919E: main (14.6.c:5)
==14525==
==14525== Invalid read of size 4
==14525==    at 0x1091E1: main (14.6.c:9)
==14525==  Address 0x4a53044 is 4 bytes inside a block of size 400 free'd
==14525==    at 0x483F0C3: free (vg_replace_malloc.c:884)
==14525==    by 0x1091CC: main (14.6.c:8)
==14525==  Block was alloc'd at
==14525==    at 0x483C855: malloc (vg_replace_malloc.c:393)
==14525==    by 0x10919E: main (14.6.c:5)
==14525==
10 20
==14525==
==14525== HEAP SUMMARY:
==14525==     in use at exit: 0 bytes in 0 blocks
==14525==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==14525==
==14525== All heap blocks were freed -- no leaks are possible
```

```
==14525==
==14525== For lists of detected and suppressed errors, rerun with: -s
==14525== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

检测到了两个无效读取