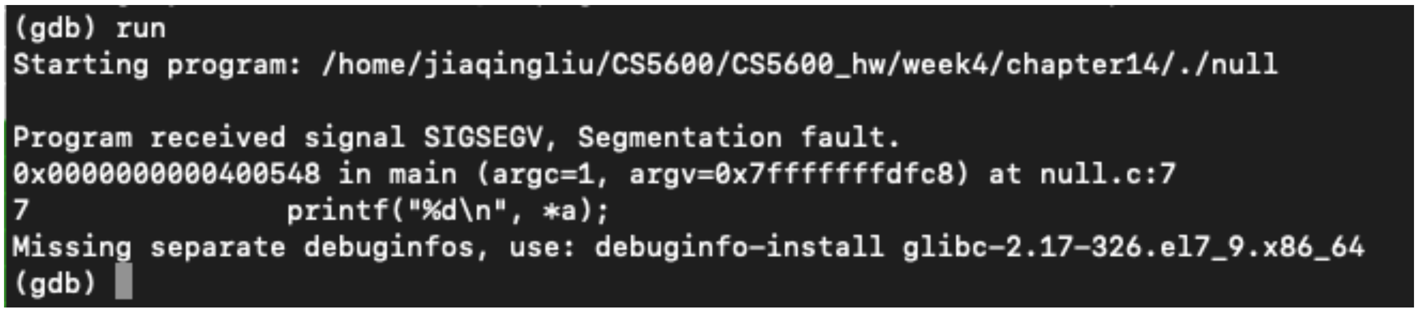
1. **First, write a simple program called null.c that creates a pointer to an integer, sets it to NULL, and then tries to dereference it. Compile this into an executable called null. What happens when you run this program?**

Segmentation fault

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?**

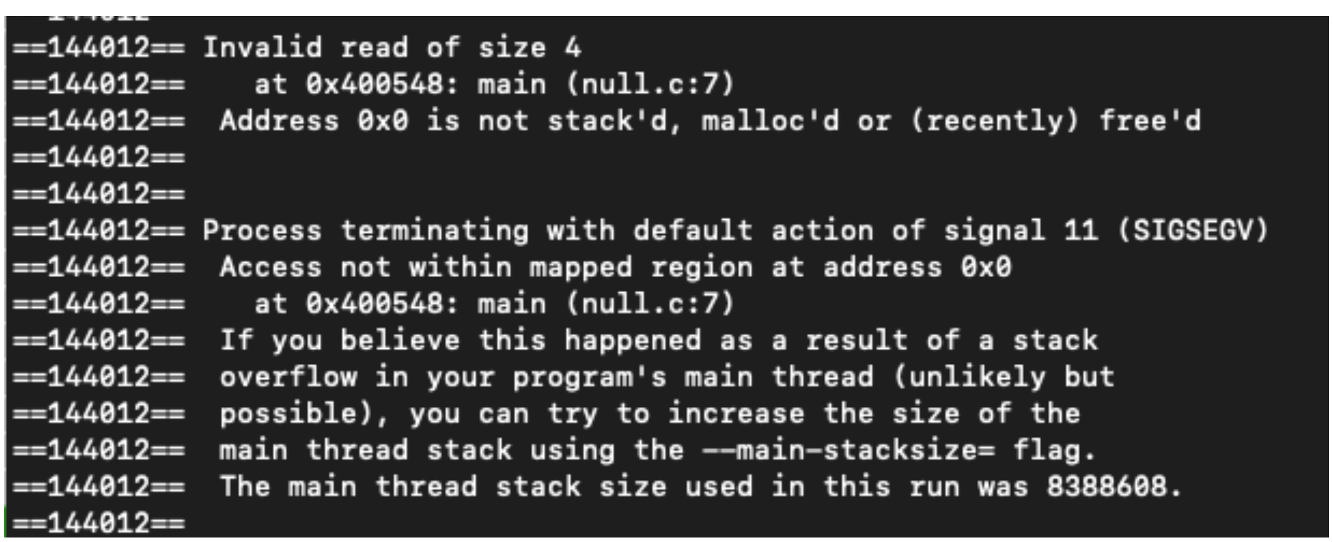


```
(gdb) run
Starting program: /home/jiaqingliu/CS5600/CS5600_hw/week4/chapter14/./null

Program received signal SIGSEGV, Segmentation fault.
0x0000000000400548 in main (argc=1, argv=0x7fffffffdfc8) at null.c:7
7               printf("%d\n", *a);
Missing separate debuginfos, use: debuginfo-install glibc-2.17-326.el7_9.x86_64
(gdb) 
```

Because of the printf causing the segmentation fault, and receive the signal SIGSEGV.

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?**



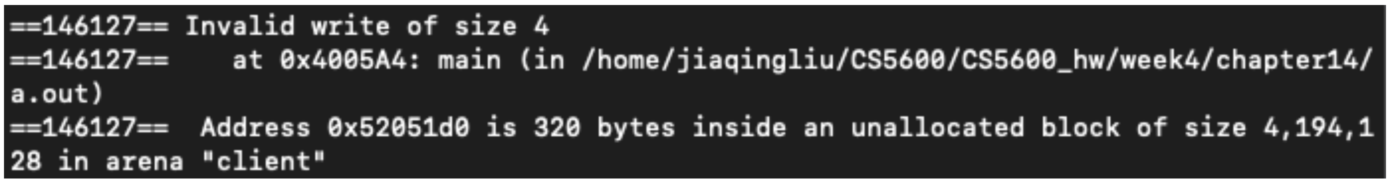
```
==144012== Invalid read of size 4
==144012==    at 0x400548: main (null.c:7)
==144012==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==144012==
==144012==
==144012== Process terminating with default action of signal 11 (SIGSEGV)
==144012==  Access not within mapped region at address 0x0
==144012==    at 0x400548: main (null.c:7)
==144012==  If you believe this happened as a result of a stack
==144012==  overflow in your program's main thread (unlikely but
==144012==  possible), you can try to increase the size of the
==144012==  main thread stack using the --main-stacksize= flag.
==144012==  The main thread stack size used in this run was 8388608.
==144012==
```

The program trying the access memory address which is not allocate by OS. Thus, the access is denied.

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)?

Gdb can not find any issure. Valgrind can find memory leak of 4 bytes integer, because we forget to free it.

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?**

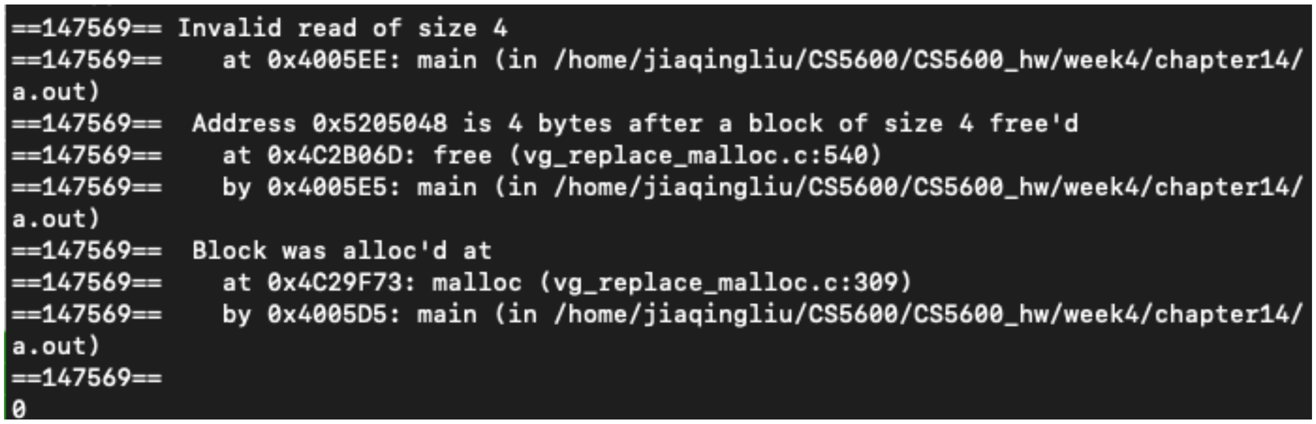


```
==146127== Invalid write of size 4
==146127==    at 0x4005A4: main (in /home/jiaqingliu/CS5600/CS5600_hw/week4/chapter14/
a.out)
==146127==  Address 0x52051d0 is 320 bytes inside an unallocated block of size 4,194,1
28 in arena "client"
```

We should use sizeof(int) not sizeof(100), because sizeof(100) is 4 bytes. When we access array[100], we actually outside the currently process virtual memory space.

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?**

Yes, it runs, and print out 0 which we never assign to it. When we use valgrind, it detected the errors, that we are trying to read freed array and also outside it original range.



```
==147569== Invalid read of size 4
==147569==    at 0x4005EE: main (in /home/jiaqingliu/CS5600/CS5600_hw/week4/chapter14/
a.out)
==147569==  Address 0x5205048 is 4 bytes after a block of size 4 free'd
==147569==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==147569==    by 0x4005E5: main (in /home/jiaqingliu/CS5600/CS5600_hw/week4/chapter14/
a.out)
==147569==  Block was alloc'd at
==147569==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==147569==    by 0x4005D5: main (in /home/jiaqingliu/CS5600/CS5600_hw/week4/chapter14/
a.out)
==147569==
0
```

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?**

The complier find out we are trying to free the integer, but free suppose only receive void pointers. And the error is throw.

8. **Try out some of the other interfaces to memory allocation. For example, create a simple vector-like data structure and related routines that use realloc() to manage the vector. Use an array to store the vectors elements; when a user adds an entry to the vector, use realloc() to allocate more space for it. How well does such a vector perform? How does it compare to a linked list? Use valgrind to help you find bugs.**

Base on my implementation, it will realloc once excessed the capacity, the performance is really bad compare to linked list. Because each time realloc we need copy paste entire old array into new address.