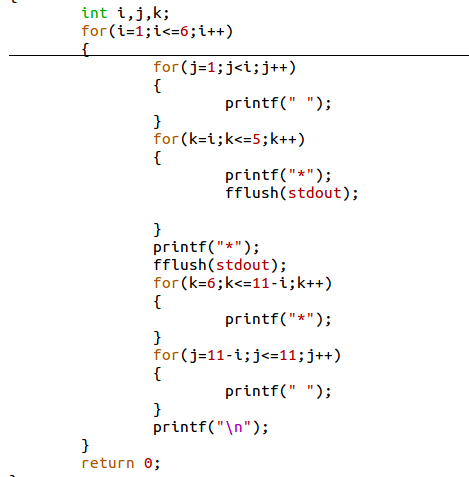
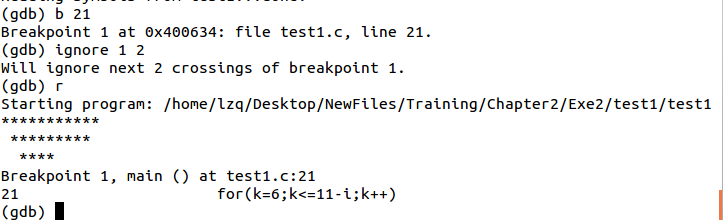
子任务1：

test1.c文件如下：

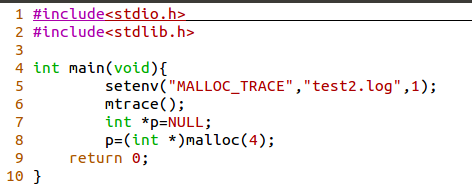


gdb设置断点

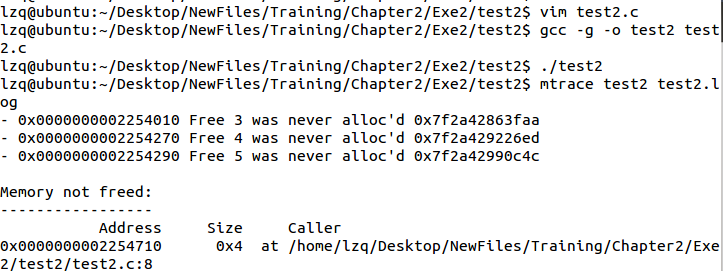


子任务2：

编写一个内存泄漏的C程序：



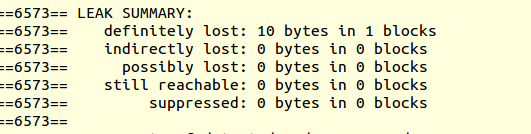
使用mtrace测试内存泄漏：（问题出现在第8行，malloc的空间没有free掉）



测试3.c的文件，使用g++ -g -o test3 3.c然后./test3一下并没有报错

使用valgrind调试一下valgrind --tool=memcheck --leak-check=full ./test3

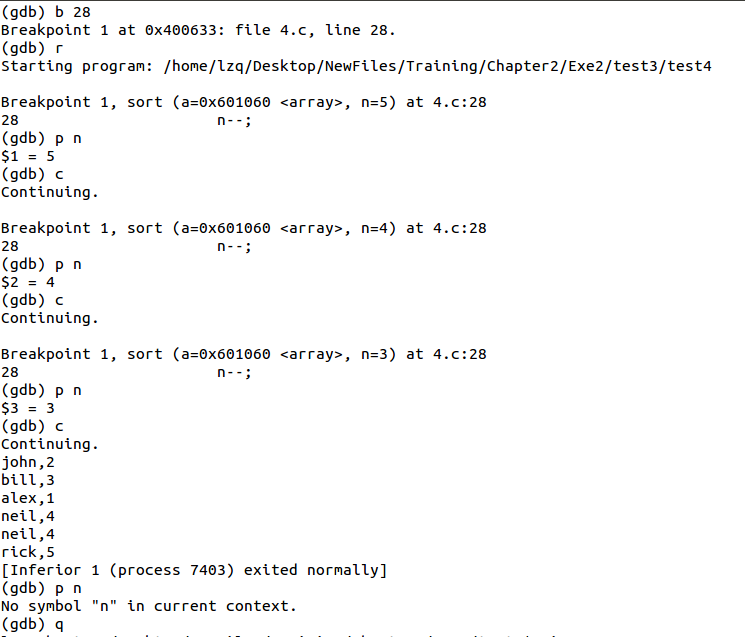
结果如下



可以看到有内存泄漏！

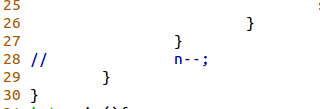
子任务3：

调试漏洞程序4.c：

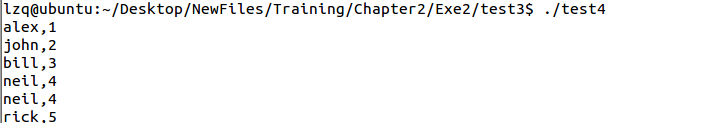


n到3时程序就结束了，排序并没有执行完整。

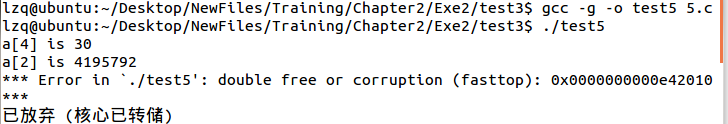
修改源代码：（把28行的n--注释掉（或者删掉都行））



运行就没问题了：

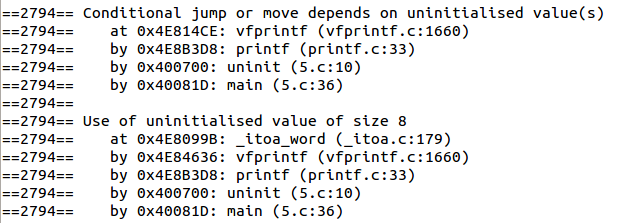


测试第二个程序5.c：

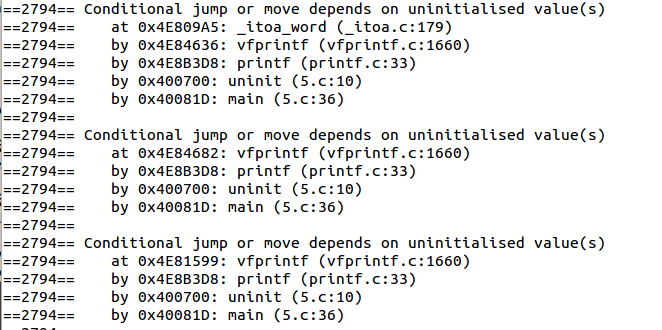


问题挺多。。

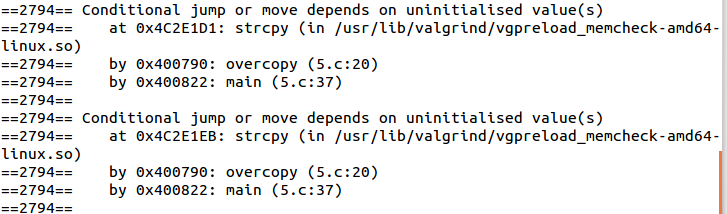
使用valgrind调试：（valgrind --tool=memcheck --leak-check=full ./test5）



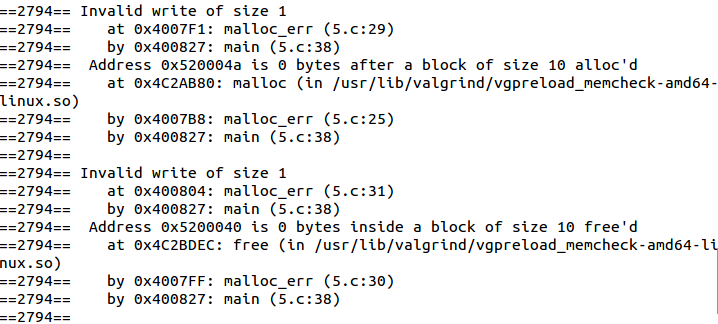
在第10行处使用了未初始化的变量，看源代码发现a[3]未初始化就被引用了。



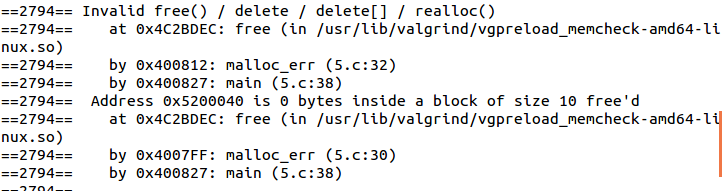
上图看出访问了未初始化变量的空间，查看源代码发现a[6]出现越界访问，数组没有那么大空间。



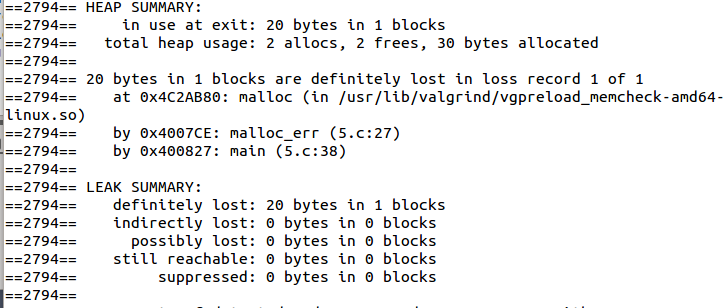
这里也是对未初始化的空间进行了访问，在源代码中发现dst[20]未初始化。



在29、31行处出现非法写入，p1[10]=5对未分配的某个不知道的空间非法写入，31行的\*p1=10是因为之前free掉自己的这块空间了（free(p2)），所以也产生了对未分配空间的非法写入。



在32行出现free的错误，是因为p1和p2指向的是同一地址空间，free(p2)后再去执行free(p1)自然会出错。



在27行申请的空间没有free，导致内存泄漏。

修改如下：

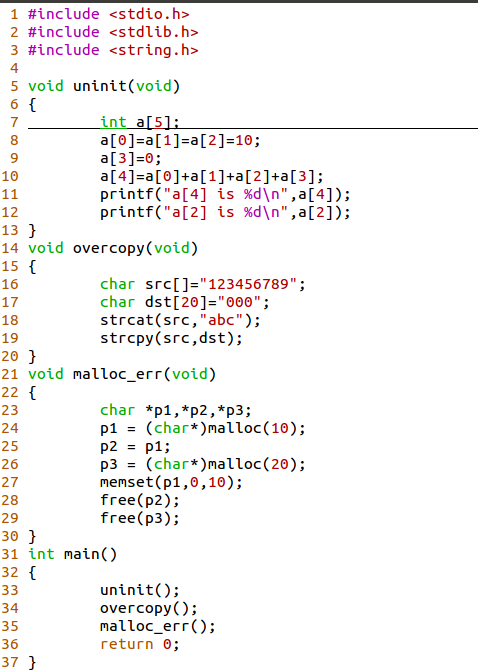
a[3]上调了几行，即先初始化；

a[2]=a[6]越界访问，直接删掉；

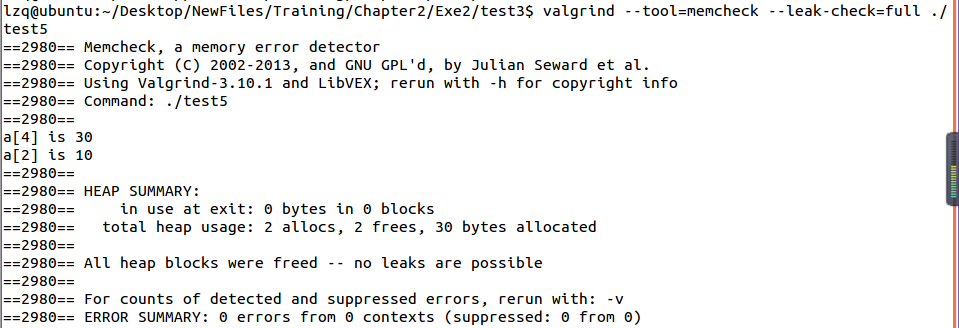
dst[20]线初始化一下；

p1[10]=5；\*p1=10；也删掉；

最后的free(p1);改为free(p3);



再用valgrind调试一下：



没问题了！