1、为什么要使用 Valgrind

内存泄漏的产生：内存泄漏（Memory Leak）是指程序中已动态分配的堆内存由于某种原因程序未释放或无法释放，造成系统内存的浪费，导致程序运行速度减慢甚至系统崩溃等严重后果！
Valgrind使用的目的：在人为编写代码不能预防消除内存泄漏的情况下，使用该工具对代码进行检测。

2、安装 Valgrind

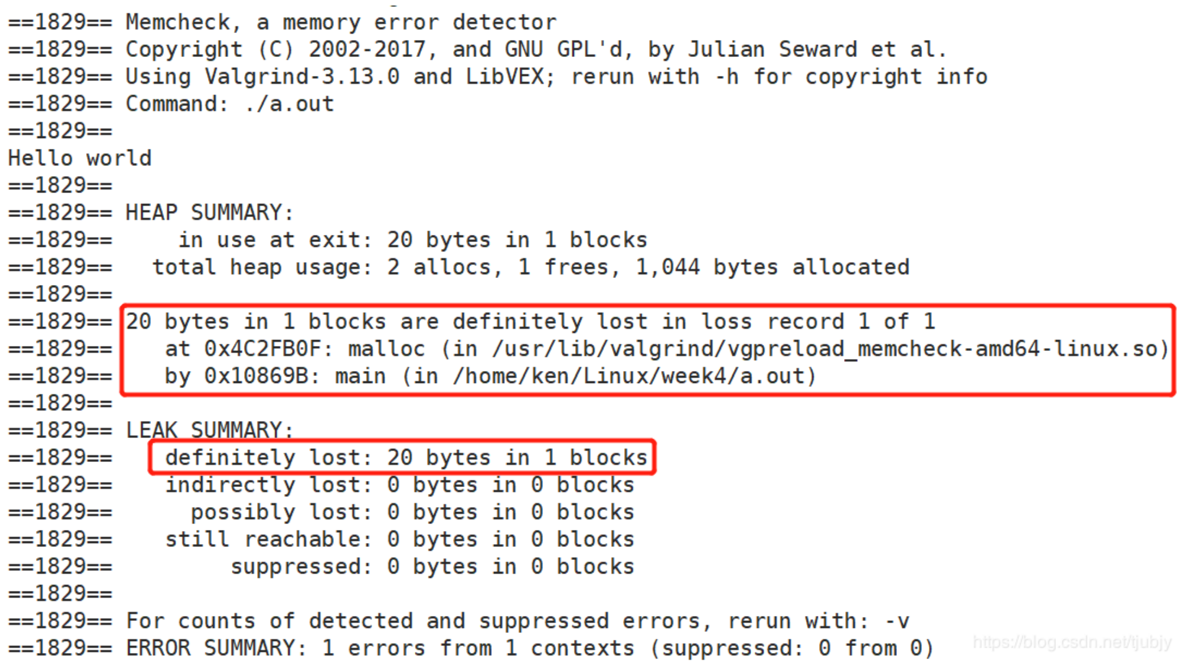
```
sudo apt-get install valgrind
```

3、如何使用

编写如下程序malloc.c

```
#include <stdlib.h>
int main()
{
    void *p;
    p=malloc(20);
    return 0;
}
```

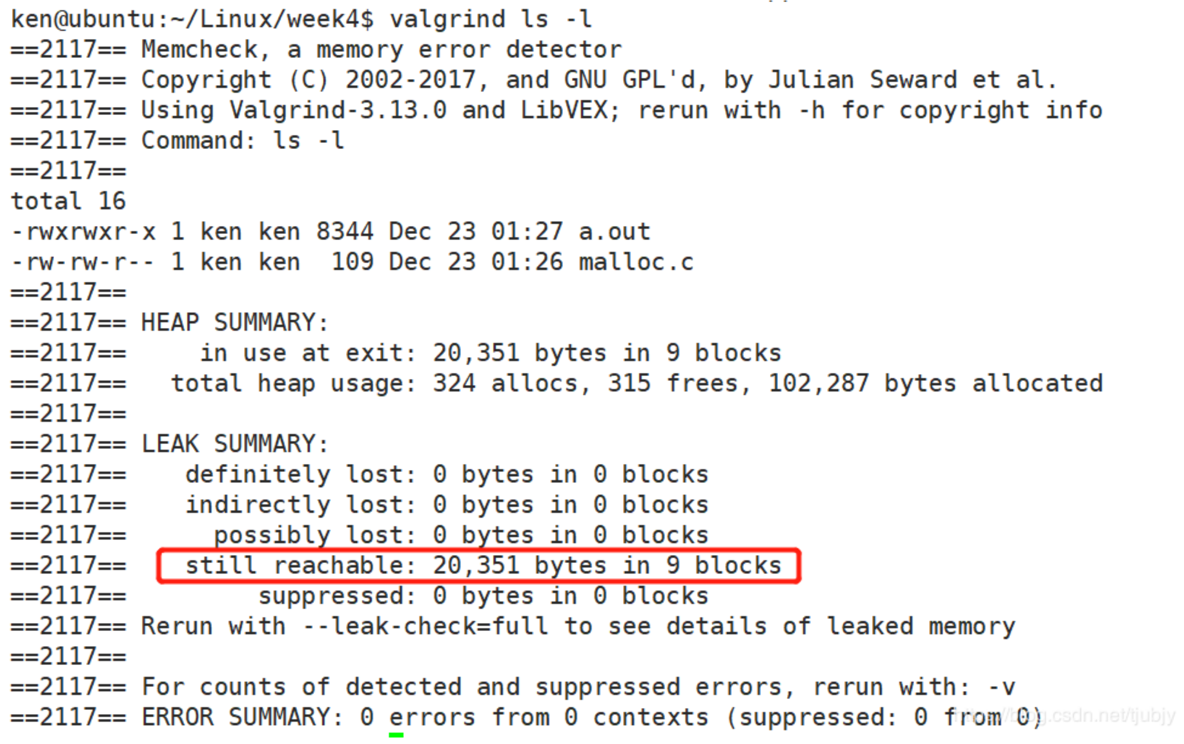
然后通过 `gcc -g malloc.c`，加入-g参数的目的是为了定位在第几行
执行 `valgrind --tool=memcheck --leak-check=full ./a.out`得到如下结果



```
==1829== Memcheck, a memory error detector
==1829== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1829== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1829== Command: ./a.out
==1829==
Hello world
==1829==
==1829== HEAP SUMMARY:
==1829==     in use at exit: 20 bytes in 1 blocks
==1829==   total heap usage: 2 allocs, 1 frees, 1,044 bytes allocated
==1829==
==1829== 20 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1829==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==1829==    by 0x10869B: main (in /home/ken/Linux/week4/a.out)
==1829==
==1829== LEAK SUMMARY:
==1829==    definitely lost: 20 bytes in 1 blocks
==1829==    indirectly lost: 0 bytes in 0 blocks
==1829==      possibly lost: 0 bytes in 0 blocks
==1829==    still reachable: 0 bytes in 0 blocks
==1829==         suppressed: 0 bytes in 0 blocks
==1829==
==1829== For counts of detected and suppressed errors, rerun with: -v
==1829== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

“definitely lost”：确认丢失。程序中存在内存泄露，应尽快修复。当程序结束时如果一块动态分配的内存没有被释放且通过程序内的指针变量均无法访问这块内存则会报这个错误。
“indirectly lost”：间接丢失。当使用了含有指针成员的类或结构时可能会报这个错误。这类错误无需直接修复，他们总是与”definitely lost”一起出现，只要修复”definitely lost”即可。例子可参考我的例程。
“possibly lost”：可能丢失。大多数情况下应视为与”definitely lost”一样需要尽快修复，除非你的程序让一个指针指向一块动态分配的内存（但不是这块内存起始地址），然后通过运算得到这块内存起始地址，再释放它。当程序结束时如果一块动态分配的内存没有被释放且通过程序内的指针变量均无法访问

这块内存的起始地址，但可以访问其中的某一部分数据，则会报这个错误。
“still reachable”：可以访问，未丢失但也未释放。如果程序是正常结束的，那么它可能不会造成程序崩溃，但长时间运行有可能耗尽系统资源，因此笔者建议修复它。如果程序是崩溃（如访问非法的地址而崩溃）而非正常结束的，则应当暂时忽略它，先修复导致程序崩溃的错误，然后重新检测。
“suppressed”：已被解决。出现了内存泄露但系统自动处理了。可以无视这类错误。这类错误我没能用例程触发，看官方的解释也不太清楚是操作系统处理的还是valgrind，也没有遇到过。所以无视他吧~

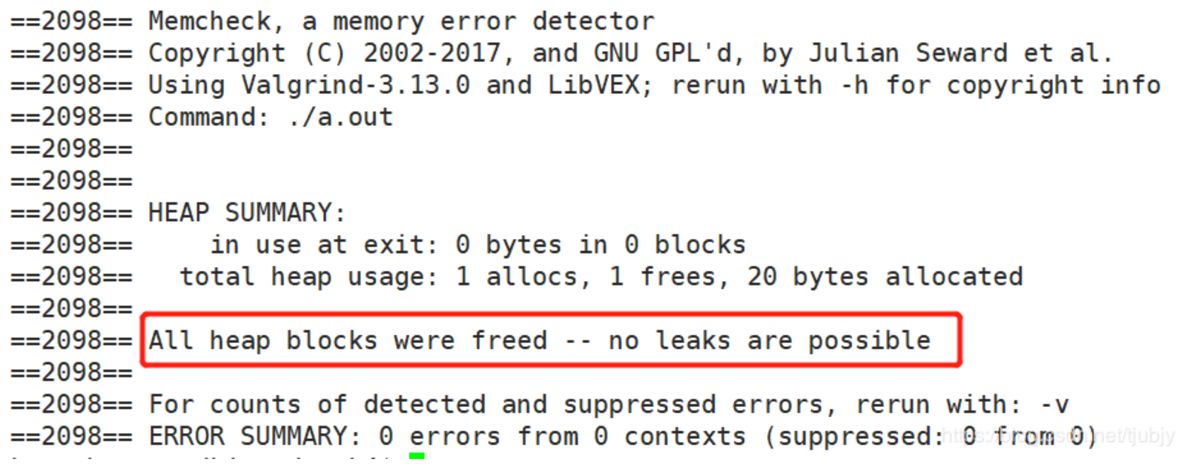


```
ken@ubuntu:~/Linux/week4$ valgrind ls -l
==2117== Memcheck, a memory error detector
==2117== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2117== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2117== Command: ls -l
==2117==
total 16
-rwxrwxr-x 1 ken ken 8344 Dec 23 01:27 a.out
-rw-rw-r-- 1 ken ken  109 Dec 23 01:26 malloc.c
==2117==
==2117== HEAP SUMMARY:
==2117==     in use at exit: 20,351 bytes in 9 blocks
==2117==   total heap usage: 324 allocs, 315 frees, 102,287 bytes allocated
==2117==
==2117== LEAK SUMMARY:
==2117==    definitely lost: 0 bytes in 0 blocks
==2117==    indirectly lost: 0 bytes in 0 blocks
==2117==      possibly lost: 0 bytes in 0 blocks
==2117==    still reachable: 20,351 bytes in 9 blocks
==2117==         suppressed: 0 bytes in 0 blocks
==2117== Rerun with --leak-check=full to see details of leaked memory
==2117==
==2117== For counts of detected and suppressed errors, rerun with: -v
==2117== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

修改程序为，并再次`gcc -g malloc.c`进行编译

```
#include <stdlib.h>

int main()
{
    void *p;
    p=malloc(20);
    free(p);
    return 0;
}
```



```
==2098== Memcheck, a memory error detector
==2098== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2098== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2098== Command: ./a.out
==2098==
==2098==
==2098== HEAP SUMMARY:
==2098==     in use at exit: 0 bytes in 0 blocks
==2098==   total heap usage: 1 allocs, 1 frees, 20 bytes allocated
==2098==
==2098== All heap blocks were freed -- no leaks are possible
==2098==
==2098== For counts of detected and suppressed errors, rerun with: -v
==2098== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

没有泄露！

4、valgrind还能干什么

- Memcheck。这是valgrind应用最广泛的工具，一个重量级的内存检查器，能够发现开发中绝大多数内存错误使用情况，比如：使用未初始化的内存，使用已经释放了的内存，内存访问越界等。
- Callgrind。它主要用来检查程序中函数调用过程中出现的问题。
- Cachegrind。它主要用来检查程序中缓存使用出现的问题。
- Helgrind。它主要用来检查多线程程序中出现的竞争问题。
- Massif。它主要用来检查程序中堆栈使用中出现的问题。
- Extension。可以利用core提供的功能，自己编写特定的内存调试工具