系统开发工具基础实验报告（四）

22110032031 张希敏

2024 年 9 月 16 日

目录

1 练习主题

（1）调试及性能分析

（2）元编程演示实验

（3）大杂烩

（4）PyTorch 编程

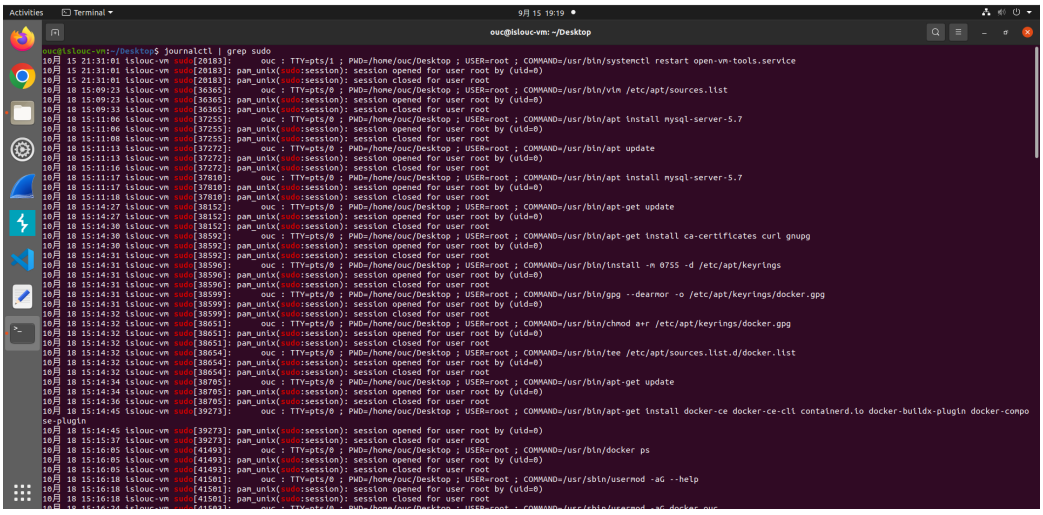
2 练习内容

2.1 四个题目

2.1.1 题目一

使用 Linux 上的 journalctl 或 macOS 上的 log show 命令来获取最近一天中超级用户的登录信息及其所执行的指令。

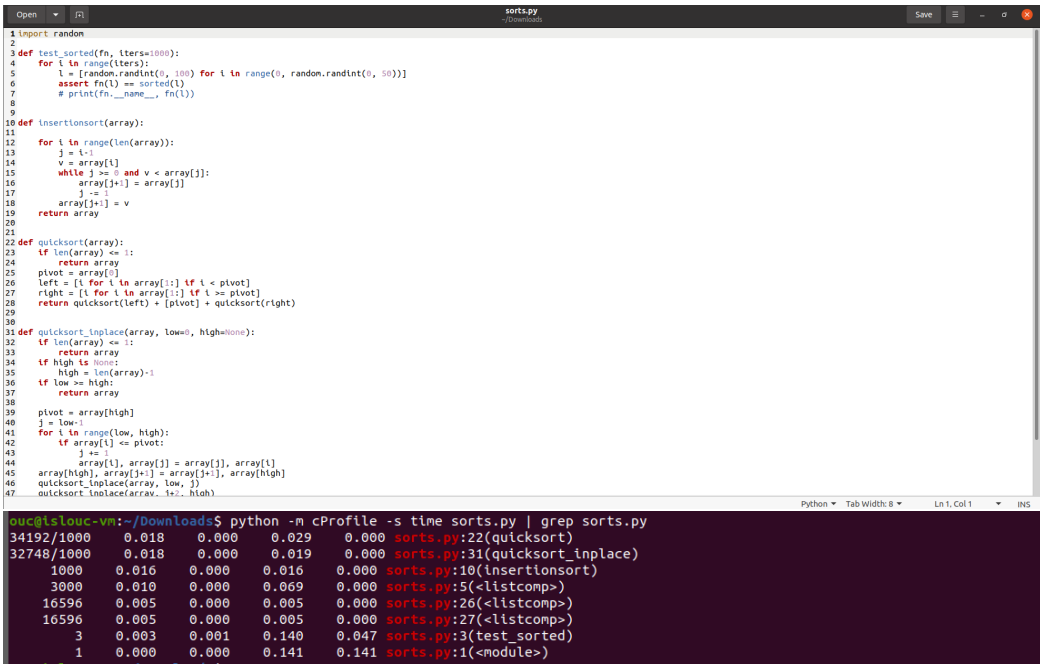
答: 输入命令 journalctl | grep sudo



2.1.2 题目二

请使用 cProfile 和 line_profiler 来比较插入排序和快速排序的性能。两种算法的瓶颈分别在哪里？然后使用 memory_profiler 来检查内存消耗，为什么插入排序更好一些？然后再看看原地排序版本的快排。

答： 使用 cProfile 按照执行时间排序，输入命令 python -m cProfile -s time sorts.py | grep sorts.py



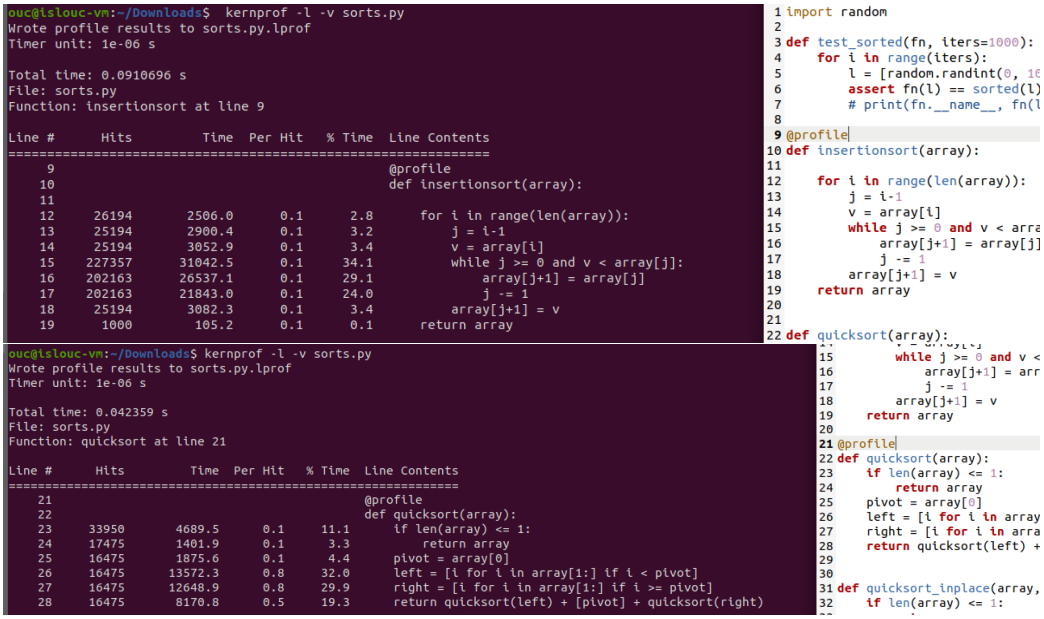
使用 line_profiler 进行分析，需要安装：

pip install line_profiler

然后为需要分析的函数添加装饰器 @profile，并执行：

kernprof -l -v sorts.py

结果如图：

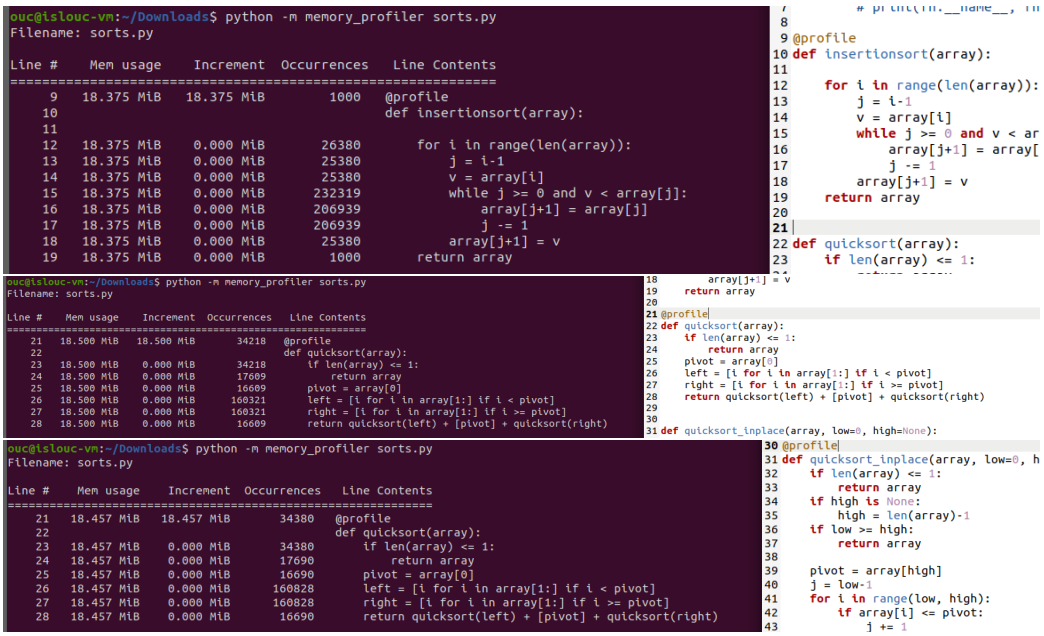


插入排序的耗时更高一些。快速排序的瓶颈在于 left 和 right 的赋值，而插入排序的瓶颈在 while 循环。

使用 memory_profiler 进行分析，需要安装：

pip install memory_profiler

同样需要添加 @profile 装饰器。结果如图：

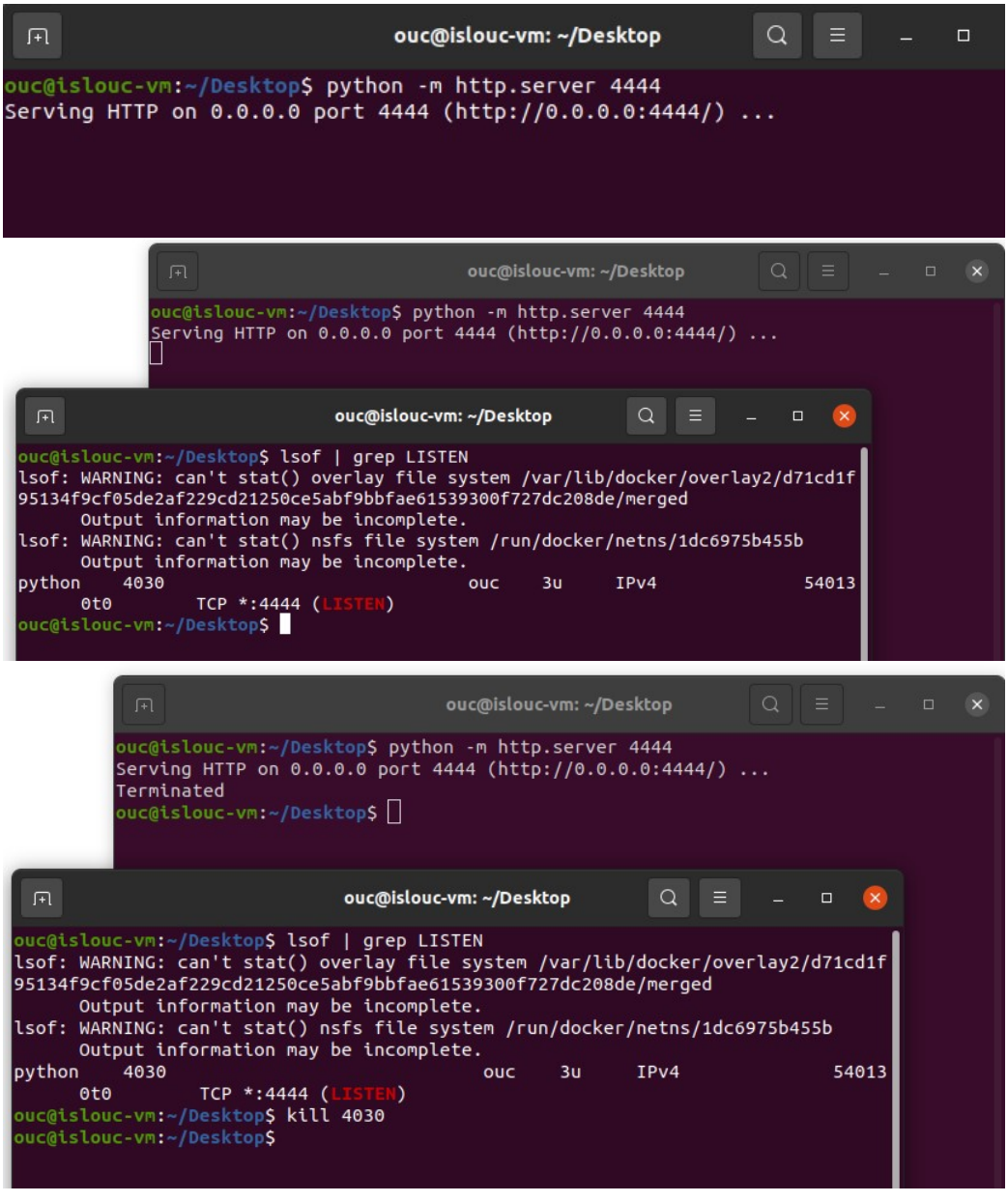


由于 test_sorted 用于测试的 list 太小了，插入排序和快速排序的内存占用相近。理论上插入排序占用的内存更小，因为插入排序的空间复杂度是 O(1)，即它是原地排序，不需要额外的存储空间；而快速排序在递归过程中会使用到系统栈空间，其空间复杂度在平均情况下为 O(log n)，最坏情况下为 O(n)。这种差异在处理大数据集时会变得更明显。

2.1.3 题目三

我们经常会遇到的情况是某个我们希望去监听的端口已经被其他进程占用了。让我们通过进程的 PID 查找相应的进程。首先执行 python -m http.server 4444 启动一个最简单的 web 服务器来监听 4444 端口。在另外一个终端中，执行 lsof | grep LISTEN 打印出所有监听端口的进程及相应的端口。找到对应的 PID 然后使用 kill <PID> 停止该进程。

答: 结果如图。



2.1.4 题目四

大多数的 makefiles 都提供了一个名为 clean 的构建目标，这并不是说我们会生成一个名为 clean 的文件，而是我们可以使用它清理文件，让 make 重新构建。你可以理解为它的作用是“撤销”所有构建步骤。在上面的 makefile 中为 paper.pdf 实现一个 clean 目标。你需要将构建目标设置为

phony。

答： 编写 makefile，如下：

```
paper.pdf: paper.tex plot-data.png
	pdflatex paper.tex

plot-%.png: %.dat plot.py
	./plot.py -i $*.dat -o $@

.PHONY: clean
clean:
	mkdir -p Untrack
	rm -f *~ .*~
	git ls-files -o | grep -v Untrack | xargs -r mv -u -t Untrack
```

将所有 untracked 文件移动到 Untrack 目录下，达到清理 git 仓库的目的。还需要设置 git 忽略该目录，用来放置 untracked 文件：

```
ouc@islouc-vm:~/Desktop/gjjc/missing-semester-cn.github.io$ cat .gitignore
.ruby-version
.bundle/
_site/
.jekyll-metadata
untrack
```

结果如图：

```
ouc@islouc-vm:~/Desktop/gjjc/missing-semester-cn.github.io$ make
pdflatex paper.tex
This is pdfTeX, Version 3.14159265-2.6-1.40.20 (TeX Live 2019/Debian) (preloaded format=pdflatex)
 restricted \write18 enabled.
entering extended mode
(./paper.tex
LaTeX2e <2020-02-02> patch level 2
L3 programming layer <2020-02-14>
(/usr/share/texlive/texmf-dist/tex/latex/base/article.cls
Document Class: article 2019/12/20 v1.4l Standard LaTeX document class
(/usr/share/texlive/texmf-dist/tex/latex/base/size12.clo))
(/usr/share/texlive/texmf-dist/tex/latex/l3backend/l3backend-pdfmode.def)
No file paper.aux.
[1{/usr/share/texlive/texmf-dist/fonts/map/pdftex/updmap/pdftex.map}]
(./paper.aux) )</usr/share/texlive/texmf-dist/fonts/type1/public/amsfonts/cm/cm
r12.pfb></usr/share/texlive/texmf-dist/fonts/type1/public/amsfonts/cm/cmr17.pfb
>
Output written on paper.pdf (1 page, 17738 bytes).
Transcript written on paper.log.
ouc@islouc-vm:~/Desktop/gjjc/missing-semester-cn.github.io$ make clean
mkdir -p Untrack
rm -f *~ .*~
git ls-files -o | grep -v Untrack | xargs -r mv -u -t Untrack
ouc@islouc-vm:~/Desktop/gjjc/missing-semester-cn.github.io$ ls
_2019  404.html  CNAME        favicon.ico  Gemfile.lock  index.md  lectures.html  paper.tex      README.md   static
_2020  about.md  _config.yml  Gemfile      _includes     _layouts  license.md     plot-data.png  robots.txt  Untrack
ouc@islouc-vm:~/Desktop/gjjc/missing-semester-cn.github.io$ cd untrack
bash: cd: untrack: No such file or directory
ouc@islouc-vm:~/Desktop/gjjc/missing-semester-cn.github.io$ cd Untrack
ouc@islouc-vm:~/Desktop/gjjc/missing-semester-cn.github.io/Untrack$ ls
makefile  paper.aux  paper.log  paper.pdf  paper.tex  plot-data.png
```

2.2 20 个实例

（1）向系统日志中写日志： 使用 logger 并且如何找到能够将其存入系统日志的条目。

```
ouc@islouc-vm:~/Desktop/gjjc$ logger "Hello Logs"
ouc@islouc-vm:~/Desktop/gjjc$ journalctl --since "1m ago" | grep Hello
9月 15 21:35:02 islouc-vm ouc[4377]: Hello Logs
```

（2）gdb： disas 命令查看反汇编代码。这里以 bomb 作为实例，gdb bomb, disas phase_2 直接查看第 2 个炸弹执行函数的反汇编代码。

```
Reading symbols from bomb...done.
(gdb) disas phase_2
Dump of assembler code for function phase_2:
   0x0000000000400efc <+0>:     push   %rbp
   0x0000000000400efd <+1>:     push   %rbx
   0x0000000000400efe <+2>:     sub    $0x28,%rsp
   0x0000000000400f02 <+6>:     mov    %rsp,%rsi
   0x0000000000400f05 <+9>:     callq  0x40145c <read_six_numbers>
   0x0000000000400f0a <+14>:    cmpl   $0x1,(%rsp)
   0x0000000000400f0e <+18>:    je     0x400f30 <phase_2+52>
   0x0000000000400f10 <+20>:    callq  0x40143a <explode_bomb>
   0x0000000000400f15 <+25>:    jmp    0x400f30 <phase_2+52>
   0x0000000000400f17 <+27>:    mov    -0x4(%rbx),%eax
   0x0000000000400f1a <+30>:    add    %eax,%eax
   0x0000000000400f1c <+32>:    cmp    %eax,(%rbx)
   0x0000000000400f1e <+34>:    je     0x400f25 <phase_2+41>
   0x0000000000400f20 <+36>:    callq  0x40143a <explode_bomb>
   0x0000000000400f25 <+41>:    add    $0x4,%rbx
   0x0000000000400f29 <+45>:    cmp    %rbp,%rbx
   0x0000000000400f2c <+48>:    jne    0x400f17 <phase_2+27>
   0x0000000000400f2e <+50>:    jmp    0x400f3c <phase_2+64>
   0x0000000000400f30 <+52>:    lea    0x4(%rsp),%rbx
   0x0000000000400f35 <+57>:    lea    0x18(%rsp),%rbp
   0x0000000000400f3a <+62>:    jmp    0x400f17 <phase_2+27>
   0x0000000000400f3c <+64>:    add    $0x28,%rsp
   0x0000000000400f40 <+68>:    pop    %rbx
   0x0000000000400f41 <+69>:    pop    %rbp
   0x0000000000400f42 <+70>:    retq
End of assembler dump.
(gdb)
```

（3）gdb： 在某一位置打断点并使程序运行到该断点处停止。这里以 bomb 作为实例，b read_six_numbers 在 read_six_numbers 函数入口处打上断点。r 运行，程序运行到 read_six_number 断点处自动停止。

```
The program is not being run.
(gdb) r
Starting program: /data/workspace/myshixun/step1/bomb
BFD: /usr/lib/debug/.build-id/9d/a9b30d436da7f984fc9904fbea9ab251f0c401.debug: u
nable to initialize decompress status for section .debug_aranges
BFD: /usr/lib/debug/.build-id/9d/a9b30d436da7f984fc9904fbea9ab251f0c401.debug: u
nable to initialize decompress status for section .debug_aranges
warning: File "/usr/lib/debug/.build-id/9d/a9b30d436da7f984fc9904fbea9ab251f0c40
1.debug" has no build-id, file skipped
BFD: /usr/lib/debug/.build-id/a6/5a82147cd98619525049d4c29cc45f88d1fb00.debug: u
nable to initialize decompress status for section .debug_aranges
BFD: /usr/lib/debug/.build-id/a6/5a82147cd98619525049d4c29cc45f88d1fb00.debug: u
nable to initialize decompress status for section .debug_aranges
warning: File "/usr/lib/debug/.build-id/a6/5a82147cd98619525049d4c29cc45f88d1fb0
0.debug" has no build-id, file skipped
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
aaa

Breakpoint 1, 0x000000000040145c in read_six_numbers ()
```

（4）gdb: 输入指令 l (list)，显示当前行的上、下若干行 C 语句的内容

```
(gdb) l
23      #include <stdio.h>
24      #include <stdlib.h>
25      #include "support.h"
26      #include "phases.h"
27
28      /*
29       * Note to self: Remember to erase this file so my victims will have no
30       * idea what is going on, and so they will all blow up in a
31       * spectaculary fiendish explosion. -- Dr. Evil
32       */
(gdb)
```

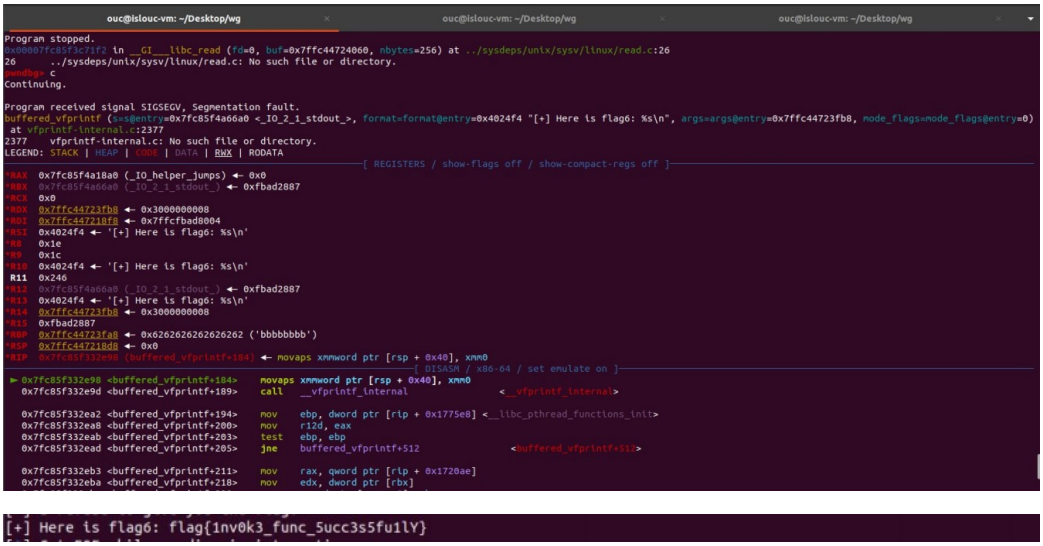
（5）gdb: 输入指令 n i/step i，时程序单步向前执行 i 条机器指令

```
(gdb) n 1
69          printf("Welcome to my fiendish little bomb. You have 6 phases with\n");
```

（6）gdb: 输入指令 n/step，单步执行 C 语句

```
(gdb) b 26
Breakpoint 1 at 0x80489db: file bomb.c, line 26.
(gdb) r
Starting program: /home/a/桌面/bomb/bomb

Breakpoint 1, main (argc=1, argv=0xffffd134) at bomb.c:37
37      {
(gdb) n
45          if (argc == 1) {
(gdb) n
46              infile = stdin;
(gdb) n
67          initialize_bomb();
```

（7）pwndbg: 安装 pwndbg 并进行调试。pwndbg 在 GDB 的基础上，增加了针对二进制安全、漏洞利用和 CTF 挑战等特定场景的功能和命令，如 ROP 链构造、内存分析等。这里以逆向 pwn 文件 game 为例，文件已上传至 github。这里在试图覆盖返回地址获取 flag 时出现了段错误，用 pwndbg 调试查看栈和报错信息，查找并加入合适的 gadget 后实现栈平衡，再次运行，成功获得了 flag。



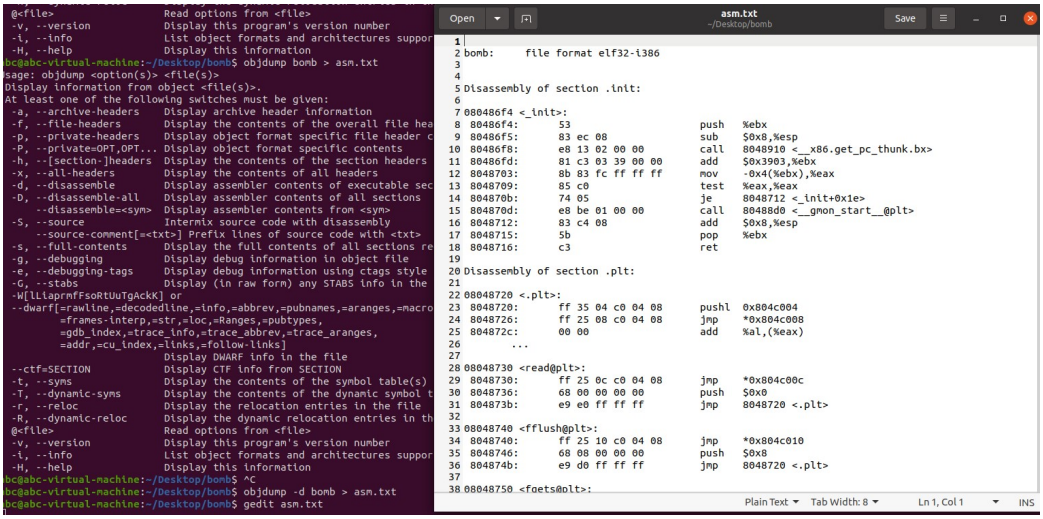
（8）objdump 的使用: 使用 objdump 反汇编 bomb 得到汇编语言代码

答: objdump -d bomb > asm.txt

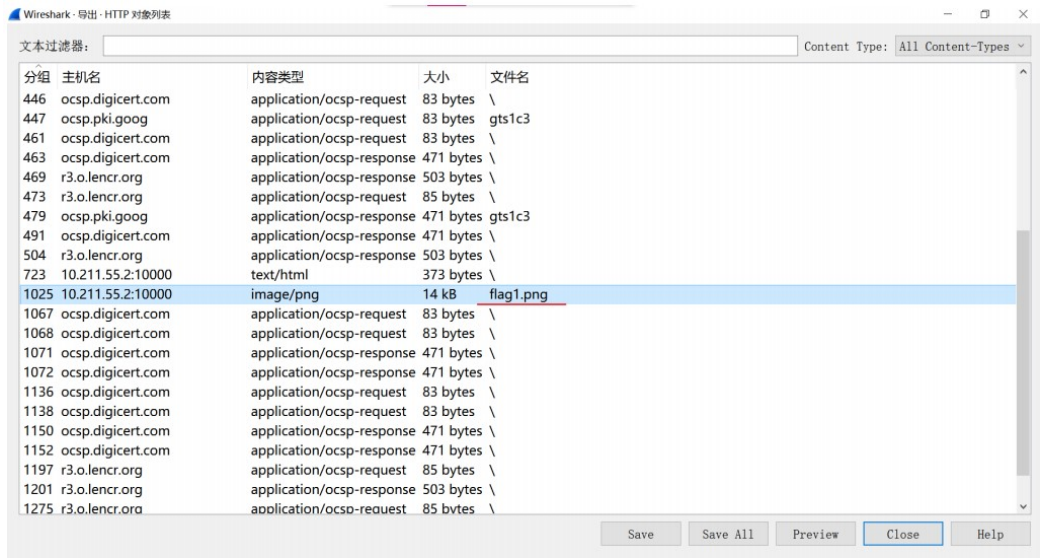
“>”：重定向，将反汇编出来的源程序输出至 asm.txt 文件中.

打开反汇编源代码文件

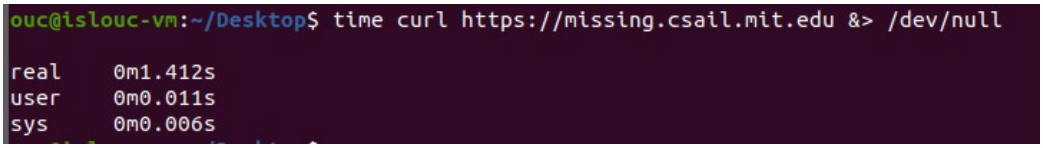
gedit asm.txt



（9）Wireshark： Wireshark 这样的网络数据包分析工具可以帮助用户获取网络数据包的内容并基于不同的条件进行过滤。以 you-got-me.pcapng 为例，用 wireshark 分析，过滤 http 对象，文件列表中发现 flag1.png, 导出图片即得 flag。文件已上传至 github。



10）性能分析-计时： 执行一个用于发起 HTTP 请求的命令，通过 time 命令查看该命令执行的 CPU 内核时间、CPU 用户时间和真实时间，



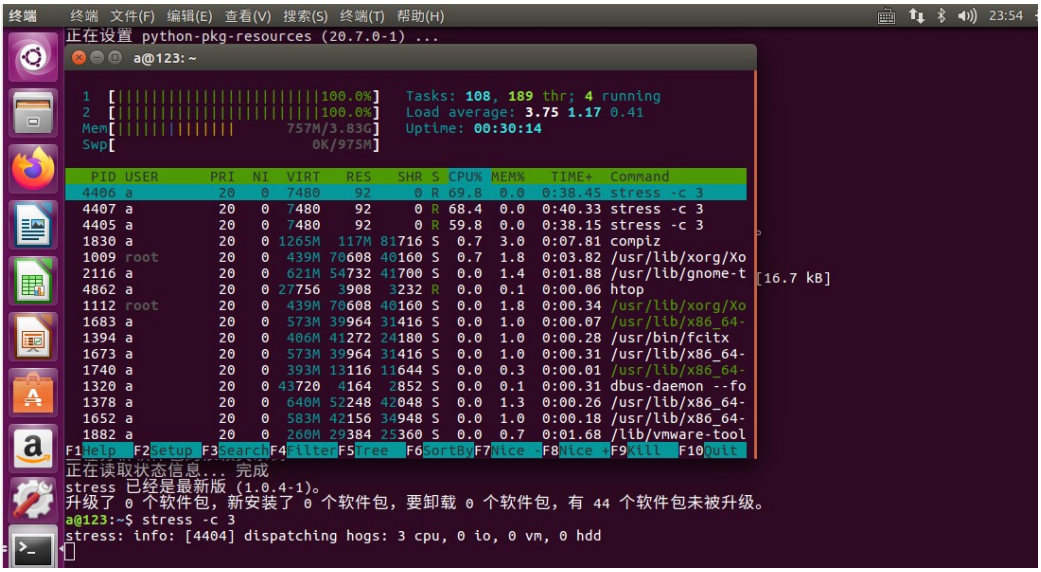
（11）perf： perf 可以报告不佳的缓存局部性（poor cache locality）、大量的页错误（page faults）或活锁（livelocks）。以 sort.py 为例，使用 perf 检查每个插入排序算法的循环次数、缓存命中和丢失。

```
50
51 if __name__ == '__main__':
52     test_sorted(insertionsort)
```

```
a@123:~/桌面/bomb$ sudo perf stat -e cycles,cache-references,cache-misses python3 sorts.py

 Performance counter stats for 'python3 sorts.py':

       99,149,068      cycles
                0      cache-references
                0      cache-misses              #    0.000 % of all cache refs

      0.054595422 seconds time elapsed
```

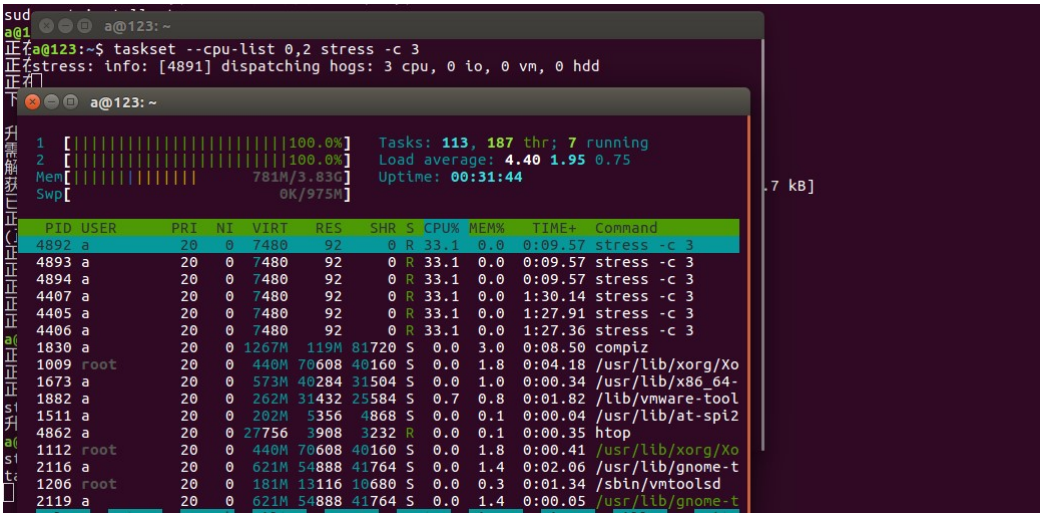
（12）可视化： 这里有一些用于计算斐波那契数列 Python 代码，它为计算每个数字都定义了一个函数。将代码拷贝到文件中使其变为一个可执行的程序。首先安装 pycallgraph 和 graphviz。并使用 pycallgraph graphviz – ./fib.py 来执行代码并查看 pycallgraph.png 这个文件。

（13）进程资源的可视化： 限制进程资源也是一个非常有用的技术。执行 stress -c 3 创建负载并使用 htop 对 CPU 消耗进行可视化。



（14）进程资源的可视化： 执行 taskset –cpu-list 0,2 stress -c 3 限制资源消耗，并可视化。stress 占用了 3 个 CPU 吗？为什么没有？

答： 运行 taskset –cpu-list 0,2 stress -c 3 时限制了 stress 进程只能在 CPU 0 和 CPU 2 上运行。也就是说，请求了 3 个 CPU 密集型任务，但只指定了两个可用的 CPU 核心。阅读 man taskset 可知，如果进程被限制在少于请求数量的 CPU 核心上，那么这些核心将尽可能平均地分担负载。但是，由于只有 2 个核心可用，所以实际上只有 2 个 stress 的 CPU 密集型任务能够并行运行。所以 stress 没有占用 3 个 CPU 。



（15）命令行标志参数： 使用 rm 删除一个叫 -r 的文件。

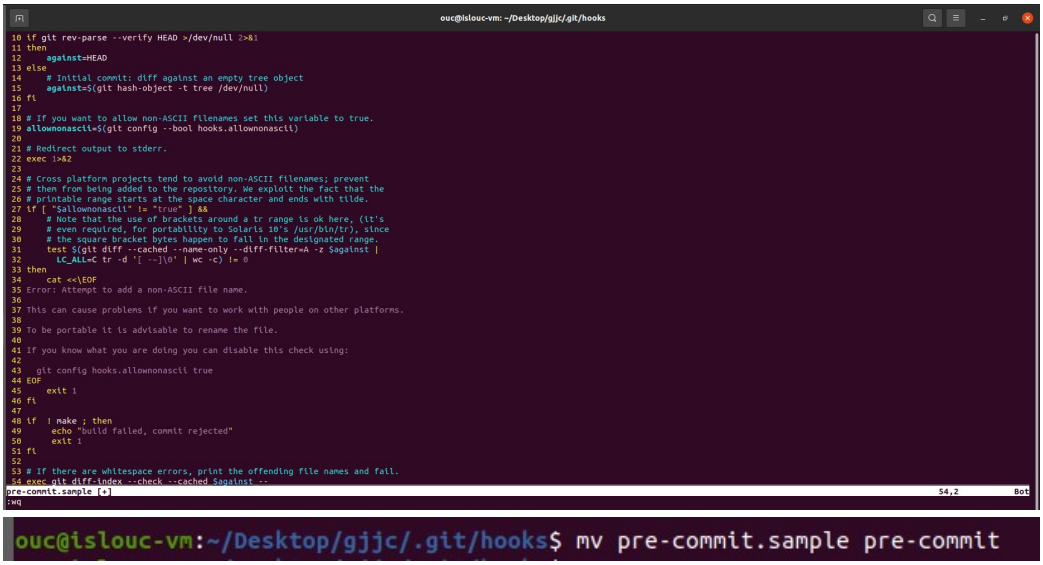
```
ouc@islouc-vm:~/Downloads$ vim -- -r
ouc@islouc-vm:~/Downloads$ ls
fib.py  makefile  paper.tex  plot-data.png  pwndbg  -r  sorts.py  sorts.py.lprof
ouc@islouc-vm:~/Downloads$ rm -- -r
ouc@islouc-vm:~/Downloads$ ls
fib.py  makefile  paper.tex  plot-data.png  pwndbg  sorts.py  sorts.py.lprof
```

（16）git： Git 可以作为一个简单的 CI 系统来使用，在任何 git 仓库中的.git/hooks 目录中，您可以找到一些文件（当前处于未激活状态），它们的作用和脚本一样，当某些事件发生时便可以自动执行。请编写一个 pre-commit 钩子，它会在提交前执行 make paper.pdf 并在出现构建失败的情况拒绝您的提交。

答： 修改.git/hooks 目录下面的 pre-commit.sample 文件，加入以下几行：

```
if ! make ; then
    echo "build failed, commit rejected"
    exit 1
    fi
```

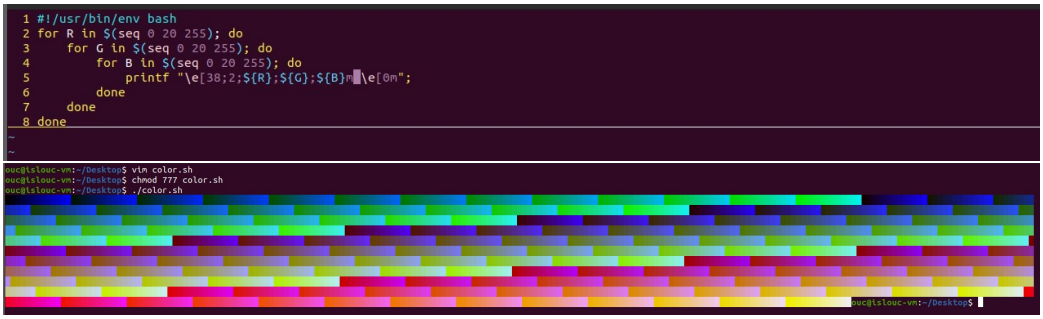
然后将其命名为 pre-commit



（17）可读性： 以彩色文本显示终端信息时可读性更好。执行命令打印红色，测试用户的终端是否支持真彩色。

（18）彩色文本： 在终端中打印多种颜色。

答： 编写脚本如下，运行脚本，查看终端的彩色输出。可以通过调整 RGB 值来获取输出文本的不同颜色。



（19）markdown： 安装 markdown 编写软件 marktext 并写一个文件，转成 pdf 格式。

（文件已上传至 github）


F: › 课程归档 › 数据结构与算法 › md实例.md
数据结构与算法实验一
实验题目：多项式乘法问题
实验目的：
设计一个一元稀疏多项式简单计算器。
实验内容与要求：
W 1871
File
Edit
Paragraph
Format
Window
Theme
View
Help
New Tab Ctrl+T
New Window Ctrl+N
Open File... Ctrl+O
Open Folder... Ctrl+Shift+O
Open Recent
Save Ctrl+S
Save As... Ctrl+Shift+S
Auto Save
Move To...
Rename...
Import...
Export
Print
Preferences... Ctrl+逗号
Close Tab Ctrl+W
Close Window Ctrl+Shift+W
Quit Ctrl+Q
HTML
PDF
md实例.md
md实例.pdf

（20）pytorch 实例： 在 pycharm 写 Python 程序对图片进行 Sobel 算子卷积操作与空间域滤波。

实例要求： 将 Sobel 算子编码到 pytorch 卷积核中，并用编码的卷积核对图像 ori.jpg 执行卷积操作，输出结果（水平梯度图像、垂直梯度图像和梯度幅值图像），理解卷积操作与空间域滤波的关系。

实例解答： (1) 加载图像，转换为灰度图像，方便创立 tensor 数据；升维，添加 batch 维度。

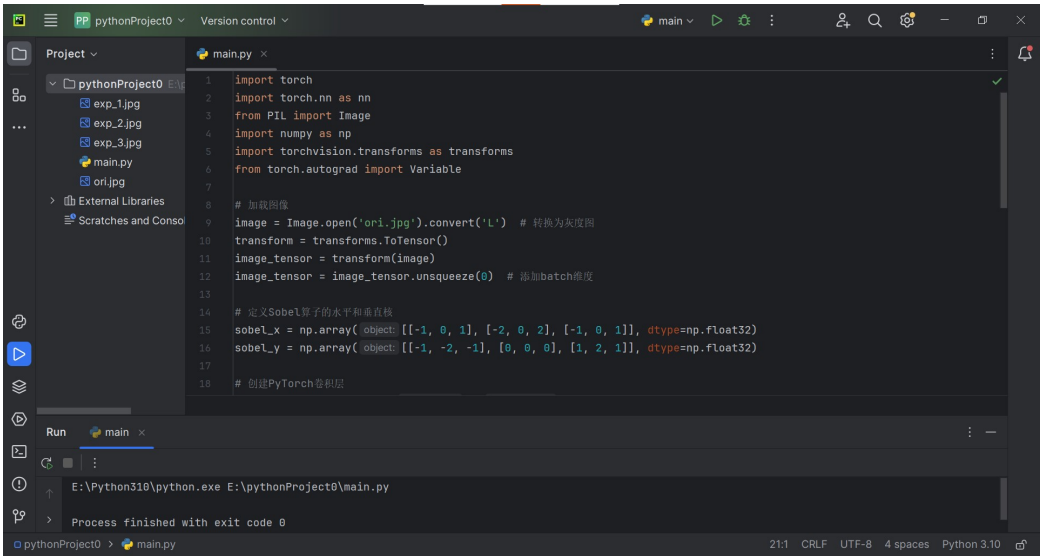
(2) 定义 Sobel 算子的水平和垂直核，创建 PyTorch 卷积层，设置卷积核权重。

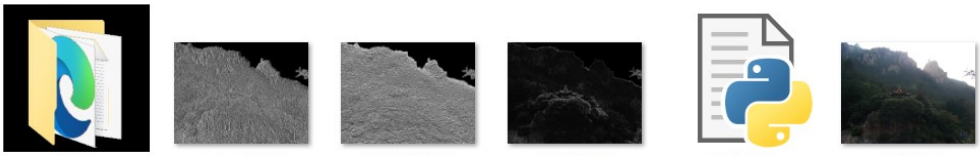
(3) 对图像应用水平和垂直 Sobel 滤波器得到水平梯度数据和垂直梯度数据；进一步计算梯度幅值图像。

(4) 将上一步数据更改到 [0, 255] 范围并转换为 uint8 类型，以便保存为图像文件。

(5) 保存结果。

（具体代码及输出结果已上传至 github 库-task4-py0）



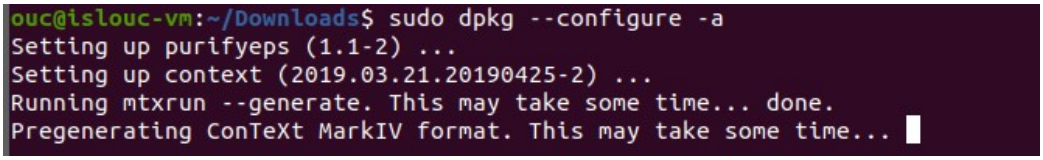


3 问题及解决方案

（1） 问题：运行 perf 时报错。

```
ouc@islouc-vm:~/Downloads$ perf stat -e cycles,cache-references,cache-misses python3 sorts.py

Command 'perf' not found, but can be installed with:

sudo apt install linux-intel-iotg-5.15-tools-common    # version 5.15.0-1064.70~20.04.1, or
sudo apt install linux-oem-5.6-tools-common            # version 5.6.0-1017.17
sudo apt install linux-tools-common                    # version 5.4.0-195.215
sudo apt install linux-iot-tools-common                # version 5.4.0-1043.44
sudo apt install linux-nvidia-tegra-5.15-tools-common  # version 5.15.0-1027.27~20.04.1

ouc@islouc-vm:~/Downloads$ sudo apt install linux-intel-iotg-5.15-tools-common
E: dpkg was interrupted, you must manually run 'sudo dpkg --configure -a' to correct the problem.
```

解决方案：发现是之前装 texlive 时进程未完全完成安装便终止，故在运行并更新 perf 时遇到 dpkg（Debian 包管理器）配置未完成的问题。根据系统提示，手动运行 sudo dpkg –configure -a 来修正这个问题。（这一步耗费时间较长，后续直接换了虚拟机安装 perf）



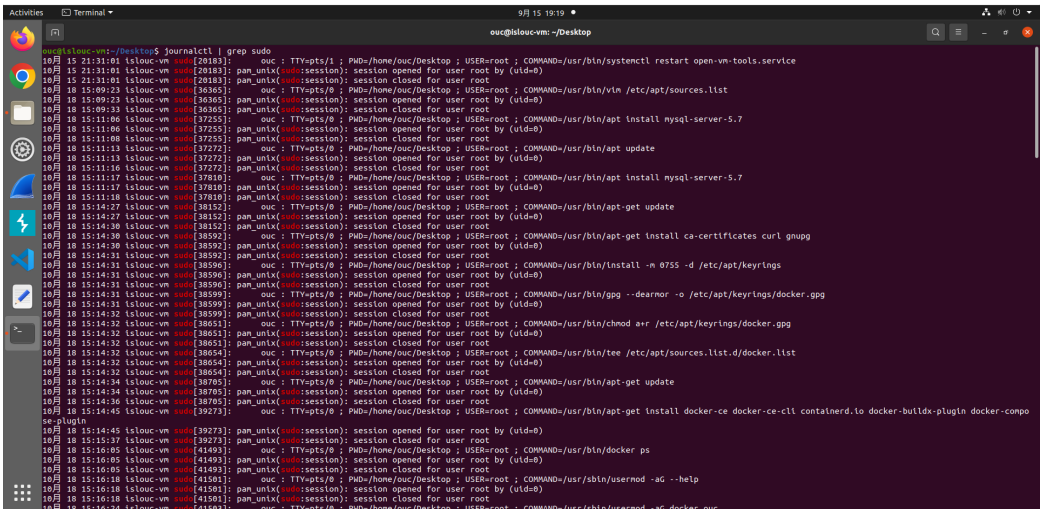
4 解题感悟

本实验中，通过练习调试及性能分析、元编程演示、大杂烩以及 PyTorch 的题目，我不仅学习到了 Linux 命令、各种性能分析工具以及调试工具的使用方法，还加深了对 Python 编程、系统资源管理及性能可视化分析的理解。本次实验的内容多而杂，对我们的综合素养和学习能力进行了锻炼，提高了我解决问题的能力。

在实例中引用到了我之前学过的知识，温故而知新，让我又对这些工具的使用方法有了新的理解。在网络攻防先导实践课上使用过的 pwndbg、wireshark 和 markdown，在计算机系统基础课上做 bomb 拆弹实验使用的 gdb，在图像处理与分析课上做卷积相关程序使用的 pytorch，在本实验中再次有所温习。除此之外也学到了不少新知识，比如 git，cprofile 等性能分析工具，perf 等命令等。通过本次练习，我的综合素养得到了进一步的提升。

5 github 链接

https://github.com/zxm2580/xtkfgjjc-202408.git


Activities
Terminal
9月 15 19:19
ouc@lslouc-vm: ~/Desktop
ouc@lslouc-vm:~/Desktop$ journalctl | grep sudo
10月 15 21:31:01 lslouc-vm sudo[20183]: ouc : TTY=pts/1 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/systemctl restart open-vm-tools.service
10月 15 21:31:01 lslouc-vm sudo[20183]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 15 21:31:01 lslouc-vm sudo[20183]: pam_unix(sudo:session): session closed for user root
10月 18 15:09:23 lslouc-vm sudo[36365]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/vim /etc/apt/sources.list
10月 18 15:09:23 lslouc-vm sudo[36365]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:09:33 lslouc-vm sudo[36365]: pam_unix(sudo:session): session closed for user root
10月 18 15:11:06 lslouc-vm sudo[37255]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt install mysql-server-5.7
10月 18 15:11:06 lslouc-vm sudo[37255]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:11:08 lslouc-vm sudo[37255]: pam_unix(sudo:session): session closed for user root
10月 18 15:11:13 lslouc-vm sudo[37272]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt update
10月 18 15:11:13 lslouc-vm sudo[37272]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:11:16 lslouc-vm sudo[37272]: pam_unix(sudo:session): session closed for user root
10月 18 15:11:17 lslouc-vm sudo[37810]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt install mysql-server-5.7
10月 18 15:11:17 lslouc-vm sudo[37810]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:11:18 lslouc-vm sudo[37810]: pam_unix(sudo:session): session closed for user root
10月 18 15:14:27 lslouc-vm sudo[38152]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get update
10月 18 15:14:27 lslouc-vm sudo[38152]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:14:30 lslouc-vm sudo[38152]: pam_unix(sudo:session): session closed for user root
10月 18 15:14:30 lslouc-vm sudo[38592]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get install ca-certificates curl gnupg
10月 18 15:14:30 lslouc-vm sudo[38592]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:14:31 lslouc-vm sudo[38592]: pam_unix(sudo:session): session closed for user root
10月 18 15:14:31 lslouc-vm sudo[38596]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/install -m 0755 -d /etc/apt/keyrings
10月 18 15:14:31 lslouc-vm sudo[38596]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:14:31 lslouc-vm sudo[38596]: pam_unix(sudo:session): session closed for user root
10月 18 15:14:31 lslouc-vm sudo[38599]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/gpg --dearmor -o /etc/apt/keyrings/docker.gpg
10月 18 15:14:31 lslouc-vm sudo[38599]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:14:32 lslouc-vm sudo[38599]: pam_unix(sudo:session): session closed for user root
10月 18 15:14:32 lslouc-vm sudo[38651]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/chmod a+r /etc/apt/keyrings/docker.gpg
10月 18 15:14:32 lslouc-vm sudo[38651]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:14:32 lslouc-vm sudo[38651]: pam_unix(sudo:session): session closed for user root
10月 18 15:14:32 lslouc-vm sudo[38654]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/tee /etc/apt/sources.list.d/docker.list
10月 18 15:14:32 lslouc-vm sudo[38654]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:14:32 lslouc-vm sudo[38654]: pam_unix(sudo:session): session closed for user root
10月 18 15:14:34 lslouc-vm sudo[38705]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get update
10月 18 15:14:34 lslouc-vm sudo[38705]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:14:36 lslouc-vm sudo[38705]: pam_unix(sudo:session): session closed for user root
10月 18 15:14:45 lslouc-vm sudo[39273]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
10月 18 15:14:45 lslouc-vm sudo[39273]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:15:37 lslouc-vm sudo[39273]: pam_unix(sudo:session): session closed for user root
10月 18 15:16:05 lslouc-vm sudo[41493]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/docker ps
10月 18 15:16:05 lslouc-vm sudo[41493]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:16:05 lslouc-vm sudo[41493]: pam_unix(sudo:session): session closed for user root
10月 18 15:16:18 lslouc-vm sudo[41501]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/sbin/usermod -aG --help
10月 18 15:16:18 lslouc-vm sudo[41501]: pam_unix(sudo:session): session opened for user root by (uid=0)
10月 18 15:16:18 lslouc-vm sudo[41501]: pam_unix(sudo:session): session closed for user root