

# 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
2 练习内容	1
2.1 四个题目	1
2.1.1 题目一	1
2.1.2 题目二	2
2.1.3 题目三	4
2.1.4 题目四	5
2.2 20 个实例	7
3 问题及解决方案	19
4 解题感悟	20
5 github 链接	20

# 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

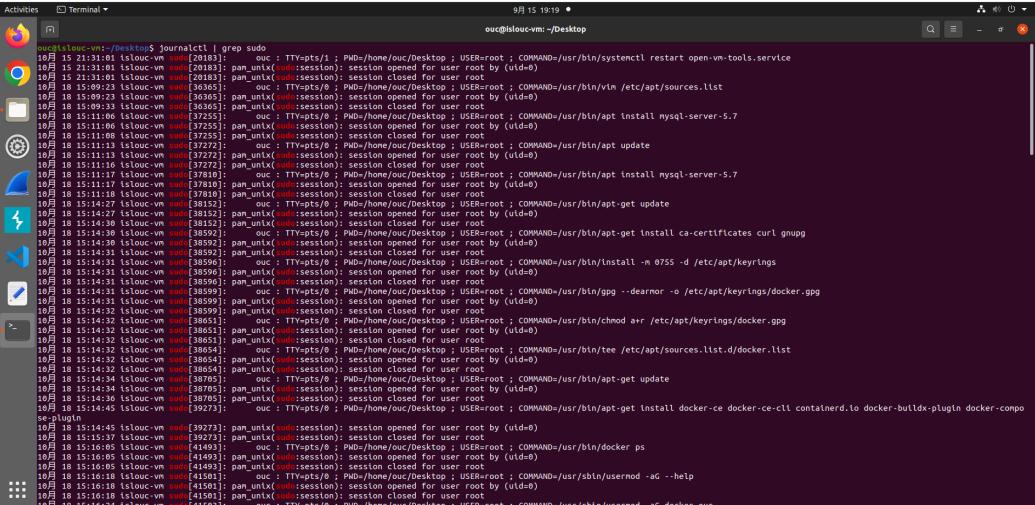
# 2 练习内容

## 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



```
journalctl | grep sudo
16月 18 21:31:01 tsilouc-vn sudo[2018]:    out : TTY&pts/1 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/bin/systemctl restart open-vn-tools.service
16月 18 21:31:01 tsilouc-vn sudo[2018]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:09:23 tsilouc-vn sudo[3636]:    out : TTY&pts/0 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/bin/vim /etc/apt/sources.list
16月 18 15:09:23 tsilouc-vn sudo[3636]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:11:06 tsilouc-vn sudo[3725]:    out : TTY&pts/0 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/bin/apt install mysql-server-5.7
16月 18 15:11:06 tsilouc-vn sudo[3725]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:11:06 tsilouc-vn sudo[3725]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:11:13 tsilouc-vn sudo[3727]:    out : TTY&pts/0 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/bin/apt update
16月 18 15:11:13 tsilouc-vn sudo[3727]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:11:17 tsilouc-vn sudo[3781]:    out : TTY&pts/0 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/bin/apt install mysql-server-5.7
16月 18 15:11:17 tsilouc-vn sudo[3781]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:11:18 tsilouc-vn sudo[3781]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:14:27 tsilouc-vn sudo[3815]:    out : TTY&pts/0 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get update
16月 18 15:14:27 tsilouc-vn sudo[3815]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:14:30 tsilouc-vn sudo[3815]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:14:30 tsilouc-vn sudo[3859]:    out : TTY&pts/0 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get install ca-certificates curl gnupg
16月 18 15:14:31 tsilouc-vn sudo[3859]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:14:31 tsilouc-vn sudo[3859]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:14:31 tsilouc-vn sudo[3859]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:14:31 tsilouc-vn sudo[3859]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:14:31 tsilouc-vn sudo[3859]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:14:31 tsilouc-vn sudo[3859]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:14:32 tsilouc-vn sudo[3865]:    out : TTY&pts/0 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/bin/chmod a+r /etc/apt/keyrings/docker.gpg
16月 18 15:14:32 tsilouc-vn sudo[3865]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:14:32 tsilouc-vn sudo[3865]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:14:32 tsilouc-vn sudo[3865]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:14:32 tsilouc-vn sudo[3865]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:14:32 tsilouc-vn sudo[3865]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:14:34 tsilouc-vn sudo[3876]:    out : TTY&pts/0 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get update
16月 18 15:14:34 tsilouc-vn sudo[3876]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:14:36 tsilouc-vn sudo[3876]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:14:45 tsilouc-vn sudo[3927]:    out : TTY&pts/0 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get install docker-ce docker-ce-cll containerd.io docker-buildx-plugin docker-compose
16月 18 15:14:45 tsilouc-vn sudo[3927]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:15:37 tsilouc-vn sudo[3927]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:16:05 tsilouc-vn sudo[4149]:    out : TTY&pts/0 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/bin/docker ps
16月 18 15:16:05 tsilouc-vn sudo[4149]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:16:05 tsilouc-vn sudo[4149]: pam_unix(sudo:session): session closed for user root by (uid=0)
16月 18 15:16:18 tsilouc-vn sudo[4150]:    out : TTY&pts/0 ; PM0:/home/loc/Desktop ; USER=root ; COMMAND=/usr/sbin/usermod -aG --help
16月 18 15:16:18 tsilouc-vn sudo[4150]: pam_unix(sudo:session): session opened for user root by (uid=0)
16月 18 15:16:18 tsilouc-vn sudo[4150]: pam_unix(sudo:session): session closed for user root by (uid=0)
```

## 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

The screenshot shows a code editor with the file 'sorts.py' open. The code contains implementations of insertion sort, quicksort, and quicksort\_inplace. Below the code editor is a terminal window displaying the output of the command 'python -m cProfile -s time sorts.py | grep sorts.py'. The terminal output shows the execution times for various parts of the sorting functions.

```
1 Import random
2
3 def test_sorted(fn, iters=1000):
4     for fn in range(iters):
5         l = [random.randint(0, 100) for i in range(0, random.randint(0, 50))]
6         assert fn(l) == sorted(l)
7         # print(f'{fn.__name__}, {fn(l)}')
8
9
10 def insertionsort(array):
11     for i in range(len(array)):
12         j = i
13         v = array[i]
14         while j >= 0 and v < array[j]:
15             array[j+1] = array[j]
16             j -= 1
17         array[j+1] = v
18     return array
19
20
21
22 def quicksort(array):
23     if len(array) <= 1:
24         return array
25     pivot = array[0]
26     left = [i for i in array[1:] if i < pivot]
27     right = [i for i in array[1:] if i >= pivot]
28     return quicksort(left) + [pivot] + quicksort(right)
29
30
31 def quicksort_inplace(array, low=0, high=None):
32     if len(array) <= 1:
33         return array
34     if high is None:
35         high = len(array)-1
36     if low >= high:
37         return array
38
39     pivot = array[high]
40     j = low
41     for i in range(low, high):
42         if array[i] <= pivot:
43             j += 1
44             array[i], array[j] = array[j], array[i]
45     array[high], array[j] = array[j], array[high]
46     quicksort_inplace(array, low, j)
47     quicksort_inplace(array, j+1, high)
48
49
50
51 ouc@lsouc-vm:~/Downloads$ python -m cProfile -s time sorts.py | grep sorts.py
34192/1000    0.018    0.000    0.029    0.000 sorts.py:22(quicksort)
32748/1000    0.018    0.000    0.019    0.000 sorts.py:31(quicksort_inplace)
1000    0.016    0.000    0.016    0.000 sorts.py:10(insertionsort)
3000    0.010    0.000    0.069    0.000 sorts.py:5(<listcomp>)
16596    0.005    0.000    0.005    0.000 sorts.py:26(<listcomp>)
16596    0.005    0.000    0.005    0.000 sorts.py:27(<listcomp>)
3    0.003    0.001    0.140    0.047 sorts.py:3(test_sorted)
1    0.000    0.000    0.141    0.141 sorts.py:1(<module>)
```

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：

```

ouc@tslouc-vn:~/Downloads$ kernprof -l -v sorts.py
Wrote profile results to sorts.py.lprof
Timer unit: 1e-06 s

Total time: 0.0910696 s
File: sorts.py
Function: insertionsort at line 9

Line #      Hits       Time  Per Hit   % Time  Line Contents
=====      ==       ===     ===       ===     =====
          9           @profile
          10          def insertionsort(array):
          11
          12    26194      2506.0      0.1      2.8      for i in range(len(array)):
          13    25194      2900.4      0.1      3.2      j = i-1
          14    25194      3052.9      0.1      3.4      v = array[i]
          15   227357      31042.5      0.1      34.1      while j >= 0 and v < array[j]:
          16   202163      26537.1      0.1      29.1      array[j+1] = array[j]
          17   202163      21843.0      0.1      24.0      j -= 1
          18    25194      3082.3      0.1      3.4      array[j+1] = v
          19    1000       105.2      0.1      0.1      return array
          20
          21 @profile
ouc@tslouc-vn:~/Downloads$ kernprof -l -v sorts.py
Wrote profile results to sorts.py.lprof
Timer unit: 1e-06 s

Total time: 0.042359 s
File: sorts.py
Function: quicksort at line 21

Line #      Hits       Time  Per Hit   % Time  Line Contents
=====      ==       ===     ===       ===     =====
          21           @profile
          22          def quicksort(array):
          23    33950      4689.5      0.1      11.1      if len(array) <= 1:
          24    17475      1401.9      0.1      3.3      return array
          25   16475      1875.6      0.1      4.4      pivot = array[0]
          26   16475      13572.3      0.8      32.0      left = [i for i in array[1:] if i < pivot]
          27   16475      12648.9      0.8      29.9      right = [i for i in array[1:] if i >= pivot]
          28   16475      8170.8      0.5      19.3      return quicksort(left) + [pivot] + quicksort(right)
          29
          30 @profile
          31 def quicksort_inplace(array,
          32   if len(array) <= 1:
          33
          34   while j >= 0 and v < arr
          35     arr[j+1] = arr[j]
          36     j -= 1
          37   arr[j+1] = v
          38
          39 @profile
          40 def quicksort(array):
          41   if len(array) <= 1:
          42     return array
          43   pivot = array[0]
          44   left = [i for i in array[1:] if i < pivot]
          45   right = [i for i in array[1:] if i >= pivot]
          46   return quicksort(left) + [pivot] + quicksort(right)
          47
          48 @profile
          49 def quicksort_inplace(array, low=0, high=None):
          50   if len(array) <= 1:
          51     return array
          52   pivot = array[high]
          53   j = low-1
          54   for i in range(low, high):
          55     if array[i] <= pivot:
          56       j += 1
          57       array[j], array[i] = array[i], array[j]
          58   array[j+1] = v
          59
          60 @profile
          61 def quicksort_inplace(array, low=0, h

```

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：

```

ouc@tslouc-vn:~/Downloads$ python -m memory_profiler sorts.py
Filename: sorts.py
Line #      Mem usage     Increment Occurrences   Line Contents
=====      ======     ======     =====
          9    18.375 MiB   18.375 MiB      1000  @profile
          10          def insertionsort(array):
          11
          12    18.375 MiB   0.000 MiB    26380      for i in range(len(array)):
          13    18.375 MiB   0.000 MiB    25380      j = i-1
          14    18.375 MiB   0.000 MiB    25380      v = array[i]
          15    18.375 MiB   0.000 MiB   232319      while j >= 0 and v < array[j]:
          16    18.375 MiB   0.000 MiB   206939      array[j+1] = array[j]
          17    18.375 MiB   0.000 MiB   206939      j -= 1
          18    18.375 MiB   0.000 MiB    25380      array[j+1] = v
          19    18.375 MiB   0.000 MiB      1000      return array
          20
          21 @profile
          22 def quicksort(array):
          23   if len(array) <= 1:
          24     return array
          25   pivot = array[0]
          26   left = [i for i in array[1:] if i < pivot]
          27   right = [i for i in array[1:] if i >= pivot]
          28   return quicksort(left) + [pivot] + quicksort(right)
          29
          30 @profile
          31 def quicksort_inplace(array, low=0, high=None):
          32   if len(array) <= 1:
          33     return array
          34   if high is None:
          35     high = len(array)-1
          36   if low >= high:
          37     return array
          38   pivot = array[high]
          39   j = low-1
          40   for i in range(low, high):
          41     if array[i] <= pivot:
          42       j += 1
          43       array[j], array[i] = array[i], array[j]
          44
          45 @profile
          46 def quicksort_inplace(array, low=0, h

```

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。

The image shows a vertical stack of five terminal windows, each with a dark background and light-colored text. The windows are arranged from top to bottom:

- Terminal 1:** Shows the command `python -m http.server 4444` being run, followed by the output "Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...".
- Terminal 2:** Shows the same command and output as Terminal 1.
- Terminal 3:** Shows the command `lsof | grep LISTEN` being run. The output includes several Docker-related entries and a line for the Python process: "python 4030 0t0 TCP \*:4444 (LISTEN)".
- Terminal 4:** Shows the command `python -m http.server 4444` running again, followed by "Terminated".
- Terminal 5:** Shows the command `lsof | grep LISTEN` running again, followed by the command `kill 4030` and the prompt "ouc@islouc-vm:~/Desktop\$".

#### 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：

```
1 paper.pdf: paper.tex plot-data.png
2     pdflatex paper.tex
3
4 plot-%.png: %.dat plot.py
5     ./plot.py -i $*.dat -o $@
6
7 .PHONY: clean
8 clean:
9     mkdir -p Untrack
10    rm -f *~.*~
11    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/cmr12.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      * spectacular 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=0xfffffd134) 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。

```

Program stopped.
0x000077c807c1ff in __GI_lIBC_read [fd=0, buf=0x7ffc44724060, nbytes=256] at ./sysdeps/unix/sysv/linux/read.c:26
26
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0xf7fc44723fb0 <[vfprintf_internal+0x10]> in _IO_2_1_stdout_ at vfprintf_internal.c:377
377 vfprintf_internal.c: No such file or directory.
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]
RWX 0x7fc44723fb0 (<_IO_2_1_stdout_>) ← 0x7fc44723fb0
RWX 0x7fc44723fb4 (<_IO_2_1_stdout_>) ← 0x7fc44723fb4
RWX 0x0
RDX 0x7fc44723fb8 ← 0x300000000008
RDI 0x7fc44723fb0 ← 0x7fc44723fb0
RSI 0x7fc44723fb4 ← '[+] Here is flag6: %s\n'
RBP 0x0
RCX 0x1c
RCX 0x1c
R10 0x4024f4 ← '[+] Here is flag6: %s\n'
R11 0x24
R12 0x7fc44723fb0 (<_IO_2_1_stdout_>) ← 0x7fc44723fb0
R13 0x4024f4 ← '[+] Here is flag6: %s\n'
R14 0x7fc44723fb8 ← 0x300000000008
R15 0xbfd2887
RSP 0x7fc44723fb8 ← 0x6262626262626262 ('bbbbbbbb')
RIP 0x7fc44723fb0 ← 0x0
RIP 0x7fc44723fb0 (buffered_vfprintf+104) ← movaps xmmword ptr [rsp + 0x40], xmm0
[ D154M / x86-64 / set emulate on ]
0x7fc44723fb0 <buffered_vfprintf+104> movaps xmmword ptr [rsp + 0x40], xmm0
0x7fc44723fb9 <buffered_vfprintf+189> call _vfprintf_Internal
0x7fc44723fb9 <_vfprintf_Internal>
0x7fc44723fb2 <buffered_vfprintf+194> nov    ebp, dword ptr [rip + 0x1775e8] <__libc_pthread_functions_init>
0x7fc44723fb8 <buffered_vfprintf+200> nov    r12d, eax
0x7fc44723fb4 <buffered_vfprintf+203> test   ebp, ebp
0x7fc44723fb0 <buffered_vfprintf+205> jne    buffered_vfprintf+512
0x7fc44723fb0 <buffered_vfprintf+512>
0x7fc44723fb2 <buffered_vfprintf+211> nov    rax, qword ptr [rip + 0x1720ae]
0x7fc44723fb8 <buffered_vfprintf+218> nov    edx, dword ptr [rbx]
[+] Here is flag6: flag{inv0k3_func_5ucc3ss5fu1lY}

```

(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

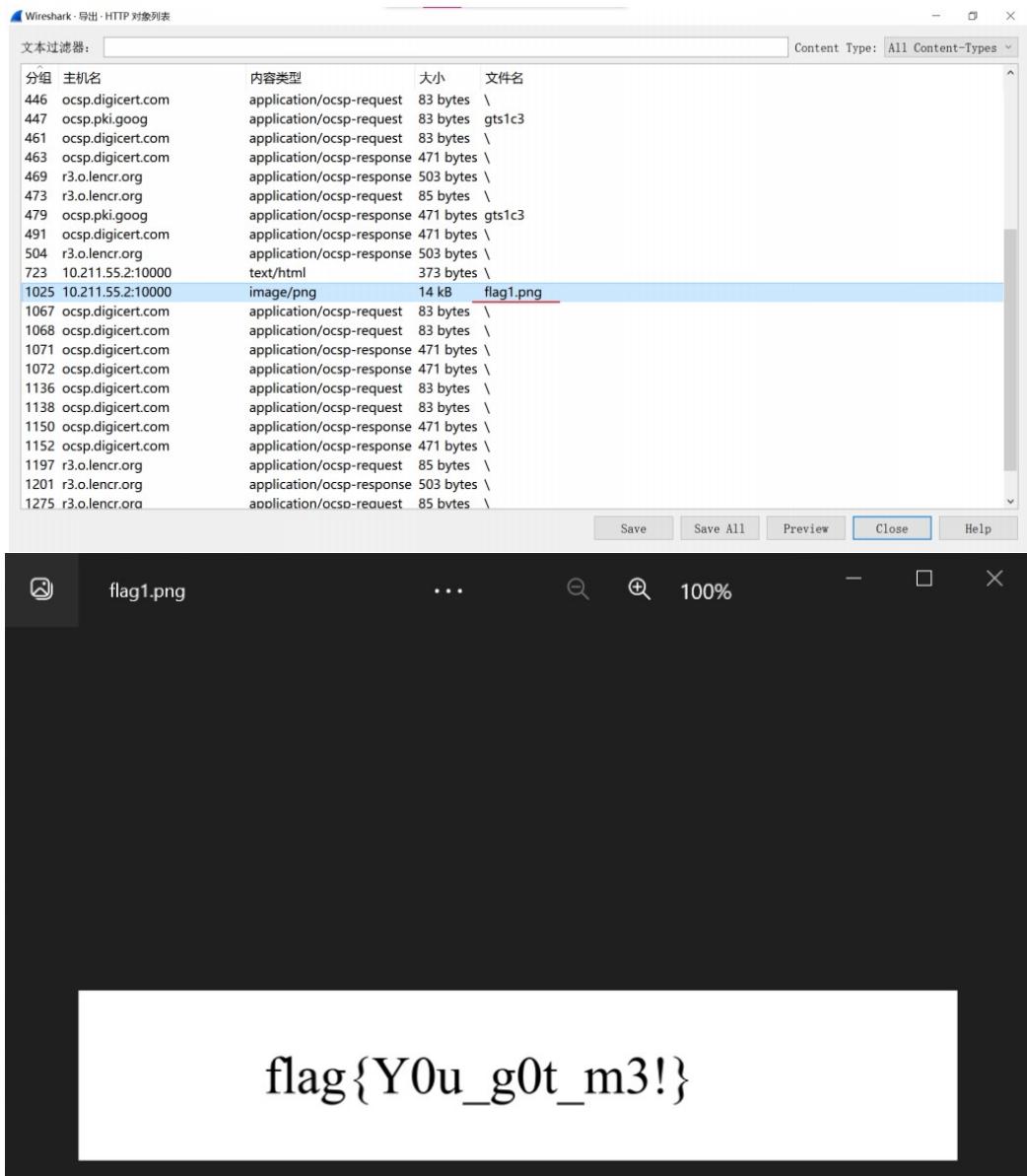
```

@file: Read options from <file>
-v, --version Display this program's version number
-i, --info List object formats and architectures supported
-H, --help Display this information
bc@abc-virtual-machine:~/Desktop/bomb$ objdump bomb > asm.txt
Display information from object <file(s)>
At least one of the following switches must be given:
-a, --all-headers Display all headers
-f, --file-headers Display the contents of the overall file header
-p, --private-headers Display object format specific file header
-P, --private=OPT,OPT... Display object format specific contents
-h, --section-headers Display the contents of the section headers
-x, --all-headers Display the contents of all headers
-d, --disassemble Display assembly contents of executable sections
-D, --disassemble-all Display assembly contents of all sections
--disassemble=<sym> Display member contents from <sym>
-s, --source Insert source code with disassembly
--source-comment[=<txt>] Prefix lines of source code with <txt>
-s, --full-contents Display the full contents of all sections
-g, --debugging Display debug information in object file
-e, --debugging-tags Display debug information using ctags style
-C, --stabs Display (in raw form) any STABS info in the
-W[!L]apline[!F]e[!R]U[!T]a[!C]K[!I]
--dwarf[!F]e[!R]U[!T]a[!C]K[!I],--dwarf-line[<info>],--abbrev[<pubnames>],--aranges[<macro
--frames[<interp>,<str>,<loc>,<Ranges>,<pubtypes>,
--sgdb_index[<info>],--trace_abrev[<trace_aranges>,
--addr[<cu_index>],--elinks[<follow_elinks>]
--ctf[<SECTION>] Display CTF info from SECTION
-t, --sym Display the contents of the symbol table(s)
-l, --dynamic-syms Display the contents of the dynamic symbol table
-r, --reloc Display the relocation entries in the file
-R, --dynamic-reloc Display the dynamic relocation entries in the file
@file> Read options from <file>
-v, --version Display this program's version number
-i, --info List object formats and architectures supported
-H, --help Display this information
bc@abc-virtual-machine:~/Desktop/bomb$ C
bc@abc-virtual-machine:~/Desktop/bomb$ objdump -d bomb > asm.txt
bc@abc-virtual-machine:~/Desktop/bomb$ gedit asm.txt

```

The terminal shows the usage of objdump, the command to generate the assembly dump, and the command to open it in a text editor. The assembly dump itself is too large to show here but is visible in the terminal window.

**(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，

```
ouc@islouc-vm:~/Desktop$ time curl https://missing.csail.mit.edu &> /dev/null
real    0m1.412s
user    0m0.011s
sys     0m0.006s
```

**(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。

fibonacci.py  
~/Downloads

```

1#!/usr/bin/env python
2def fib0(): return 0
3
4def fib1(): return 1
5
6s = """def fib(): return fib() + fib()"""
7
8if __name__ == '__main__':
9
10    for n in range(2, 10):
11        exec(s.format(n, n-1, n-2))
12    # from functools import lru_cache
13    # for n in range(10):
14    #     exec("fib = lru_cache(1)(fib)".format(n, n))
15    print(eval("fib9"))

```

successfully installed pycallgraph

Defaulting to user installation because normal site-packages is not writable

Collecting pycallgraph

Downloading pycallgraph-1.0.1.tar.gz (36 kB)

Preparing metadata (setup.py) ... done

building ...

Building wheel for pycallgraph (setup.py) ... done

Created wheel for pycallgraph: filename=pycallgraph-1.0.1-py3-none-any.whl size=35942 sha256=ec45d51cae7e2abd3ba4081c691d2ebcd4580af59505c73d3f73c2503f390e4a

Stored in directory: /home/ouc/.cache/pip/wheels/c1/c4/a0/22b61ff9ca8988bb8dd03ecd019b84697a39e7b187bc5798

successfully installed pycallgraph-1.0.1

Successfully installed pycallgraph-1.0.1

[notice] A new release of pip is available: 23.2.1 --> 24.2
[notice] To update, run: python3 -m pip install --upgrade pip
ouc@ouc-VirtualBox:~/Downloads\$ vtn fibonact.py

Defaulting to user installation because normal site-packages is not writable

Collecting graphviz

Obtaining dependency information for graphviz from https://files.pythonhosted.org/packages/00/be/d9db2d1d52697c6adc9eacf50e8965b6345cc143f671e1ed08818d5cf/graphviz-0.20.3-py3-none-any.whl.metadata

Downloading graphviz-0.20.3-py3-none-any.whl (47 kB)
 Downloading graphviz-0.20.3-py3-none-any.whl (47 kB)

Installing collected packages: graphviz

47.1/47.1 kB 728.7 kB/s eta 0:00:00

successfully installed graphviz-0.20.3

(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。

```

终端 终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H) 23:54
正在设置 python-pkg-resources (20.7.0-1) ...
a@123: ~

1 [|||||100.0%] Tasks: 108, 189 thr: 4 running
2 [|||||100.0%] Load average: 3.75 1.17 0.41
Mem[||||| 757M/3.83G] Uptime: 00:30:14
Swp[ 0K/975M]

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
4406 a 20 0 7480 92 0 R 69.8 0.0 0:38.45 stress -c 3
4407 a 20 0 7480 92 0 R 68.4 0.0 0:40.33 stress -c 3
4405 a 20 0 7480 92 0 R 59.8 0.0 0:38.15 stress -c 3
1838 a 20 0 1265M 117M 81716 S 0.7 3.0 0:07.81 compiz
1009 root 20 0 439M 70608 40160 S 0.7 1.8 0:03.82 /usr/lib/xorg/Xo
2116 a 20 0 621M 54732 41700 S 0.0 1.4 0:01.88 /usr/lib/gnome-t [16.7 kB]
4862 a 20 0 27756 3908 3232 R 0.0 0.1 0:00.06 htop
1112 root 20 0 439M 70608 40160 S 0.0 1.8 0:00.34 /usr/lib/xorg/Xo
1683 a 20 0 573M 30964 31416 S 0.0 1.0 0:00.07 /usr/lib/x86_64-
1394 a 20 0 406M 41272 24180 S 0.0 1.0 0:00.28 /usr/bin/fcitx
1673 a 20 0 573M 30964 31416 S 0.0 1.0 0:00.31 /usr/lib/x86_64-
1740 a 20 0 393M 13116 11644 S 0.0 0.3 0:00.01 /usr/lib/x86_64-
1320 a 20 0 43720 4164 2852 S 0.0 0.1 0:00.31 dbus-daemon --fo
1378 a 20 0 640M 52248 42048 S 0.0 1.3 0:00.26 /usr/lib/x86_64-
1652 a 20 0 583M 42156 34948 S 0.0 1.0 0:00.18 /usr/lib/x86_64-
1882 a 20 0 260M 2384 25360 S 0.0 0.7 0:01.68 /lib/vmware-tool

F1 Help F2 Setup F3 Search F4 Filter F5 Free F6 Sort By F7 Nice F8 Nice F9 Kill F10 Exit

正在读取状态信息... 完成
stress 已经是最新版 (1.0.4-1)。
升级了 0 个软件包, 新安装了 0 个软件包, 要卸载 0 个软件包, 有 44 个软件包未被升级。
a@123:~$ stress -c 3
stress: info: [4404] dispatching hogs: 3 cpu, 0 io, 0 vm, 0 hdd

```

(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。

The screenshot shows a terminal window with two tabs. The active tab displays system monitoring information:

```

sud a@1 a@123:~ 
正在 a@123:~ $ taskset --cpu-list 0,2 stress -c 3
正在 stress: info: [4891] dispatching hogs: 3 cpu, 0 io, 0 vm, 0 hdd
正在
升
需
解
获
Swp[          0K/975M]
正

```

Below this, a top command output shows system load and process details:

```

1 [||||| 100.0%] Tasks: 113, 187 thr; 7 running
2 [||||| 100.0%] Load average: 4.40 1.95 0.75
Mem[ 781M/3.83G] Uptime: 00:31:44
7 kB]

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
4891 a 20 0 7480 92 0 R 33.1 0.0 0:09.57 stress -c 3
4893 a 20 0 7480 92 0 R 33.1 0.0 0:09.57 stress -c 3
4894 a 20 0 7480 92 0 R 33.1 0.0 0:09.57 stress -c 3
4407 a 20 0 7480 92 0 R 33.1 0.0 1:30.14 stress -c 3
4405 a 20 0 7480 92 0 R 33.1 0.0 1:27.91 stress -c 3
4406 a 20 0 7480 92 0 R 33.1 0.0 1:27.36 stress -c 3
1830 a 20 0 1267M 119M 81720 S 0.0 3.0 0:08.50 compiz
1009 root 20 0 440M 76608 40160 S 0.0 1.8 0:04.18 /usr/lib/xorg/Xo
1673 a 20 0 573M 48284 31504 S 0.0 1.0 0:00.34 /usr/lib/x86_64-
1882 a 20 0 262M 31432 25584 S 0.7 0.8 0:01.82 /lib/vmware-tool
1511 a 20 0 202M 5356 4868 S 0.0 0.1 0:00.04 /usr/lib/at-spi2
4862 a 20 0 27756 3908 3232 R 0.0 0.1 0:00.35 htop
1112 root 20 0 440M 76608 40160 S 0.0 1.8 0:00.41 /usr/lib/xorg/Xo
2116 a 20 0 621M 54888 41764 S 0.0 1.4 0:02.06 /usr/lib/gnome-t
1206 root 20 0 181M 13116 10680 S 0.0 0.3 0:01.34 /sbin/vmtoolsd
2119 a 20 0 621M 54888 41764 S 0.0 1.4 0:00.05 /usr/lib/gnome-t

```

(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下拒绝您的提交。

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

```

ouc@islouc-vm:~/Desktop/gjjc/.git/hooks$ cat pre-commit.sample
10 if git rev-parse --verify HEAD >/dev/null 2>&1
11 then
12     against=$HEAD
13 else
14     # Initial commit: diff against an empty tree object
15     against=$(git hash-object -t tree /dev/null)
16 fi
17
18 # If you want to allow non-ASCII filenames set this variable to true.
19 allownonascii=$(git config --bool hooks.allownonascii)
20
21 # Redirect output to stderr.
22 exec 1>&2
23
24 # Cross platform projects tend to avoid non-ASCII filenames; prevent
25 # them from being added to the repository. We exploit the fact that the
26 # printable range starts at the space character and ends with tilde.
27 # If you enable this, make sure to add it to .gitignore.
28 # Note that the use of brackets around a tr range is ok here. (It's
29 # even required, for portability to Solaris iO's /usr/bin/tr), since
30 # the square bracket bytes happen to fall in the designated range.
31 test $(git diff --cached --name-only | diff-filter=A -z $against |
32 LC_ALL=C tr '[:space:]' '\n' | wc -c) != 0
33 then
34     cat <<EOF
35 Error: Attempt to add a non-ASCII file name.
36
37 This can cause problems if you want to work with people on other platforms.
38
39 To be portable it is advisable to rename the file.
40
41 If you know what you are doing you can disable this check using:
42
43 git config hooks.allownonascii true
44 EOF
45 exit 1
46 fi
47
48 if ! make ; then
49     echo "build failed, commit rejected"
50     exit 1
51 fi
52
53 # If there are whitespace errors, print the offending file names and fail.
54 exec git diff-index --check --cached $against --
pre-commit.sample [+]

```

ouc@islouc-vm:~/Desktop/gjjc/.git/hooks\$ mv pre-commit.sample 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。

```

ouc@islouc-vm:~/Desktop/gjjc/.git/hooks$ echo -e "\e[38;2;255;0;0mThis is red\e[0m"
This is red

```

(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。

```

1 #!/usr/bin/env bash
2 for R in $(seq 0 20 255); do
3     for G in $(seq 0 20 255); do
4         for B in $(seq 0 20 255); do
5             printf "\e[38;2;${R};${G};${B}m\033[0m";
6         done
7     done
8 done
~ ~ ~
ouc@islouc-vm:~/Desktop$ vi color.sh
ouc@islouc-vm:~/Desktop$ chmod +x color.sh
ouc@islouc-vm:~/Desktop$ ./color.sh

```

(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)

## H: 数据结构与算法实验一

实验题目：多项式乘法问题

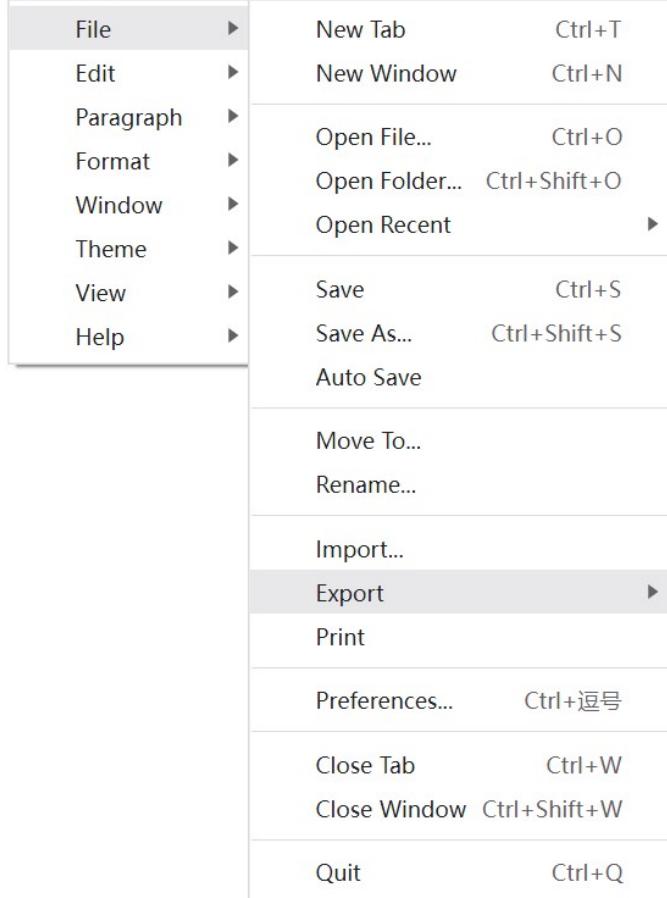
实验目的：

设计一个一元稀疏多项式简单计算器。

实验内容与要求：

三 W 1871

F: > 课程归档 >



## 与算法实验一

### 多项式乘法问题

流多项式简单计算器。

要求：

简单计算器的基本功能是：

“输出形式为整数序列：

约系数和指数，序列按指

多项式b相乘，建立多项式

实验步骤：

-  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。

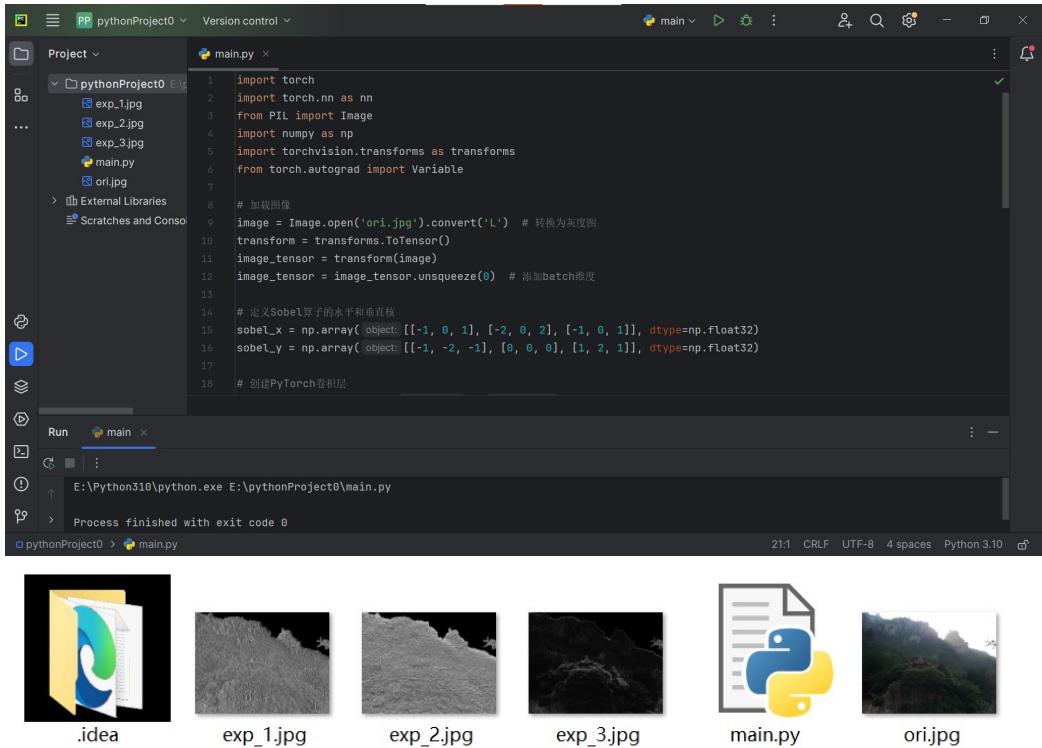
(2) 定义 Sobel 算子的水平和垂直核，创建 PyTorch 卷积层，设置卷积核权重。

(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）

```
ouc@islouc-vm:~/Downloads$ sudo dpkg --configure -a
Setting up purifyeps (1.1-2) ...
Setting up context (2019.03.21.20190425-2) ...
Running mtxrun --generate. This may take some time... done.
Pregenerating ConTeXt MarkIV format. This may take some time... ■
```

## 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>

