

系统工具开发基础实验

曹瑜 22020007007

September 14, 2024

1 实验内容

调试及性能分析

元编程

大杂烩

pytorch 入门

2 实验目的

- (1) 通过 shellcheck 等工具检查脚本问题，学习性能分析和监控工具，找到程序中最耗时、最耗资源的部分。
- (2) 学习构建系统、代码测试以及依赖管理。
- (3) 复习本学期学习的其他内容，练习拓展内容。

3 实验步骤

3.1 调试及性能分析

3.1.1 使用 log show 命令获取系统日志中和 sudo 有关的条目

使用 Linux 上的 journalctl。如果找不到相关信息，您可以执行一些无害的命令，例如 sudo ls 然后再次查看。

```

ouc@islouc-vm:~/Desktop/systemtool/solution$ journalctl | grep sudo
12月 09 17:43:22 islouc-vm sudo[4106]:      ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get install
samba
12月 09 17:43:22 islouc-vm sudo[4106]: pam_unix(sudo:session): session opened for user root by (uid=0)
12月 09 17:44:39 islouc-vm sudo[4106]: pam_unix(sudo:session): session closed for user root
12月 09 18:05:06 islouc-vm sudo[3947]:      ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get install
samba
12月 09 18:05:06 islouc-vm sudo[3947]: pam_unix(sudo:session): session opened for user root by (uid=0)
12月 09 18:05:07 islouc-vm sudo[3947]: pam_unix(sudo:session): session closed for user root
12月 09 18:20:14 islouc-vm sudo[4242]:      ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/gedit /etc/samba
/smb.conf
12月 09 18:20:14 islouc-vm sudo[4242]: pam_unix(sudo:session): session opened for user root by (uid=0)
12月 09 18:22:25 islouc-vm sudo[4242]: pam_unix(sudo:session): session closed for user root
4月 25 20:12:15 islouc-vm sudo[31716]:     ouc : TTY=pts/2 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt install trac
eroute
4月 25 20:12:15 islouc-vm sudo[31716]: pam_unix(sudo:session): session opened for user root by (uid=0)
4月 25 20:12:27 islouc-vm sudo[31716]: pam_unix(sudo:session): session closed for user root
5月 11 22:14:08 islouc-vm sudo[16114]:     ouc : TTY=pts/1 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/snap install val
grind
5月 11 22:14:08 islouc-vm sudo[16114]: pam_unix(sudo:session): session opened for user root by (uid=0)
5月 11 22:14:12 islouc-vm sudo[16114]: pam_unix(sudo:session): session closed for user root
5月 11 22:14:34 islouc-vm sudo[16129]:     ouc : TTY=pts/1 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/snap install val
grind
5月 11 22:14:34 islouc-vm sudo[16129]: pam_unix(sudo:session): session opened for user root by (uid=0)
5月 11 22:14:36 islouc-vm sudo[16129]: pam_unix(sudo:session): session closed for user root
5月 11 22:14:51 islouc-vm sudo[16143]:     ouc : TTY=pts/1 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt install valg
rind
5月 11 22:14:51 islouc-vm sudo[16143]: pam_unix(sudo:session): session opened for user root by (uid=0)
5月 11 22:15:15 islouc-vm sudo[16143]: pam_unix(sudo:session): session closed for user root
6月 12 22:10:29 islouc-vm sudo[4821]:     ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get update
6月 12 22:10:29 islouc-vm sudo[4821]: pam_unix(sudo:session): session opened for user root by (uid=0)
6月 12 22:10:53 islouc-vm sudo[4821]: pam_unix(sudo:session): session closed for user root
6月 12 22:11:40 islouc-vm sudo[5387]:     ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt-get install p
ostgresql postgresql-client
6月 12 22:11:40 islouc-vm sudo[5387]: pam_unix(sudo:session): session opened for user root by (uid=0)
6月 12 22:12:56 islouc-vm sudo[5387]: pam_unix(sudo:session): session closed for user root
6月 12 22:13:33 islouc-vm sudo[8911]:     ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=postgres ; COMMAND=/bin/bash
6月 12 22:13:33 islouc-vm sudo[8911]: pam_unix(sudo:session): session opened for user postgres by (uid=0)
6月 12 22:21:41 islouc-vm sudo[8978]:     ouc : TTY=pts/1 ; PWD=/home/ouc/Desktop ; USER=postgres ; COMMAND=/bin/bash
6月 12 22:21:41 islouc-vm sudo[8978]: pam_unix(sudo:session): session opened for user postgres by (uid=0)
6月 12 23:27:34 islouc-vm sudo[8978]: pam_unix(sudo:session): session closed for user postgres
6月 12 23:27:38 islouc-vm sudo[8911]: pam_unix(sudo:session): session closed for user postgres
6月 12 23:27:58 islouc-vm sudo[9539]:     ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=postgres ; COMMAND=/bin/bash
6月 12 23:27:58 islouc-vm sudo[9539]: pam_unix(sudo:session): session opened for user postgres by (uid=0)
6月 12 23:28:26 islouc-vm sudo[9556]:     ouc : TTY=pts/1 ; PWD=/home/ouc/Desktop ; USER=postgres ; COMMAND=/bin/bash
6月 12 23:28:26 islouc-vm sudo[9556]: pam_unix(sudo:session): session opened for user postgres by (uid=0)

```

Figure 1: 3.1.1 获取系统日志中 sudo 有关的条目

3.1.2 安装 shellcheck 并尝试对下面的脚本进行检查

脚本文件如下:

```

#!/bin/sh
## Example: a typical script with several problems
for f in $(ls *.m3u)
do
    grep -qi hq.*mp3 $f \
        && echo -e 'Playlist $f contains a HQ file in mp3 format'
done

```

在 /.vimrc 文件中添加以下内容以安装 neomake 插件

```

call plug#begin()
Plug 'neomake/neomake'
call plug#end()

```

配置 neomake: 在 /.vimrc 文件中添加以下配置:

```

" Enable Neomake for shellcheck
let g:neomake_enabled_makers = {'sh': ['shellcheck']}

```

在 vim 执行: PlugInstall 安装插件

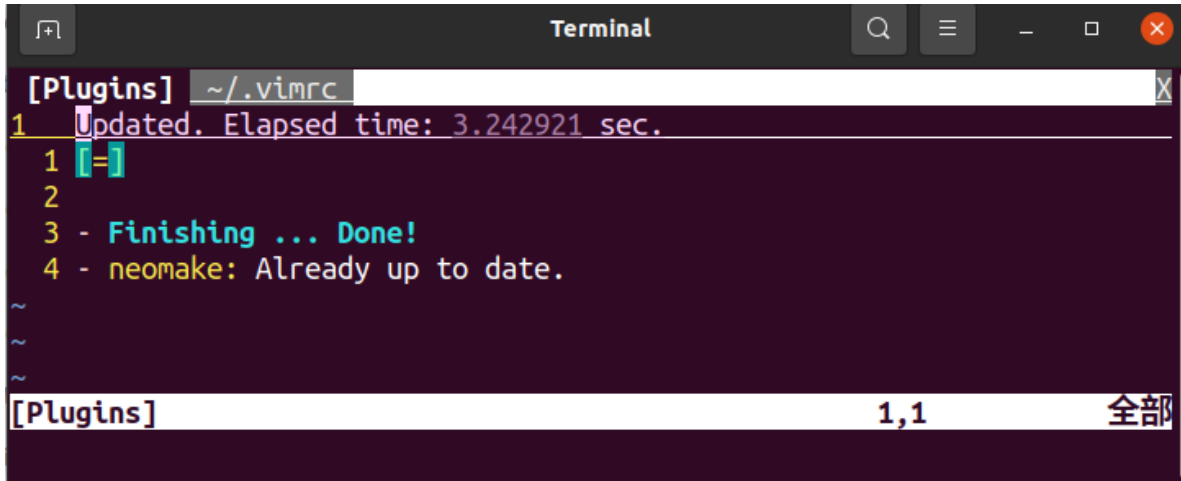


Figure 2: 3.1.2 PlugInstall 安装插件

然后在需要检查的 shell 脚本中执行: Neomake 执行 shellcheck 检查

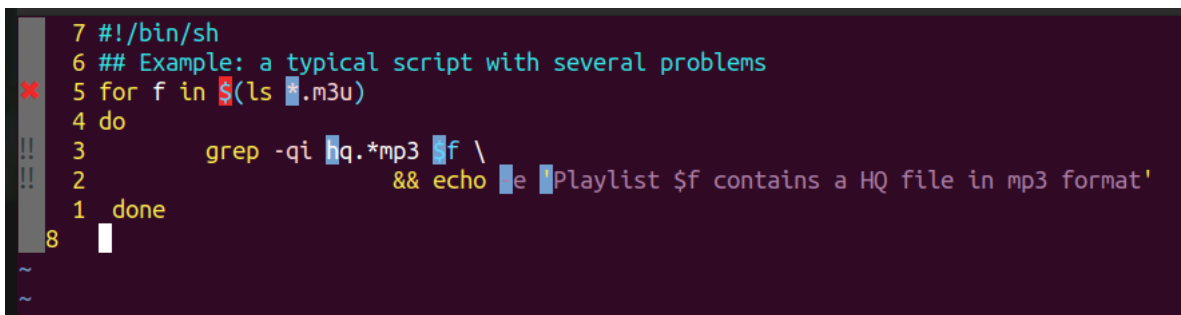


Figure 3: 3.1.2 警告和错误提醒

光标移动到对应行时可以看到警告或错误。

3.1.3 通过记忆法来对其进行优化并生成图片

将以下 Python 代码保存到文件 fib.py 中:

```
#!/usr/bin/env python

def fib0(): return 0
def fib1(): return 1

s = """def fib{}(): return fib{}() + fib{}()"""

if __name__ == '__main__':
    for n in range(2, 10):
        exec(s.format(n, n-1, n-2))
```

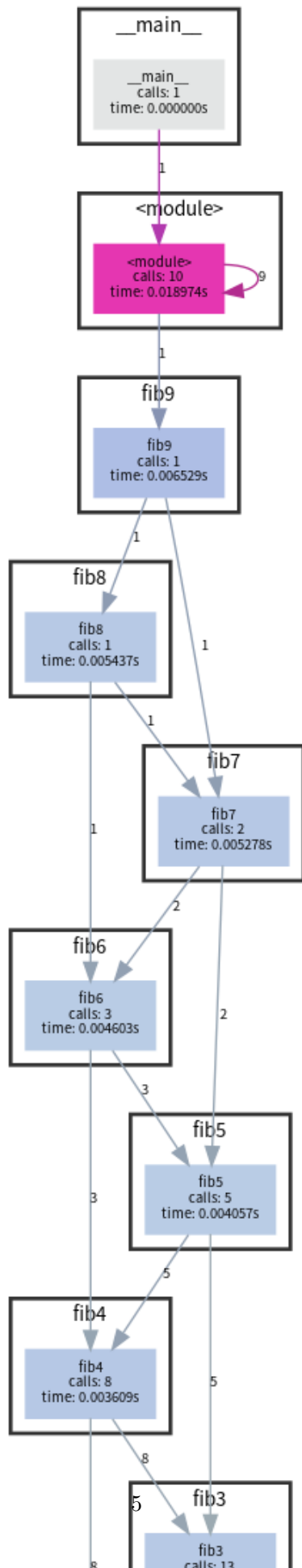
```
# from functools import lru_cache
# for n in range(10):
#     exec("fib{} = lru_cache(1)(fib{})".format(n, n))
print(eval("fib9()"))
```

安装 pycallgraph 和 graphviz:

```
ouc@islouc-vm:~/Desktop/systemtool/solution$ pip install "setuptools<58.0.0"
Requirement already satisfied: setuptools<58.0.0 in /usr/lib/python3/dist-packages (45.2.0)
ouc@islouc-vm:~/Desktop/systemtool/solution$ pip install pycallgraph
Collecting pycallgraph
  Downloading pycallgraph-1.0.1.tar.gz (36 kB)
Building wheels for collected packages: pycallgraph
  Building wheel for pycallgraph (setup.py) ... done
  Created wheel for pycallgraph: filename=pycallgraph-1.0.1-py3-none-any.whl size=35942 sha256=600e2a56c49e2959a6d64da24be85dcc1cadfab38d69801c9977b644414f767
  Stored in directory: /home/ouc/.cache/pip/wheels/c1/6c/a0/22b61ff9ca89881bb8d030ecd019b84697a39e7b187bc57938
Successfully built pycallgraph
Installing collected packages: pycallgraph
Successfully installed pycallgraph-1.0.1
ouc@islouc-vm:~/Desktop/systemtool/solution$
```

Figure 4: 3.1.3 安装 pycallgraph 和 graphviz

生成调用图（未使用缓存）



删除 fib.py 中的注释内容，再次执行 `pycallgraph graphviz - ./fib.py`，生成使用缓存的调用图

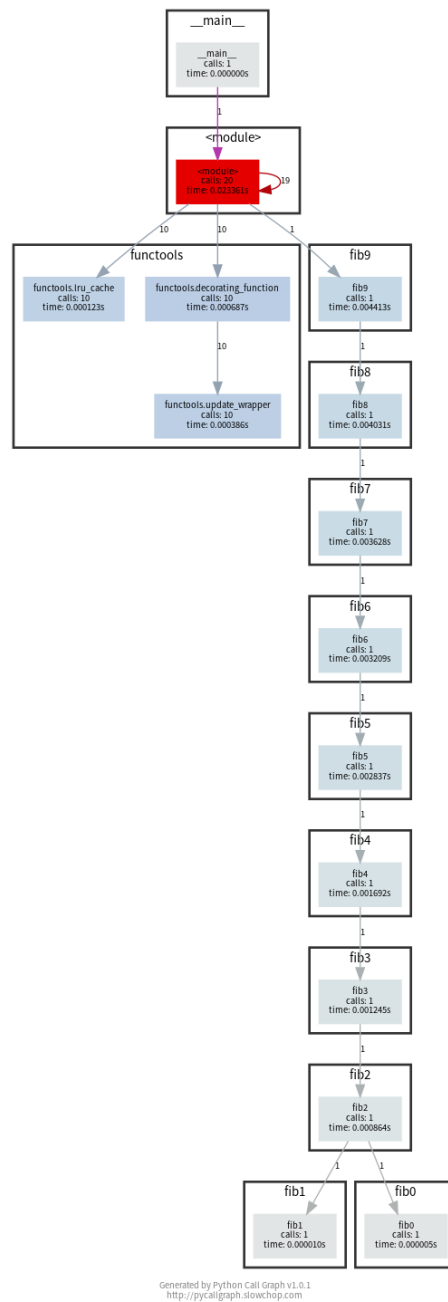


Figure 6: 3.1.3 使用缓存的调用图

3.1.4 解决端口被占用的问题并终止相关进程

启动一个简单的 web 服务器

```
ouc@islouc-vm:~/Desktop/systemtool/solution$ lsof -i -P -n | grep LISTEN
python3 6869 ouc    3u  IPv4 103665      0t0  TCP *:4444 (LISTEN)
ouc@islouc-vm:~/Desktop/systemtool/solution$
```

Figure 7: 3.1.4 启动 web 服务器

查找监听该端口的进程：打开另外一个终端，使用 `lsof` 命令来列出所有正在监听的端口以及对应的进程

通过找到的 PID (6869)，输入 `kill <PID>` 终止该进程

```
ouc@islouc-vm:~/Desktop/systemtool/solution$ python3 -m http.server 4444
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
```

Figure 8: 3.1.4 终止进程

3.1.5 限制进程资源

执行 `stress -c 3` 并使用 `htop` 对 CPU 消耗进行可视化。现在，执行 `taskset -cpu-list 0,2 stress -c 3` 并可视化。stress 占用了 3 个 CPU 吗？为什么没有？

首先是设备正常运行状态下的资源占用情况

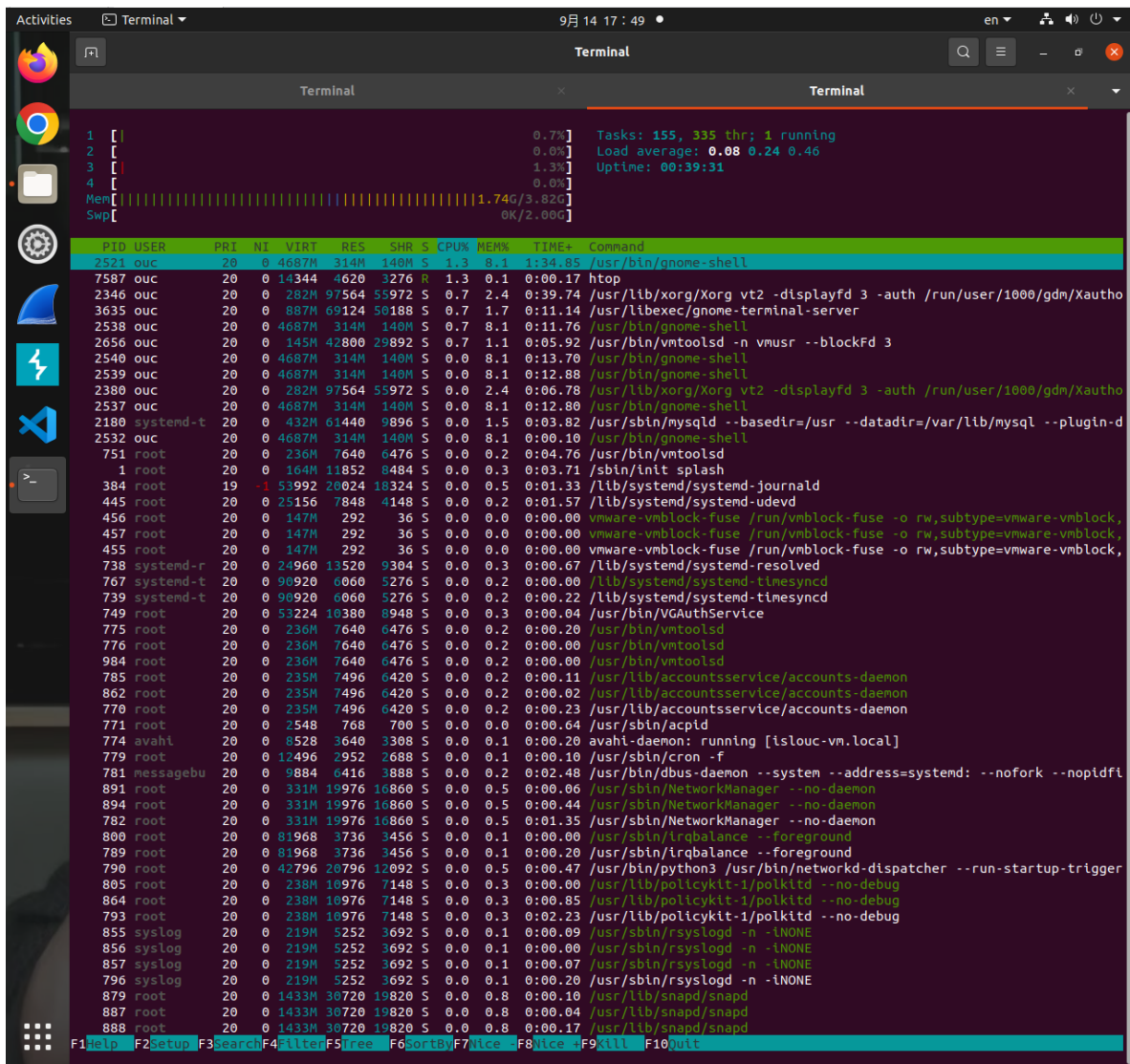


Figure 9: 3.1.5 设备正常运行状态下的资源占用情况

创建负载：stress -c 3，查看其资源占用情况

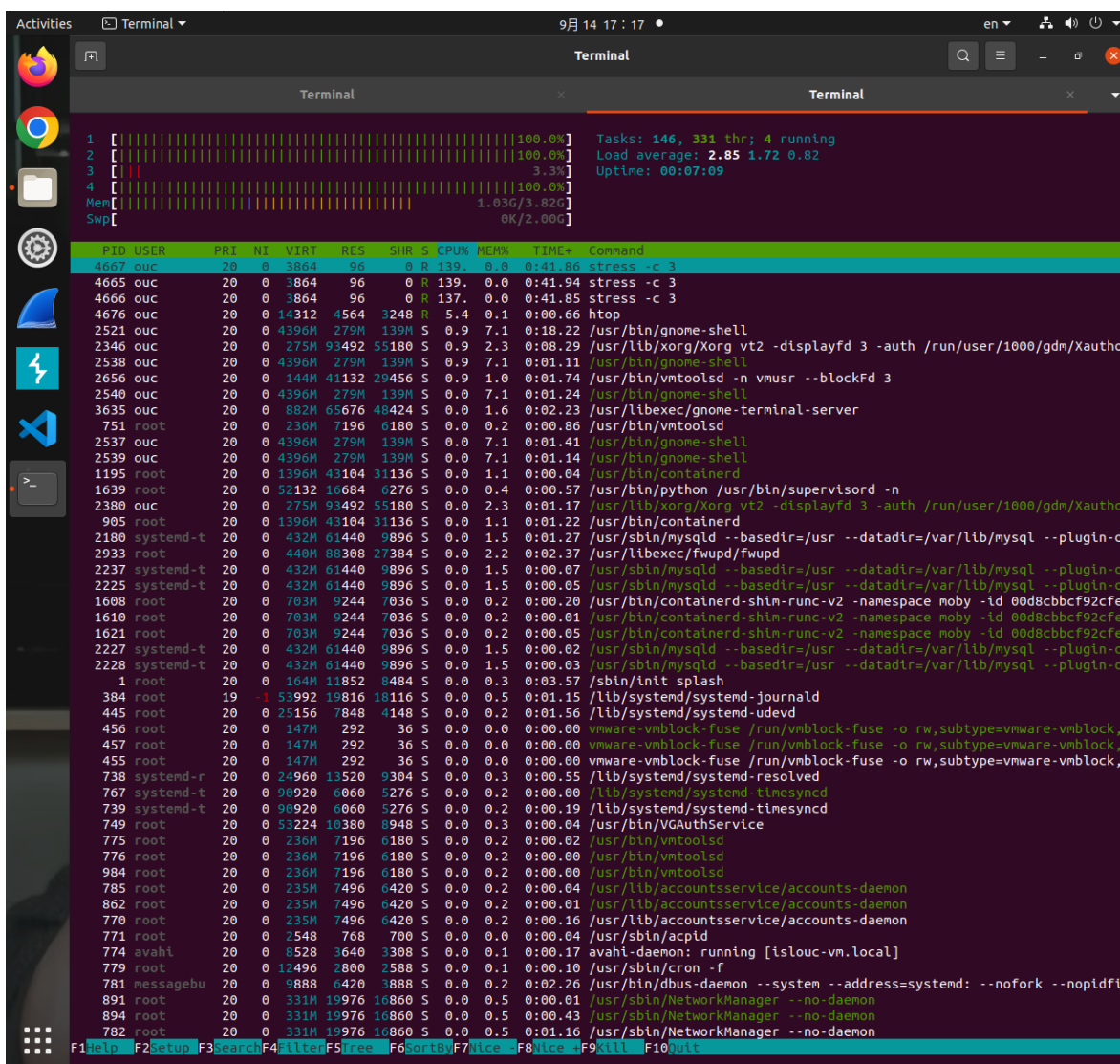


Figure 10: 3.1.5 创建负载

可以看到 stress 生成了 3 个 CPU 负载线程，且它们的负载会分布在所有可用的 CPU 上。
限制资源消耗：taskset -cpu-list 0,2 stress -c 3，查看其资源占用情况

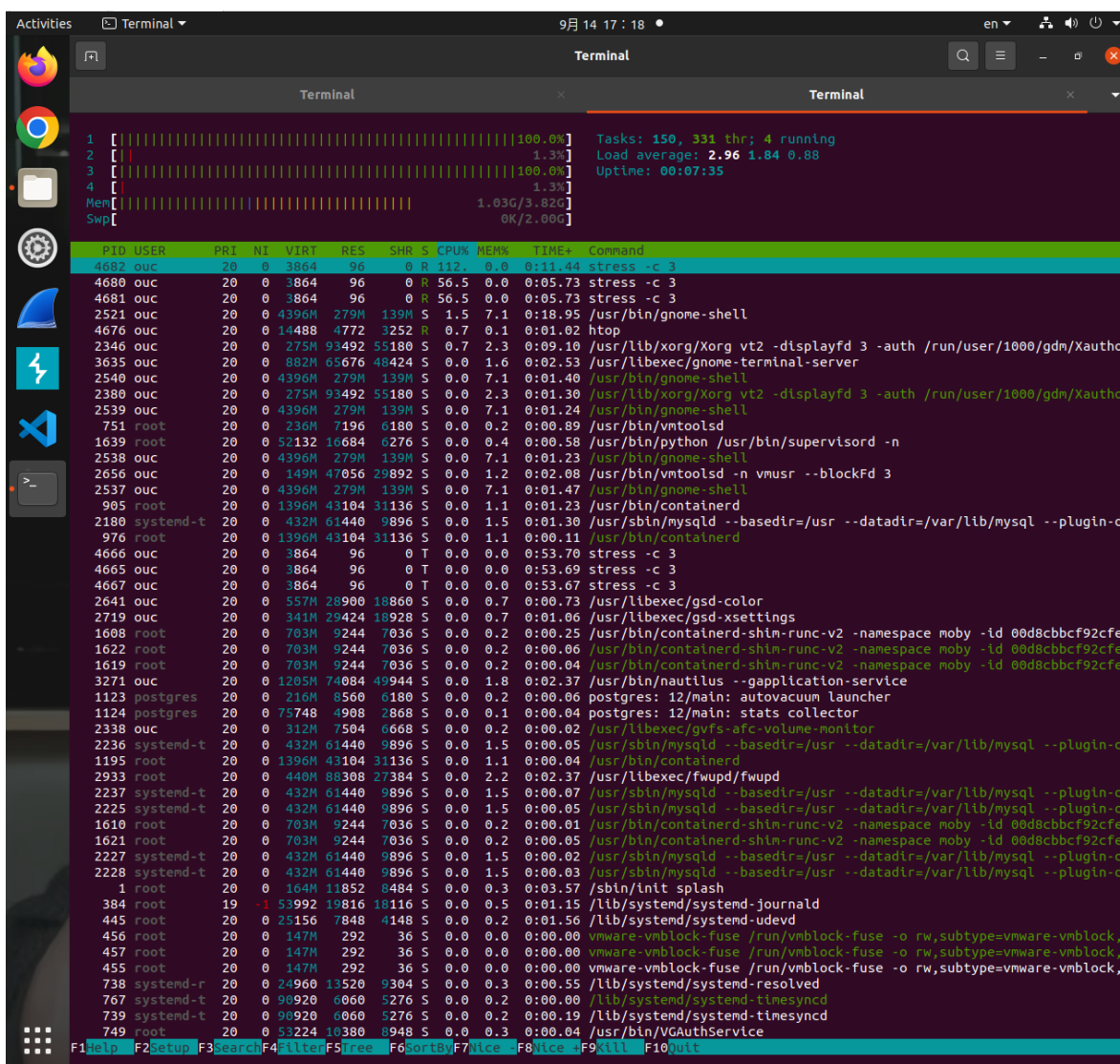


Figure 11: 3.1.5 限制资源消耗

可以看到 stress 进程的负载现在只在 CPU 0 和 CPU 2 上运行。CPU 1 和 CPU 3 没有负载，因为 stress 进程被限制在了 CPU 0 和 CPU 2 上。

3.1.6 curl ipinfo.io 命令或执行 HTTP 请求并获取关于您 IP 的信息

```
luc@isluc-vm:~/Desktop/systemtool/solution$ curl ipinfo.io
{
  "ip": "112.224.194.62",
  "city": "Shanghai",
  "region": "Shanghai",
  "country": "CN",
  "loc": "31.2222,121.4581",
  "org": "AS4837 CHINA UNICOM China169 Backbone",
  "postal": "200000",
  "timezone": "Asia/Shanghai",
  "readme": "https://ipinfo.io/missingauth"
```

Figure 12: 3.1.6 获取 IP 信息

3.2 元编程

3.2.1 清理文件

1. 准备工作环境:

可通过下面的命令检查工具是否安装

```
make --version
pdflatex --version
git --version
```

2. 创建 Latex 文件

创建一个 LaTeX 文件 paper.tex。这个文件将是你构建的源文件。

文件内容为:

```
\documentclass{article}
\begin{document}
\title{Sample Paper}
\author{Author Name}
\date{\today}
\maketitle

\section{Introduction}
This is a sample document.

\end{document}
```

3. 创建 Makefile

在相同的工作目录中, 创建一个名为 Makefile 的文件。这个文件将定义如何构建和清理文档。

在 Makefile 中添加以下内容:

```

# 定义要生成的目标
paper.pdf: paper.tex
    pdflatex paper.tex

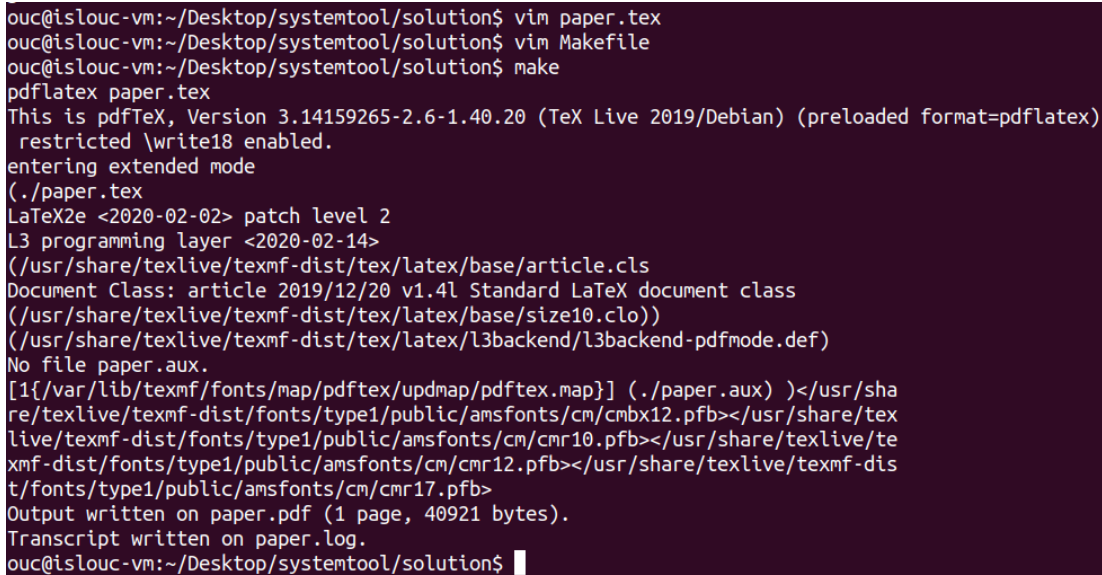
# 定义 phony 目标
.PHONY: clean

# 清理构建产生的文件
clean:
    rm -f paper.pdf paper.log paper.aux paper.out

```

4. 构建目标

使用 make 命令来构建 paper.pdf 文件



```

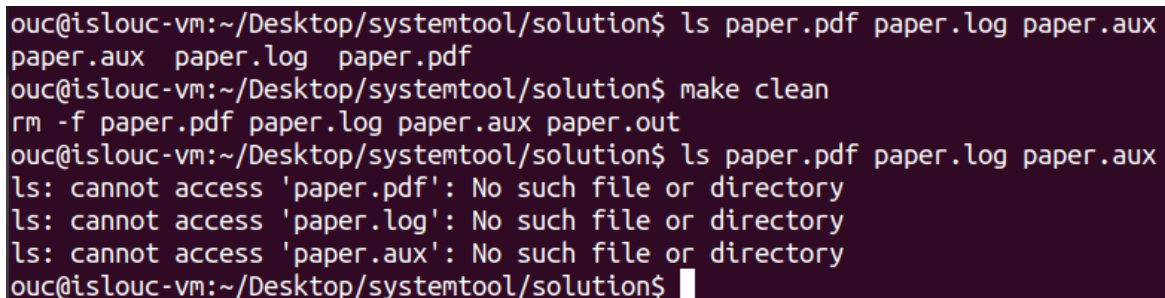
ouc@islouc-vm:~/Desktop/systemtool/solution$ vim paper.tex
ouc@islouc-vm:~/Desktop/systemtool/solution$ vim Makefile
ouc@islouc-vm:~/Desktop/systemtool/solution$ make
pdflatex paper.tex
This is pdfTeX, Version 3.14159265-2.6-1.40.20 (TeX Live 2019/Debian) (preloaded format=pdflatex)
 restricted \write18 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/size10.clo))
(/usr/share/texlive/texmf-dist/tex/latex/l3backend/l3backend-pdfmode.def)
No file paper.aux.
[1{/var/lib/texmf/fonts/map/pdftex/updmap/pdftex.map}] (./paper.aux) </usr/sha
re/texlive/texmf-dist/fonts/type1/public/amsfonts/cm/cmbx12.pfb></usr/share/tex
live/texmf-dist/fonts/type1/public/amsfonts/cm/cmr10.pfb></usr/share/texlive/te
xm-f-dist/fonts/type1/public/amsfonts/cm/cmr12.pfb></usr/share/texlive/texmf-dis
t/fonts/type1/public/amsfonts/cm/cmr17.pfb>
Output written on paper.pdf (1 page, 40921 bytes).
Transcript written on paper.log.
ouc@islouc-vm:~/Desktop/systemtool/solution$ █

```

Figure 13: 3.2.1 构建目标

5. 清理构建文件

使用 make clean 来清理过程中生成的文件。



```

ouc@islouc-vm:~/Desktop/systemtool/solution$ ls paper.pdf paper.log paper.aux
paper.aux paper.log paper.pdf
ouc@islouc-vm:~/Desktop/systemtool/solution$ make clean
rm -f paper.pdf paper.log paper.aux paper.out
ouc@islouc-vm:~/Desktop/systemtool/solution$ ls paper.pdf paper.log paper.aux
ls: cannot access 'paper.pdf': No such file or directory
ls: cannot access 'paper.log': No such file or directory
ls: cannot access 'paper.aux': No such file or directory
ouc@islouc-vm:~/Desktop/systemtool/solution$ █

```

Figure 14: 3.2.1 清理构建文件

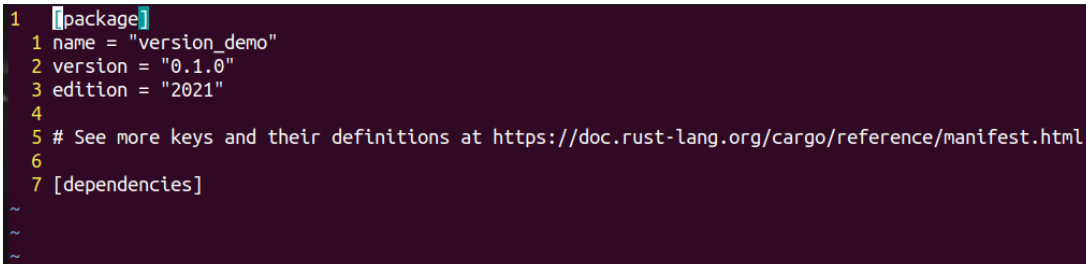
3.2.2 Rust 的构建系统

1. 创建一个 Rust 项目

```
cargo new version_demo
cd version_demo
```

这将创建一个名为 version_demo 的新目录，并在其中生成一个基本的 Rust 项目结构。

2. 编辑 Cargo.toml 文件，打开文件可以看到以下内容：



```
1 [[package]]
2 name = "version_demo"
3 version = "0.1.0"
4 edition = "2021"
5 # See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html
6
7 [dependencies]
~
~
~
```

Figure 15: 3.2.2 编辑 Cargo.toml 文件

在 Cargo.toml 文件中的 [dependencies] 部分，可以添加不同的版本要求，比如尖号 () 表示兼容指定版本及其向后兼容的版本，波浪号 () 表示指定一个版本范围其中包括指定版本和其向前兼容的版本，通配符 (*) 表示对版本号的任意匹配。

3. 更新依赖并构建项目：cargo build

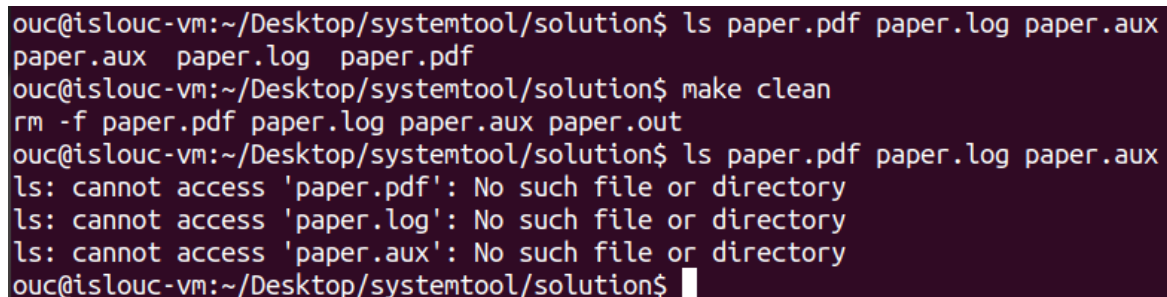
4. 测试版本要求

在 src/main.rs 文件中，添加一些代码来使用 regex crate 以测试是否工作正常：

```
use regex::Regex;

fn main() {
    let re = Regex::new(r"^\d{4}-\d{2}-\d{2}$").unwrap();
    let date = "2024-09-14";
    println!("Is '{}' a valid date? {}", date, re.is_match(date));
}
```

然后运行程序



```
ouc@islouc-vm:~/Desktop/systemtool/solution$ ls paper.pdf paper.log paper.aux
paper.aux paper.log paper.pdf
ouc@islouc-vm:~/Desktop/systemtool/solution$ make clean
rm -f paper.pdf paper.log paper.aux paper.out
ouc@islouc-vm:~/Desktop/systemtool/solution$ ls paper.pdf paper.log paper.aux
ls: cannot access 'paper.pdf': No such file or directory
ls: cannot access 'paper.log': No such file or directory
ls: cannot access 'paper.aux': No such file or directory
ouc@islouc-vm:~/Desktop/systemtool/solution$
```

Figure 16: 3.2.1 测试版本要求

3.2.3 pre-commit 钩子

1. 进入 Git 仓库的.git/hooks 目录
cd .git/hooks
2. 创建并编辑 pre-commit 文件
3. 编辑 pre-commit 文件

```
#!/bin/sh

# 执行 make paper.pdf
make paper.pdf

# 检查 make 命令的返回状态
if [ $? -ne 0 ]; then
    echo "构建失败，提交被拒绝。请修复构建错误后再提交。"
    exit 1
fi

# 如果构建成功，允许提交
exit 0
```

A screenshot of a terminal window with a dark purple background. The title bar at the top shows 'GNU nano 4.8' on the left and 'pre' on the right. The terminal displays the same script as the one above, with some lines highlighted in green: the shebang, the first comment, the 'make' command, the second comment, the 'if' statement, the 'echo' command, the 'fi' statement, the third comment, and the 'exit 0' command. The cursor is at the end of the last line.

```
GNU nano 4.8 pre
#!/bin/sh

# 执行 make paper.pdf
make paper.pdf

# 检查 make 命令的返回状态
if [ $? -ne 0 ]; then
    echo "构建失败，提交被拒绝。请修复构建错误后再提交。"
    exit 1
fi

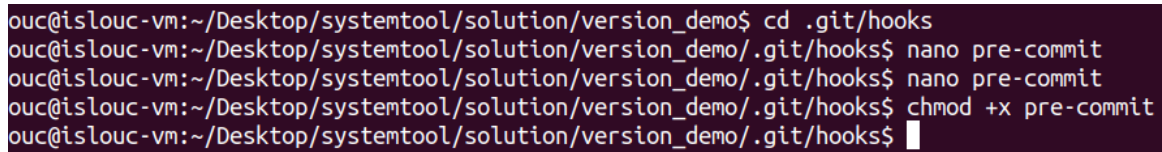
# 如果构建成功，允许提交
exit 0
```

Figure 17: 3.2.3 编辑文件

保存文件并提出编译器。

4. 使 pre-commit 文件可执行

```
chmod +x pre-commit
```



```
ouc@islouc-vm:~/Desktop/systemtool/solution/version_demo$ cd .git/hooks
ouc@islouc-vm:~/Desktop/systemtool/solution/version_demo/.git/hooks$ nano pre-commit
ouc@islouc-vm:~/Desktop/systemtool/solution/version_demo/.git/hooks$ nano pre-commit
ouc@islouc-vm:~/Desktop/systemtool/solution/version_demo/.git/hooks$ chmod +x pre-commit
ouc@islouc-vm:~/Desktop/systemtool/solution/version_demo/.git/hooks$
```

Figure 18: 3.2.3 执行文件

这样配置之后，每次执行 `git commit` 时，Git 都会在提交之前运行 `make paper.pdf`，如果构建失败，提交将被拒绝。

3.3 大杂烩

3.3.1 修改键位映射

1. 查看当前键位映射


```

ouc@islouc-vm:~/Desktop/systemtool/solution$ xmodmap -pke
keycode 8 = 
keycode 9 = Escape NoSymbol Escape
keycode 10 = 1 exclam 1 exclam
keycode 11 = 2 at 2 at
keycode 12 = 3 numbersign 3 numbersign
keycode 13 = 4 dollar 4 dollar
keycode 14 = 5 percent 5 percent
keycode 15 = 6 asciicircum 6 asciicircum
keycode 16 = 7 ampersand 7 ampersand
keycode 17 = 8 asterisk 8 asterisk
keycode 18 = 9 parenleft 9 parenleft
keycode 19 = 0 parenright 0 parenright
keycode 20 = minus underscore minus underscore
keycode 21 = equal plus equal plus
keycode 22 = BackSpace BackSpace BackSpace BackSpace
keycode 23 = Tab ISO_Left_Tab Tab ISO_Left_Tab
keycode 24 = q Q q Q
keycode 25 = w W w W
keycode 26 = e E e E
keycode 27 = r R r R
keycode 28 = t T t T
keycode 29 = y Y y Y
keycode 30 = u U u U
keycode 31 = i I i I
keycode 32 = o O o O
keycode 33 = p P p P
keycode 34 = bracketleft braceleft bracketleft braceleft
keycode 35 = bracketright braceright bracketright braceright
keycode 36 = Return NoSymbol Return
keycode 37 = Control_L NoSymbol Control_L
keycode 38 = a A a A
keycode 39 = s S s S
keycode 40 = d D d D
keycode 41 = f F f F
keycode 42 = g G g G
keycode 43 = h H h H
keycode 44 = j J j J
keycode 45 = k K k K
keycode 46 = l L l L
keycode 47 = semicolon colon semicolon colon
keycode 48 = apostrophe quotedbl apostrophe quotedbl
keycode 49 = grave asciitilde grave asciitilde
keycode 50 = Shift_L NoSymbol Shift_L
keycode 51 = backslash bar backslash bar
keycode 52 = z Z z Z
keycode 53 = x X x X
keycode 54 = c C c C

```

Figure 19: 3.3.1 查看当前键位映射

2. 创建一个新的键位映射文件

编辑 `/.Xmodmap` 文件并添加映射。

```
clear Lock
```

```
keycode 66 = Escape
```

3. 应用新的键位映射: `xmodmap /.Xmodmap`

查看新的键位映射可以看到 Caps Lock 键已映射为 Escape


```

keycode 56 = b B b B
keycode 57 = n N n N
keycode 58 = m M m M
keycode 59 = comma less comma less
keycode 60 = period greater period greater
keycode 61 = slash question slash question
keycode 62 = Shift_R NoSymbol Shift_R
keycode 63 = KP_Multiply KP_Multiply KP_Multiply KP_Multiply KP_Multiply KP_Multiply XF86ClearGrab
keycode 64 = Alt_L Meta_L Alt_L Meta_L
keycode 65 = space NoSymbol space
keycode 66 = Escape NoSymbol Escape
keycode 67 = F1 F1 F1 F1 F1 F1 XF86Switch_VT_1
keycode 68 = F2 F2 F2 F2 F2 F2 XF86Switch_VT_2
keycode 69 = F3 F3 F3 F3 F3 F3 XF86Switch_VT_3
keycode 70 = F4 F4 F4 F4 F4 F4 XF86Switch_VT_4
keycode 71 = F5 F5 F5 F5 F5 F5 XF86Switch_VT_5
keycode 72 = F6 F6 F6 F6 F6 F6 XF86Switch_VT_6
keycode 73 = F7 F7 F7 F7 F7 F7 XF86Switch_VT_7
keycode 74 = F8 F8 F8 F8 F8 F8 XF86Switch_VT_8
keycode 75 = F9 F9 F9 F9 F9 F9 XF86Switch_VT_9
keycode 76 = F10 F10 F10 F10 F10 F10 XF86Switch_VT_10
keycode 77 = Num_Lock NoSymbol Num_Lock

```

Figure 20: 3.3.1.2 应用新的键位映射

3.3.2 守护进程

编辑 crontab 文件，添加一行以定期启动守护进程：

```
@reboot /path/to/simple_daemon
```

3.3.3 FUSE

1. 安装 FUSE

```
sudo apt-get install fuse libfuse-dev
```

2. 编写一个简单的 FUSE 文件系统

```

#define FUSE_USE_VERSION 31

#include <fuse.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>

static const char *hello_str = "Hello, World!\n";
static const char *hello_path = "/hello";

static int hello_getattr(const char *path, struct stat *stbuf) {
    int res = 0;

    memset(stbuf, 0, sizeof(struct stat));
    if (strcmp(path, "/") == 0) {
        stbuf->st_mode = S_IFDIR | 0755;
    }
}

```

```

        stbuf->st_nlink = 2;
    } else if (strcmp(path, hello_path) == 0) {
        stbuf->st_mode = S_IFREG | 0444;
        stbuf->st_nlink = 1;
        stbuf->st_size = strlen(hello_str);
    } else {
        res = -ENOENT;
    }

    return res;
}

static int hello_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
                        off_t offset, struct fuse_file_info *fi) {
    (void) offset;
    (void) fi;

    if (strcmp(path, "/") != 0)
        return -ENOENT;

    filler(buf, ".", NULL, 0);
    filler(buf, "..", NULL, 0);
    filler(buf, hello_path + 1, NULL, 0);

    return 0;
}

static int hello_open(const char *path, struct fuse_file_info *fi) {
    if (strcmp(path, hello_path) != 0)
        return -ENOENT;

    if ((fi->flags & 3) != O_RDONLY)
        return -EACCES;

    return 0;
}

static int hello_read(const char *path, char *buf, size_t size, off_t offset,
                    struct fuse_file_info *fi) {
    size_t len;
    (void) fi;

```

```

    if(strcmp(path, hello_path) != 0)
        return -ENOENT;

    len = strlen(hello_str);
    if (offset < len) {
        if (offset + size > len)
            size = len - offset;
        memcpy(buf, hello_str + offset, size);
    } else {
        size = 0;
    }

    return size;
}

static struct fuse_operations hello_oper = {
    .getattr = hello_getattr,
    .readdir = hello_readdir,
    .open = hello_open,
    .read = hello_read,
};

int main(int argc, char *argv[]) {
    return fuse_main(argc, argv, &hello_oper, NULL);
}

```

3. 编译运行该文件系统

```

ouc@islouc-vm:~/Desktop/systemtool/solution$ vim simple_fs.c
ouc@islouc-vm:~/Desktop/systemtool/solution$ gcc -Wall simple_fs.c `pkg-config fuse --cflags --libs` -o simple_fs
ouc@islouc-vm:~/Desktop/systemtool/solution$ mkdir /tmp/fuse_mount
ouc@islouc-vm:~/Desktop/systemtool/solution$ ./simple_fs /tmp/fuse_mount
ouc@islouc-vm:~/Desktop/systemtool/solution$ cat /tmp/fuse_mount/hello
Hello, World!
ouc@islouc-vm:~/Desktop/systemtool/solution$

```

Figure 21: 3.3.3 FUSE

4. 卸载 FUSE 文件

```
fusermount -u /tmp/fuse_mount
```

3.3.4 API (应用程序接口)

以美国天气数据为例，为了获得某个地点的天气数据，可以发送一个 GET 请求（比如使用 curl）到 <https://api.weather.gov/points/42.3604,-71.094>。返回中会包括一系列用于获取特定信息（比如小时预报、气象观察站信息等）的 URL。通常这些返回都是 JSON 格式，可以使用 jq 等工具来选取需要的部分。

1. 首先, 使用 curl 发送 GET 请求到 <https://api.weather.gov/points/42.3604,-71.094>, 获取波士顿的天气数据。

```
ouc@islouc-vm:~/Desktop/systemtool/solution$ curl -X GET "https://api.weather.gov/points/42.3604,-71.094"
{
  "@context": [
    "https://geojson.org/geojson-ld/geojson-context.jsonld",
    {
      "@version": "1.1",
      "wx": "https://api.weather.gov/ontology#",
      "s": "https://schema.org/",
      "geo": "http://www.opengis.net/ont/geosparql#",
      "unit": "http://codes.wmo.int/common/unit/",
      "@vocab": "https://api.weather.gov/ontology#",
      "geometry": {
        "@id": "s:GeoCoordinates",
        "@type": "geo:wktLiteral"
      },
      "city": "s:addressLocality",
      "state": "s:addressRegion",
      "distance": {
        "@id": "s:Distance",
        "@type": "s:QuantitativeValue"
      },
      "bearing": {
        "@type": "s:QuantitativeValue"
      },
      "value": {
        "@id": "s:value"
      },
      "unitCode": {
        "@id": "s:unitCode",
        "@type": "@id"
      },
      "forecastOffice": {
        "@type": "@id"
      },
      "forecastGridData": {
        "@type": "@id"
      },
      "publicZone": {
        "@type": "@id"
      },
      "county": {
        "@type": "@id"
      }
    }
  ],
  "id": "https://api.weather.gov/points/42.3604,-71.094",
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [
      -71.09399999999994,
      42.360399999999998
    ]
  }
}
```

Figure 22: 3.3.4 发送 GET 请求

2. 解析返回的 JSON 数据
提取小时预报的 URL

```
ouc@islouc-vm:~/Desktop/systemtool/solution$ curl -s "https://api.weather.gov/points/42.3604,-71.094" | jq -r '.properties.forecastHourly'
https://api.weather.gov/gridpoints/BOX/70,90/forecast/hourly
```

Figure 23: 3.3.4 解析返回的 JSON 数据

3. 获取小时预报数据

使用上一步中提取的 URL, 再次发送 GET 请求以获取小时预报数据。

```

ouc@islouc-vm:~/Desktop/systemtool/solution$ curl -s "https://api.weather.gov/gridpoints/BOX/70,75/forecast/hourly" | jq
{
  "@context": [
    "https://geojson.org/geojson-ld/geojson-context.jsonld",
    {
      "@version": "1.1",
      "wx": "https://api.weather.gov/ontology#",
      "geo": "http://www.opengis.net/ont/geosparql#",
      "unit": "http://codes.wmo.int/common/unit/",
      "@vocab": "https://api.weather.gov/ontology#"
    }
  ],
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [
          -71.1884662,
          42.0570161
        ],
        [
          -71.1935905,
          42.0355711
        ],
        [
          -71.16470819999999,
          42.0317626
        ],
        [
          -71.159578,
          42.0532072
        ],
        [
          -71.1884662,
          42.0570161
        ]
      ]
    ]
  },
  "properties": {
    "units": "us",
    "forecastGenerator": "HourlyForecastGenerator",
    "generatedAt": "2024-09-14T14:09:46+00:00",
    "updateTime": "2024-09-14T13:08:16+00:00",
    "validTimes": "2024-09-14T07:00:00+00:00/P8DT6H",
    "elevation": {
      "unitCode": "wmoUnit:m",
      "value": 53.9496
    }
  }
},

```

Figure 24: 3.3.4 获取小时预报数据

4. 提取特定信息

可以使用 jq 进一步提取特定的信息，下面是当前的温度和天气描述

```

ouc@islouc-vm:~/Desktop/systemtool/solution$ curl -s "https://api.weather.gov/gridpoints/BOX/70,75/forecast/hourly" | jq -r '.properties.periods[0] | {temperature: .temperature, shortForecast: .shortForecast}'
{
  "temperature": 72,
  "shortForecast": "Sunny"
}

```

Figure 25: 3.3.4 提取特定信息

5. 写一个脚本实现

```
#!/bin/bash
```

```
# 获取波士顿的天气数据
```

```
response=$(curl -s "https://api.weather.gov/points/42.3604,-71.094")
```

```
# 提取小时预报的 URL
```

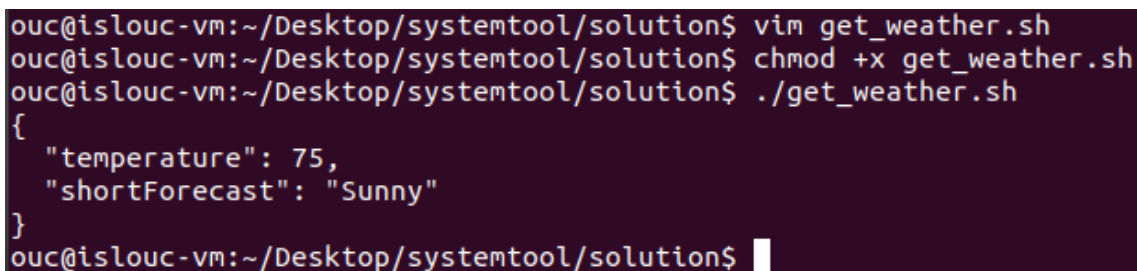
```
forecast_url=$(echo "$response" | jq -r '.properties.forecastHourly')
```

```
# 获取小时预报数据
forecast_data=$(curl -s "$forecast_url")

# 提取当前的温度和天气描述
current_weather=$(echo "$forecast_data" | jq -r '.properties.periods[0] | {temperature: .temper

# 输出结果
echo "$current_weather"
```

6. 运行脚本



```
ouc@islouc-vm:~/Desktop/systemtool/solution$ vim get_weather.sh
ouc@islouc-vm:~/Desktop/systemtool/solution$ chmod +x get_weather.sh
ouc@islouc-vm:~/Desktop/systemtool/solution$ ./get_weather.sh
{
  "temperature": 75,
  "shortForecast": "Sunny"
}
ouc@islouc-vm:~/Desktop/systemtool/solution$
```

Figure 26: 3.3.4 运行脚本

4 实验总结

通过本实验，我学习到了调试和性能分析相关知识，此外，我掌握了如何利用元编程技术提升代码的灵活性和复用性，通过实践了解了装饰器和元类的应用场景和优势。通过这些学习，我能够更深入地优化和改进我的代码，提升编程效率和程序的整体性能。通过大杂烩部分不仅复习了之前学到的一些知识，也更学会了其他的系统实用工具，比如 API、键位映射等。总之，实验完成虽然比较困难但是仍感觉受益非亲。

5 github 链接

<https://github.com/Caoyu2233/Systemtools.git>