实验报告

陈俊泽

August 2024

目录

1 练习实例

课后题部分
调试部分
实例 1（第一题）：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。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：使用的命令为 journalctl –since ”yesterday”| grep sudo，journalctl 找到了登录信息及其所执行的指令，since ”yesterday” 限定了时间为最近一天，最后使用 grep sudo 筛选了超级用户来进行输出。

实例 2（第二题）：学习这份 pdb 实践教程并熟悉相关的命令。更深入的信息您可以参考这份教程：pdb 和 gdb 的使用方法类似，如果要打断点，则可以在程序中找到想要打断点的地方，输入 pdb.set_trace()，在启动程序后，不同的参数代表着不同的操作。n：下一条命令，c：到下一个断点，p：打印变量值，q：退出等等

实例 3（第三题）：安装 shellcheck 并尝试对下面的脚本进行检查。这段代码有什么问题吗？请修复相关问题。在您的编辑器中安装一个 linter 插

件，这样它就可以自动地显示相关警告信：首先用 install shellcheck 进行安装，之后用 shellcheck a.sh 进行检查，出现了三个问题，第一是不能直接使用 ls，应修改为 for f in ./*.m3u;。第二个问题为应对参数加上引号，应修改为 if grep -qi 'hq.mp3' "$f";。第三个问题是应该使用双引号而不是单引号，应修改为 echo "Playlist $f contains a HQ file in mp3 format"。

性能分析部分

实例 4(第一题): 这里有一些排序算法的实现。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: 先使用命令 python -m cProfile -s time sorts.py，将函数按照执行时间排序进行输出，可以看到快排慢于插排。再使用 kernprof -l -v sorts.py，通过结果可以看到插排的瓶颈在 while 循环，快排的瓶颈在左右赋值。

实例 5（第二题）：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(如果您能够执行 dot, 则说明已经安装了 GraphViz.)。并使用 pycallgraph graphviz – ./fib.py 来执行代码并查看 pycallgraph.png 这个文件。fib0 被调用了多少次？我们可以通过记忆法来对其进行优化。将注释掉的部分放开，然后重新生成图片。这回每个 fibN 函数被调用了多少次: 先用命令 pip install "setuptools<58.0.0" 和 pip install pycallgraph 安装 pycallgraph，再手动安装 graphviz，最后再使用 pycallgraph graphviz – ./fib.py 来执行代码并查看 pycallgraph.png 这个文件即可。

实例 6（第三题）：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：先用 python3 -m http.server 4444 命令启动一个对 4444 端口监听的 web 服务，再使用 lsof | grep LISTEN 打印端口，找到此端口的 pid 号，最后使用

kill pid 号就可以终止该进程。

实例 7（第四题）：限制进程资源也是一个非常有用的技术。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？阅读 man taskset 来寻找答案：首先先用命令 stress -c 3 来启动负载，再利用命令 taskset –cpu-list 0,2 stress -c 3 来限制限制资源消耗。通过查看，发现没有占用 3 个 cpu，原因如下：stress -c 3 请求创建 3 个 CPU 密集型线程，但 taskset –cpu-list 0,2 将这些线程的执行限制在 CPU 0 和 CPU 2 上。因此，即使 stress 试图使用 3 个线程，由于 CPU 限制，它只能在 2 个 CPU 上运行这些线程，不会使用到第 3 个 CPU。

实例 8（第五题）:curl ipinfo.io 命令或执行 HTTP 请求并获取关于您 IP 的信息。打开 Wireshark 并抓取 curl 发起的请求和收到的回复报文。（提示：可以使用 http 进行过滤，只显示 HTTP 报文）：在 windows 下打开命令提示符，输入命令 curl www.baidu.com，同时打开 wireshark 进行抓包即可。

元编程部分

实例 9（第一题）：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。您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。您需要将构建目标设置为 phony。: 首先用命令 brew cask install basictex 安装 basictex，然后编写 Makefile，用 git ls-files -o | xargs rm -f 命令列出没有被 git 追踪的文件即可。

实例 10（第三题）: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。这样做可以避免产生包含不可构建版本的提交信息：修改 pre-commit.sample 文件并将其命名为 pre-commit 即可，代码如下：

```
if ! make ; then
echo ”build failed, commit rejected”
```

exit 1
fi

实例 11（第五题）：构建属于您的 GitHub action，对仓库中所有的.md 文件执行 proselint 或 write-good，在您的仓库中开启这一功能，提交一个包含错误的文件看看该功能是否生效：在 Github marketplace 中，可以找到，Lint Markdown 修改 blank.yml 即可。

大杂烩部分
实例 12: 问题：如何将 Caps Lock 键映射为 Ctrl 键以提高编程效率?：解决方法如下：在 Windows 上，可以使用 AutoHotkey 创建一个脚本文件，内容如下：CapsLock::Ctrl，保存并运行此脚本，即可将 Caps Lock 映射为 Ctrl 键。

实例 13：问题：如何检查并重新启动 Linux 系统上的 SSH 服务?：解决方法如下：使用 systemctl 命令。首先，检查 SSH 服务的状态：systemctl status sshd，如果服务没有运行或需要重启，可以使用以下命令：systemctl restart sshd

实例 14: 问题：如何使用 API 从天气服务中获取特定地点的天气数据?：解决方法如下：使用 curl 命令向 API 发送 GET 请求。若要获取一个特定地点的天气预报数据，可以使用以下命令：curl https://api.weather.gov/points/42.3604,-71.094 此命令会返回 JSON 格式的数据，其中包括获取天气预报所需的进一步 URL。可以使用 jq 工具来提取具体信息 curl https://api.weather.gov/points/42.3604,-71.094 | jq '.properties.periods[0].temperature'

实例 15：问题：如何使用命令行工具进行安全的文件删除，并且确认删除操作? 解决方法如下：使用 rm 命令进行文件删除，并利用 –dry-run 参数进行模拟删除。实际删除之前，先执行模拟操作来确认将要删除的文件：rm –dry-run -r /path/to/directory 这样可以看到哪些文件将被删除而不实际删除它们。如果确认没有问题，再执行实际删除操作：rm -r /path/to/directory 对于需要确认的删除操作，可以使用 -i 参数，逐个确认每个文件的删除：rm -i -r /path/to/directory 这样可以确保每个文件在删除前都需要用户确认。

实例 16: 问题: 如何在 Markdown 文档中插入图片和链接?解决方法如下：插入图片: 使用以下语法插入图片：![alt text](URL), 其中 alt text 是图片的替代文字, URL 是图片的链接。插入链接: 使用以下语法插入链接：链接文字, 其中链接文字是显示在文档中的文本，URL 是目标网页的链接。实例: 你正在编写一个项目文档，需要插入一张截图和一个相关链接。在 Markdown 文档中，你可以写：![Screenshot](https://example.com/screenshot.png) 来插入图片, 和 [Project Repository](https://github.com/example/repo) 来插入链接。

实例 17：问题: 如何使用 Hammerspoon 自动将窗口移动到特定位置? 解决方法如下: 使用 Hammerspoon 编写一个 Lua 脚本，利用 hs.window 模块来设置窗口的位置。
hs.hotkey.bind({”cmd”, ”alt”, ”ctrl”}, ”M”, function()
local win = hs.window.frontmostWindow()
if win then
win:moveToScreen(hs.screen.mainScreen())
win:move(hs.geometry.rect(100, 100, 800, 600))
end end)

实例 18: 问题: 如何使用 Vagrant 创建一个虚拟开发环境? 解决方法如下: 编写 Vagrantfile: 创建一个 Vagrantfile 来定义虚拟机的配置，包括操作系统、内存、CPU 等。实现效果如下
ruby
Vagrant.configure(”2”) do |config|
config.vm.box = ”ubuntu/bionic64”
config.vm.network ”forwarded_port”, guest: 80, host: 8080
end 在包含 Vagrantfile 的目录中运行 vagrant up 命令来启动虚拟机。Vagrant 将自动下载所需的 box 并配置虚拟机。

实例 19：问题：如何将 PyTorch 的 Tensor 转换为 Numpy 数组，并验证两者内存共享? 解决方法如下：使用 tensor.numpy() 方法将 Tensor 转换为 Numpy 数组。代码如下：python
import torch

# 创建一个 Tensor
a = torch.ones(5)
print(”Tensor a:”, a)
# 转换为 Numpy 数组
b = a.numpy()
print(”Numpy array b:”, b)
# 修改 Tensor a 的值
a.add_(1)
print(”Modified Tensor a:”, a)
print(”Modified Numpy array b:”, b)
修改 Tensor 后，Numpy 数组的值也会相应变化，证明它们共享内存空间。

实例 20: 问题：如何定义一个简单的卷积神经网络来处理 CIFAR10 数据集？解决方法如下：创建一个包含两个卷积层和三个全连接层的网络，适应彩色图片的输入通道。示例代码如下：import torch.nn as nn
import torch.nn.functional as F
class Net(nn.Module):
def _init_(self):
super(Net, self)._init_()
self.conv1 = nn.Conv2d(3, 6, 5)
self.pool = nn.MaxPool2d(2, 2)
self.conv2 = nn.Conv2d(6, 16, 5)
self.fc1 = nn.Linear(16 * 5 * 5, 120)
self.fc2 = nn.Linear(120, 84)
self.fc3 = nn.Linear(84, 10)

def forward(self, x):
x = self.pool(F.relu(self.conv1(x)))
x = self.pool(F.relu(self.conv2(x)))
x = x.view(-1, 16 * 5 * 5)
x = F.relu(self.fc1(x))
x = F.relu(self.fc2(x))
x = self.fc3(x)

```
return x

    net = Net()
```

2 解题感悟

在这次实验中，通过实践获得了宝贵的经验。首先，调试技术是实验的一部分，我学会了如何使用 pdb.set_trace() 设置断点，这对于逐步调试和查看变量状态极为重要。掌握了命令如 n（下一条命令）、c（继续到下一个断点）和 p（打印变量值），这些对定位和修复程序中的错误提供了很大帮助。

在对脚本进行静态检查时，我使用了 shellcheck 工具，这帮助我识别并修复了脚本中的潜在问题。例如，我改正了对 ls 的直接使用，改为使用 for 循环来处理文件，并且在条件判断和输出信息中正确使用了引号和双引号。这些修正使得脚本更加健壮和可靠。

性能分析部分让我对不同排序算法的性能有了更深入的了解。使用 cProfile 和 line_profiler 对比插入排序和快速排序的性能瓶颈，发现插入排序在 while 循环中瓶颈较大，而快速排序则在左右赋值中瓶颈。通过 memory_profiler 的分析，我了解了内存消耗情况，并发现原地排序的快排在内存使用上更优。此外，通过 pycallgraph 和 graphviz 工具对斐波那契数列计算程序进行了性能优化。记忆法的应用显著减少了函数调用的次数，提高了计算效率。最后在 pytouch 部分，我学习了如何用 pytest 执行单元测试，并结合 pytest 的插件进行测试覆盖率分析。创建测试用例并通过 pytest 验证功能的正确性，对发现和修复潜在问题大有裨益。