第四次实验报告

姓名：刘昱莹　　学号：22010022040

2024 年 9 月 15 日

目录

1 调试及性能分析

1.1 练习内容

1.1.1 打印调试法与日志

使用 Python3 运行学习资源中这部分的 logger.py 文件，实现不同形式的日志的输出。

1.1.2 在终端打印多种颜色

下方代码通过 seq 0 20 255 循环生成从 0 到 255、每次间隔 20 的数字序列，并分别依次赋值给 R、G、B，根据不同的 RGB 打印不同的颜色。

Listing 1: 多种颜色的输出

```
#!/usr/bin/env bash
for R in $(seq 0 20 255); do
    for G in $(seq 0 20 255); do
        for B in $(seq 0 20 255); do
            printf "\e[38;2;${R};${G};${B}m \e[0m";
        done
    done
done
```

1.1.3 第三方日志系统

使用 logger 这个 shell 程序向系统日志中写入 “Hello Logs”，并使用 journalctl 命令在 1 分钟之内的日志中查找 “Hello”。

1.1.4 调试器

对于以下冒泡排序的代码使用 Python 的调试器 pdb 进行调试。

-l：显示当前行附近的 11 行或继续执行之前的显示

-p：在当前上下文对表达式求值并打印结果

-s：执行当前行，并在第一个可能的地方停止

-n：继续执行直到当前函数的下一条语句或者 return 语句

-r：继续执行直到当前函数返回

-q：退出调试器

Listing 2: pdb 测试代码

```
import pdb                  #引入pdb库
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        pdb.set_trace()  #在这里设置断点
        for j in range(n):
            if arr[j] > arr[j+1]:
                arr[j] = arr[j+1]
```

```
                arr[j+1] = arr[j]
    return arr
print(bubble_sort([4, 2, 1, 8, 7, 6]))
```

1.1.5 使用 time 模块进行性能分析

使用 Python 中的 time 模块通过打印程序的运行时间对 sort.py 进行性能分析。

1.1.6 使用 cProfile 进行性能分析

使用 cProfile 库对 sort.py 进行性能分析。

1.2 结果

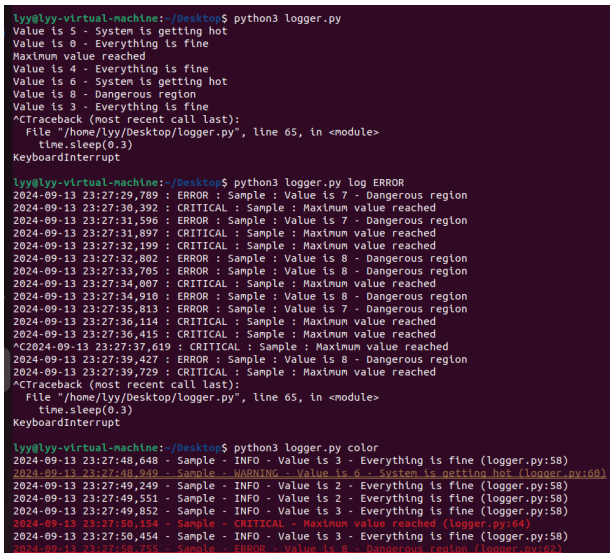

图 1: 打印调试法与日志

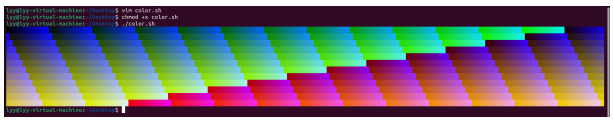

图 2: 在终端打印多种颜色

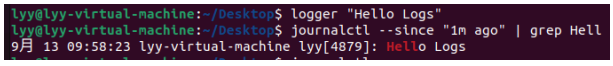

图 3: 第三方日志系统

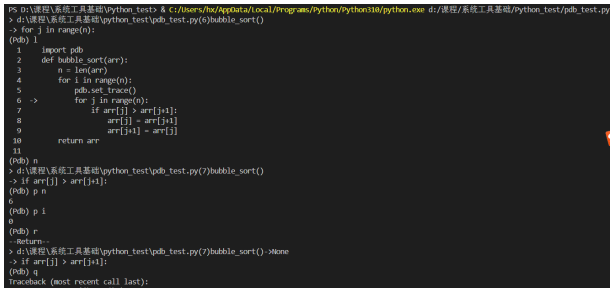

图 4: pdb 调试结果

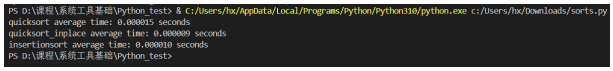

图 5: time 进行性能分析结果

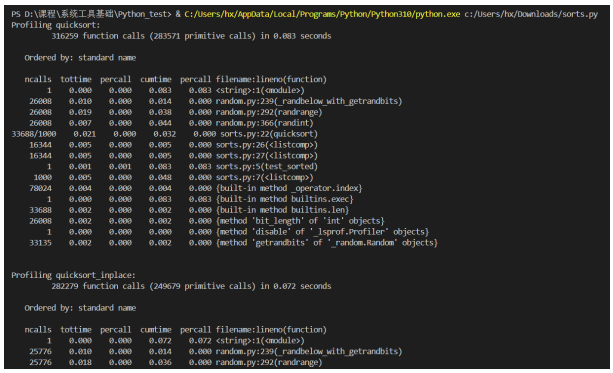

图 6: cProfile 进行性能分析结果

2 元编程

2.1 练习内容

2.1.1 构建系统

1、编写 Makefile 文件使用 make 命令构建 paper.pdf。

2、使用 make 命令构建伪目标删除构建过程中产生的文件。

2.1.2 依赖管理

在学习本节之前，我通过软件包下载过很多软件，但从没有思考过为什么是“包”，通过阅读此部分内容我了解到包里面是该软件的依赖文件。

我还了解到了版本号的概念以及相对比较常用的标准语义版本号格式：主版本号. 次版本号. 补丁号。如果新的版本没有改变 API，补丁号递增；如果添加了 API 并且该改动是向后兼容的，次版本号递增；如果修改了 API 但是它并不向后兼容，主版本号递增。

2.1.3 测试简介

我了解了不同的测试方法:

（1）单元测试（Unit test）：一种“微型测试”，用于对某个封装的特性进行测试。

（2）集成测试（Integration test）：一种“宏观测试”，针对系统的某一大部分进行，测试其不同的特性或组件是否能协同工作。

（3）回归测试（Regression test）：一种实现特定模式的测试，用于保证之前引起问题的 bug 不会再次出现。我还了解了测试套件以及模拟的语义。

2.2 结果

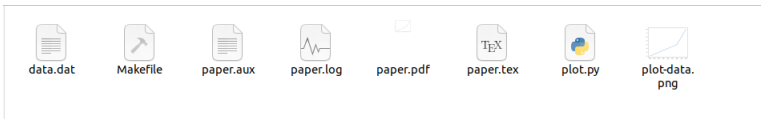

图 7: 构建过程的依赖文件及生成文件

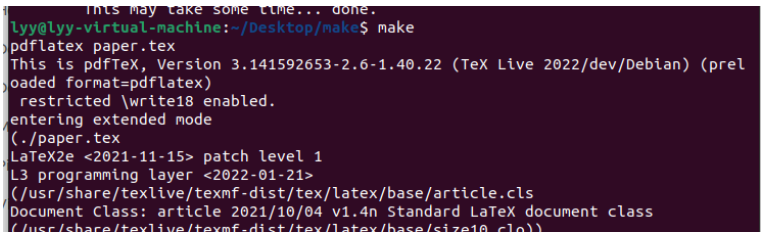

图 8: 构建结果

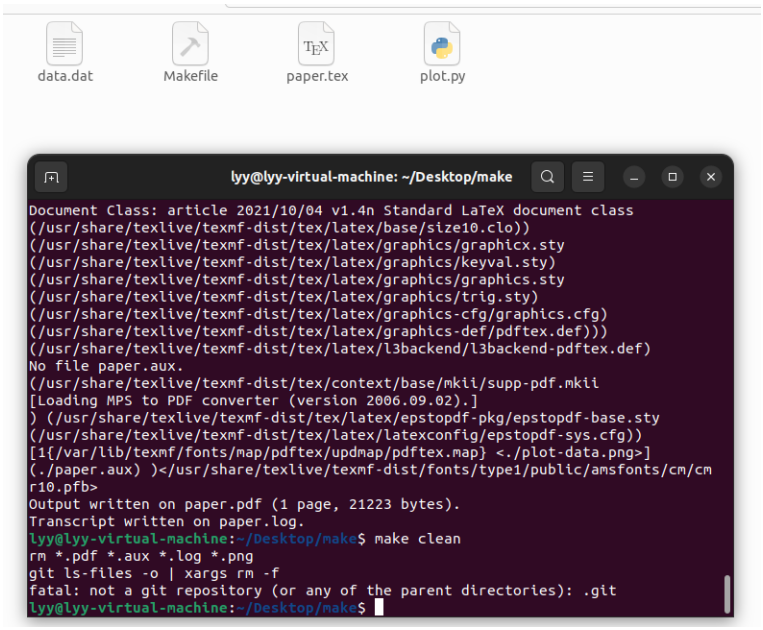

图 9: 删除生成文件

3 大杂烩

3.1 练习内容

3.1.1 修改键位映射

修改键位映射可以自定义键盘按键的功能，可以解决不适应默认键位的问题，从而提高工作效率。

3.1.2 守护进程

1、守护进程就是一系列在后台保持运行，不需要用户手动运行或者交互的进程。2、使用 systemctl status 命令看到正在运行的所有守护进程。

3.1.3 备份

1、复制存储在同一个磁盘上的数据不是备份，要将数据备份到不同的地点存储。

2、同步方案也不是备份。当数据在本地修改或损坏后，云端会同步这些更改，并且当我们无法登录这些应用时，数据将全部丢失。例如：我曾经把所有的电子笔记同步备份到百度网盘，但由于我更改手机号后，百度云盘无法登录，于是我所有的笔记都丢了。

3.1.4 API

API（应用程序接口）是软件应用之间通信的桥梁，它定义了两种软件应用之间是如何互相请求或交换数据的。开发人员可以使用 API 定义的方式与系统、服务或其他软件模块进行交互，而不需要了解其内部是如何实现的。

3.1.5 常见命令行标志参数及模式

了解了‘-’命令和‘–’命令的使用

3.1.6 Markdown 与 Github

1、Markdown 是一种轻量级的标记语言，介于 Word 和 Latex 之间，用于格式化文本。它使文本内容易于读写，同时可以生成结构化的文档。Github 中的 README 文件就是用 Markdown 编写的。

2、在 Github 中的 README 文件练习 Markdown 语法。

3、学会了在 Github 上如何与其他用户交互，了解了 issue、pull request、fork 等操作。

Listing 3: README 源代码

```
# lab4
 系统工具基础第四次实验

# 空行表示段落（1级标题）

 段落一

 段落二
## 二级标题
 ‘#’字符增加表示标题级数增加，所有符号后加空格才有用

*斜体*
**加粗**

- 项目
  - xiangmu

- 项目

1. 第一

2. 第二
    1. 子项


```

```
这行是代码：‘代码‘

‘‘‘代码块123456‘‘‘

[GitHub](https://github.com)
```

3.2 结果

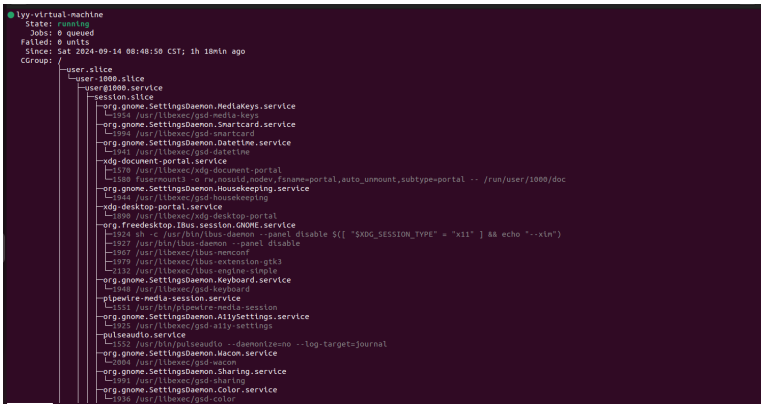

图 10: 正在运行的所有守护进程

图 11: - 命令的使用

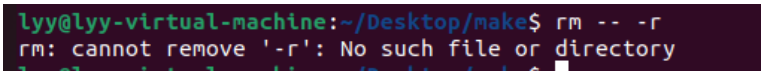

图 12: – 命令的使用

图 13: README 内容

4 pytorch

4.1 练习内容

4.1.1 Tensors

练习了 Tensors 的声明与定义以及对 Tensors 的操作。

Listing 4: tensors 的基本操作

```
from __future__ import print_function
import torch
tensor1 = torch.tensor([5.5, 3])
print('tensor1: ',tensor1)

tensor2 = tensor1.new_ones(5, 3, dtype=torch.double)
print('tensor2: ',tensor2)

tensor3 = torch.randn_like(tensor2, dtype=torch.float)
print('tensor3: ', tensor3)

tensor4 = tensor2 + tensor3
print('tensor2 + tensor3 :', tensor4)

x = torch.randn(4, 4)
y = x.view(16)
z = x.view(-1, 8)
print(x.size(), y.size(), z.size())
print(x)
```

```
print(y)
print(z)
```

4.1.2 和 Numpy 数组的转换

Tensor 和 Numpy 的数组可以相互转换，并且两者转换后共享在 CPU 下的内存空间。

4.1.3 autograd

1、tensor.requires-grad 属性：为 true 时会追踪该变量上的操作。

2、计算开始后形成的张量的.grad-fn 属性记录了形成该张量的操作，完成计算后调用.backward() 进行梯度的自动计算。

3、计算结果保存在张量的.grad 属性中。

Listing 5: 梯度计算

```
import torch
x = torch.rand(3, requires_grad=True)

y = x * 2
while y.data.norm() < 1000:
    y = y * 2

print("y:", y)
v = torch.tensor([0.1, 1.0, 0.0001], dtype=torch.float)
y.backward(v)

print("x.grad:", x.grad)    # d(y)/dx
print("x.requires_grad:", x.requires_grad)
print("(x ** 2).requires_grad:", (x ** 2).requires_grad)
with torch.no_grad():
    print("(x ** 2).with torch no grad:", (x ** 2).requires_grad)
```

4.1.4 神经网络

一个简单的卷积神经网络，包含了前向传播、损失计算、反向传播以及使用优化器来更新模型参数的过程。

Listing 6: 卷积神经网络的构建

```
import torch
import torch.nn as nn
import torch.nn.functional as F
#定义网络结构
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 输入图像是单通道，conv1 kenrnel size=5*5, 输出通道 6
        self.conv1 = nn.Conv2d(1, 6, 5)
        # conv2 kernel size=5*5, 输出通道 16
```

```
12          self.conv2 = nn.Conv2d(6, 16, 5)
13          # 全连接层
14          self.fc1 = nn.Linear(16*5*5, 120)
15          self.fc2 = nn.Linear(120, 84)
16          self.fc3 = nn.Linear(84, 10)
17
18      def forward(self, x):
19          # max-pooling 采用一个 (2,2) 的滑动窗口
20          x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
21          # 核(kernel)大小是方形的话，可仅定义一个数字，如 (2,2) 用 2 即可
22          x = F.max_pool2d(F.relu(self.conv2(x)), 2)
23          x = x.view(-1, self.num_flat_features(x))
24          x = F.relu(self.fc1(x))
25          x = F.relu(self.fc2(x))
26          x = self.fc3(x)
27          return x
28
29      def num_flat_features(self, x):
30          # 除了 batch 维度外的所有维度
31          size = x.size()[1:]
32          num_features = 1
33          for s in size:
34              num_features *= s
35          return num_features
36
37  net = Net()
38  print(net)
39  params = list(net.parameters())
40  print('The number of parameters: ', len(params))
41  # conv1.weight
42  print('The size of conv1.weight: ', params[0].size())
43  # 随机定义一个变量输入网络
44  input = torch.randn(1, 1, 32, 32)
45  out = net(input)
46  print(out)
47  # 清空所有参数的梯度缓存，然后计算随机梯度进行反向传播
48  net.zero_grad()
49  out.backward(torch.randn(1, 10))
50
51  # 定义损失函数
52  output = net(input)
53  # 定义伪标签
54  target = torch.randn(10)
55  # 调整大小，使得和 output 一样的 size
56  target = target.view(1, -1)
57  criterion = nn.MSELoss()
```

```
58
59 loss = criterion(output, target)
60 print(loss)
61 # MSELoss
62 print(loss.grad_fn)
63 # Linear layer
64 print(loss.grad_fn.next_functions[0][0])
65 # Relu
66 print(loss.grad_fn.next_functions[0][0].next_functions[0][0])
67
68 # 反向传播
69 # 清空所有参数的梯度缓存
70 net.zero_grad()
71 print('conv1.bias.grad before backward')
72 print(net.conv1.bias.grad)
73
74 loss.backward()
75
76 print('conv1.bias.grad after backward')
77 print(net.conv1.bias.grad)
78
79 # 更新权重
80 # 简单实现权重的更新例子
81 learning_rate = 0.01
82 for f in net.parameters():
83     f.data.sub_(f.grad.data * learning_rate)
84 import torch.optim as optim
85 # 创建优化器
86 optimizer = optim.SGD(net.parameters(), lr=0.01)
87
88 # 在训练过程中执行下列操作
89 optimizer.zero_grad() # 清空梯度缓存
90 output = net(input)
91 loss = criterion(output, target)
92 loss.backward()
93 # 更新权重
94 optimizer.step()
```

4.2 结果

```
                                > & C:/Users/hx/AppData/Local/Programs/Python/Python310/python.exe d:/课程/系统工具基础/Python_test/torch_test.py
tensor1:  tensor([5.5000, 3.0000])
tensor2:  tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
tensor2 + tensor3 : tensor([[ 1.0641,  0.0376,  1.9394],
        [ 1.6533,  2.0698,  2.4640],
        [ 1.5761,  1.2556, -0.4245],
        [ 1.7560,  0.7727,  1.1107],
        [-0.4882,  1.2196,  0.9627]], dtype=torch.float64)
torch.Size([4, 4]) torch.Size([16]) torch.Size([2, 8])
tensor([[[ 1.0520, -0.9231,  1.5167, -0.9840],
        [ 0.0289,  0.3537, -0.0985, -0.4206],
        [ 0.3815, -1.1562, -0.2960, -1.6003],
        [ 0.0398, -0.3108, -0.0707, -0.3784]]])
tensor([ 1.0520, -0.9231,  1.5167, -0.9840,  0.0289,  0.3537, -0.0985, -0.4206,
         0.3815, -1.1562, -0.2960, -1.6003,  0.0398, -0.3108, -0.0707, -0.3784])
tensor([[[ 1.0520, -0.9231,  1.5167, -0.9840,  0.0289,  0.3537, -0.0985, -0.4206],
         [ 0.3815, -1.1562, -0.2960, -1.6003,  0.0398, -0.3108, -0.0707, -0.3784]]])
```

图 14: tensors 的基本操作

```
24    a = torch.ones(5)
25    b = a.numpy()
26    print(a)
27    print(b)
28    a.add_(1)
29    print(a)
30    print(b)
31    a = np.ones(5)
32    b = torch.from_numpy(a)
33    print(a)
34    print(b)
35    np.add(a, 1, out=a)
36    print(a)
37    print(b)
38
```

```
问题    输出    调试控制台    终端    端口

PS D:\课程\系统工具基础\Python_test> & C:/Users/hx/AppData/Local/Programs/Pyth
tensor([1., 1., 1., 1., 1.])
[1. 1. 1. 1. 1.]
tensor([2., 2., 2., 2., 2.])
[2. 2. 2. 2. 2.]
[1. 1. 1. 1. 1.]
tensor([1., 1., 1., 1., 1.], dtype=torch.float64)
[2. 2. 2. 2. 2.]
tensor([2., 2., 2., 2., 2.], dtype=torch.float64)
PS D:\课程\系统工具基础\Python_test>
```

图 15: Numpy 和 Tensors 的数组转换

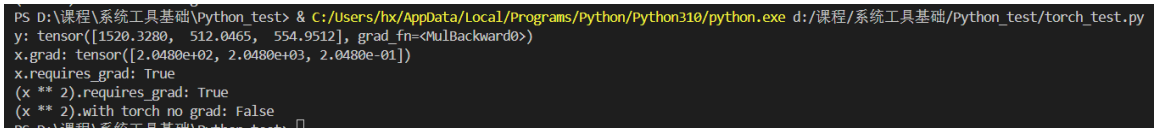

图 16: 梯度计算

5 解题感悟

在此次实验中，我接触到了很多新内容，这大大扩大了我的知识面，丰富了我的计算机基础知识。在学习过程中，不乏有很多地方是看不懂的，但是通过自主学习，逐步搜索查找，也学到了许多额外的知识，例如虚拟机的网络连接、numpy 数组、对电脑终端的操作等。随着了解的增多，我认为计算机真的是一个十分十分复杂的系统，需要学习的还有很多很多，

6 Github 链接

ttps://gitub.com/yuying019828/lab4