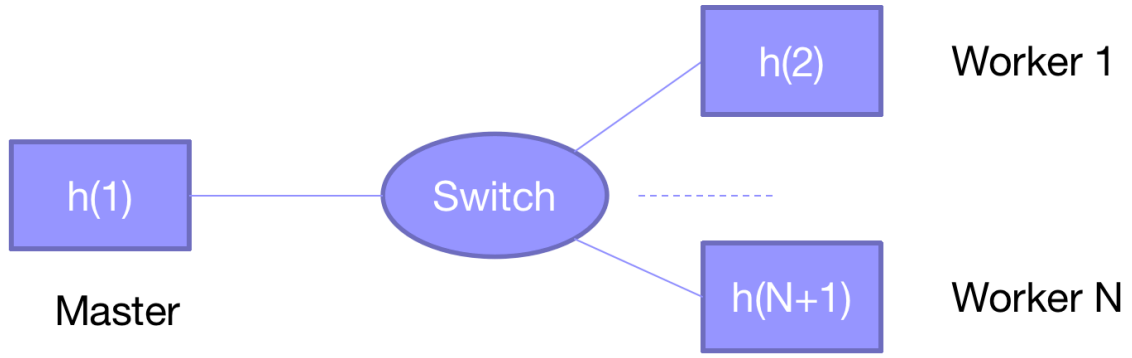
网络实验报告二

雷正宇 2016K8009909005

Socket应用编程实验

实验内容

使用 socket 网络编程 API 设计一个分布式字符统计程序：



本实验中，且取 N 为3，需要统计的文本是英文版小说《战争与和平》，位于本地磁盘。

Mininet 启动脚本 topo.py 已经给出，需要设计的部分只有 Master 和 Worker 的程序。其中 Master 是任务的分发者，具体的统计任务由 Workers 来做，最后 Master 收集数据并汇总打印。

实验流程

编写 Worker 的逻辑

本实验中，workers 的工作逻辑在 worker.py 文件中：

```
from socket import *
from threading import *
from json import *

def worker_begin():
    # server
    server = socket(AF_INET, SOCK_STREAM)
    host = '0.0.0.0'
    post = 12345
    server.bind((host, post))

    server.listen(1)
```

```
13
14      while True:
15          client, addr = server.accept()
16          print("Connection with: " + str(addr) + "is OK!")
17          msg = loads(client.recv(1024))
18          with open(msg['file_dir'], "r") as novel:
19              raw_string = novel.read()[msg['start'] : msg['end']].lower()
20
21          dic = {}
22          for char in range(ord('a'), ord('z') + 1):
23              dic[chr(char)] = raw_string.count(chr(char))
24
25          cooked_string = dumps(dic).encode()
26          print(cooked_string)
27          client.send(cooked_string)
28          client.close()
29
30
31  if __name__ == "__main__":
32      worker_begin()
```

worker 的工作逻辑比较简单：用 socket 构造函数申请一个 socket 对象，绑定，让它监听端口 12345。然后陷入工作循环，等待客户端的主动连接，一旦连接成功，接收客户端发来的信息。在本例中，为了使代码更加清晰，我使用 json 作为信息收发的格式。

收来的信息中包含 start（一个整数，代表字符统计的起始位置），end（一个整数，代表字符统计的终止位置），file_dir（一个字符串，代表需要处理的文件的相对位置）。随后根据 file_dir 打开文本文件，截取需要统计的部分，将从 'a' 到 'z' 26个字母的数量存放在一个字典中（文本已经被处理为全小写形式）。

最后将这个字典以 json 的格式发给客户端。

编写 Master 的逻辑

Master 在此任务中是任务的分发者，同时是客户端。它的代码在 master.py 中：

```
1   from socket import *
2   from threading import *
3   from json import *
4
5   novel_dir = "./war_and_peace.txt"
6   config_dir = "./workers.conf"
7   alphabet = {}
8   for char in range(ord('a'), ord('z') + 1):
9       alphabet[chr(char)] = 0
10
11  with open(novel_dir, "r") as f:
12      novel_len = len(f.read())
13
14  lock = Lock()
```

```
15
16
17  class Master(Thread):
18      def __init__(self, client_no, client_ip, amount):
19          Thread.__init__(self)
20          self.client_no = client_no
21          self.client_ip = client_ip
22          self.amount = amount
23
24
25      def run(self):
26          client = socket(AF_INET, SOCK_STREAM)
27          post = 12345
28          host = self.client_ip
29          client.connect((host, post))
30
31          start = int(novel_len * (self.client_no / self.amount))
32          end = int(novel_len * ((self.client_no + 1) / self.amount))
33
34          # print(start, end)
35          msg = dumps({
36              'start': start,
37              'end': end,
38              'file_dir': novel_dir
39          })
40          # print(msg.encode())
41          client.send(msg.encode())
42
43          ans = loads(client.recv(1024))
44          lock.acquire()
45          for letter in ans.keys():
46              alphabet[letter] += ans[letter]
47          lock.release()
48
49          client.close()
50
51
52  def master_begin():
53      # client
54      config_list = []
55      with open(config_dir, "r") as config:
56          for line in config:
57              config_list.append(line.replace('\n', ''))
58
59      n = len(config_list)
60
61      masters = []
62      try:
63          for i in range(n):
```

```
            masters.append(Master(i, config_list[i], n))
    except:
        print("Failed to create a master thread")

    for master in masters:
        master.start()

    for master in masters:
        master.join()

    # print(alphabet)
    for k, v in alphabet.items():
        print(k, " ", v)


if __name__ == "__main__":
    master_begin()
```

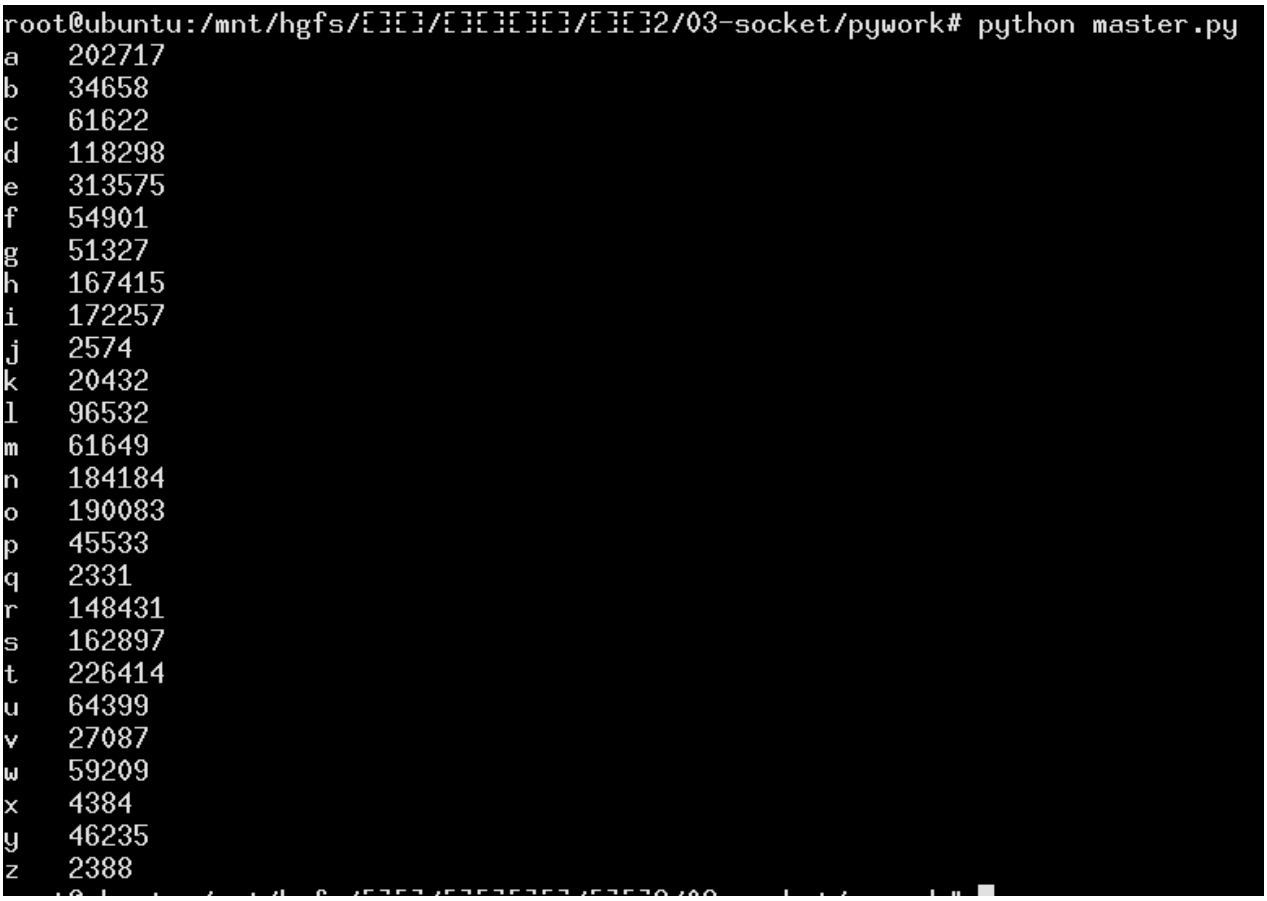
Master 的代码量稍微大一些，总共可分为两块。第一块 master_begin() 函数，它启动3个线程（线程数量由配置文件中 IP 信息的数量决定），并将线程序号和它们需要连接的服务器的 IP 信息作为参数传入。等到三个线程结束后，主线程打印用于统计字母数量的字典。

每个线程在初始化时得到了自己的序号、线程的数量和对应的 IP。作为客户端，线程设置好 IP 和端口信息后主动发起 TCP 连接，然后根据自己的序号和线程总数将待统计文本的长度等分为三等分，并把属于本线程要处理的部分的起始位置、终止位置和文件相对路径发送给服务器（Worker）。发送格式我采用了 json。

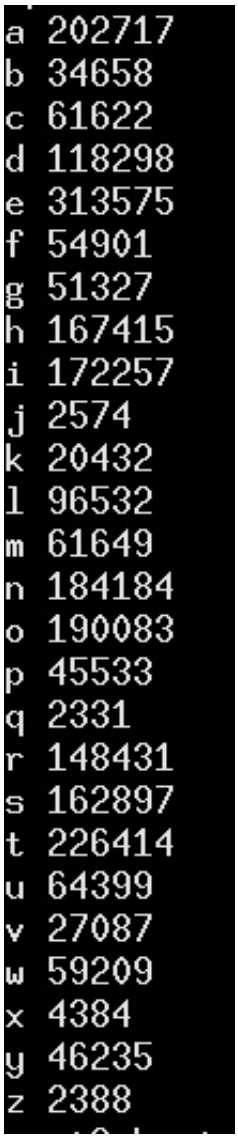
线程发送自己的信息后分别进入等待状态。等到服务器将局部统计信息返回，线程将对应的字母个数添加到 Master 的存放统计信息的字典中，最后一并交予主线程打印。在统计过程中，最好加上线程锁，避免数据漏加。

实验结果和分析

最后统计的结果如图：



和老师给的 reference 文件测试出来的结果完全一致：



说明各部分功能运行是正确的。

实验过程中的二三事

python版本

在实验中也遇到了不少问题，首先是 python 环境的切换。我在脚本中使用了少许 python3 风格的代码，导致 python2 运行它时发生错误，所以我想把我用于实验的虚拟机环境默认 python 版本切换为 python3.6. 然而在更换后 mininet 不能够运行了。

我意识到时 mininet 在安装时安装到了 python2 的库中，我不想再切回 python2 了，所以我打开 mininet 的目录，试图使用 easy install 工具再次安装 mininet 库。可是系统提示我找不到 setuptools 模块了。除了这些，其后还遇到一连串各种各样的问题，在这部分耗费的时间远多于实验本身，可折腾死我了。

曾记得，python2 和 python3 之间的版本切换已经让人很恼火。python3.7 刚发布时我就更新了，以至于后来我在使用 Tensorflow 做机器学习项目时不得不退回到 3.6，而我当时使用的工具是 anaconda。我刚刚接触这些工具，使用得并不熟练，导致我安装了不少次 python，而且磁盘里多了很多庞大无比的环境文件夹。

字符统计不精确

我并不是第一遍运行程序就得到了正确的结果。可以说，大部分代码正确完成了，但出现了一些小的纰漏，导致统计结果和正确结果之间出现了误差。

首先是我使用了 os.path.getsize() 函数来计算文件的大小。那时我没有意识到这个大小是文件字节数，和文件文本的字符个数并不相同。最后统计的结果比正确结果稍微大了一些。

json的使用

其实最初试图用C语言完成这个实验时是按照讲义，使用一串整形数字来收发信息的。但是，这种方法不仅很麻烦，需要自己定义信息格式的协议，而且扩展性很差（虽然这个程序只会在这个实验里用到），不如用 json。使用 json 对我来说有不少好处，python 的字典本来就对 json 有很好的支持，而且究其根本，使用 json 和纯整形数组相比，效率上不会差很多，只是多了几个引号、逗号而已，但处理起来要方便多了。

另一个好处是，纯字符串形式的数据在传输时不用考虑大小尾端的转换等问题。总之，json 是个好东西，Hash 是个讨人喜的容器。