socket 应用编程实验

熊子威

2015K8009915050

1 实验内容

本实验目标是构建一个基于 socket 的分布式字符统计程序并在 mininet 中运行这个程序。程序分为 master 和 worker 两个部分。master 通过 socket 接口向两个 worker 分发任务(需要统计字符数的文件路径,该文件中那个部分需要当前 worker 去统计),而两个 worker 则通过 socket 接口接受来自 master 的任务分配然后进行字母数目的统计,统计完毕后将二十六个字母的数目再各自发回给 master,然后 master 将两者的统计数据加和并打印出来。

2 实验流程

因为第一次使用 socket 并进行网络编程,对各个 API 的工作方式不是很了解,因此写代码时是一边测试一边迭代进行的。

首先编写一个简单的 master 和 worker, 确保 master 能够和一个 worker 连接上, 然后这个 worker 发送一条打招呼信息, 然后让 worker 将这条信息打印出来。

然后修改这个程序,添加多线程,让 master 同时和两个 worker 打招呼,确认 listen() 这个接口是否只需要主线程监听即可。结果是只需要主线程监听,两个工作线程负责 accept 然后处理数据传输和处理即可。

然后修改这个程序,让 master 按照顺序分别向两个 worker 发送两条消息,确认 socket 对消息发送和接受的顺序是如何处理的。然后发现 send 和 recv 是匹配在一起不会发生乱序之后,进一步修改程序。

这一步的修改是为 mastert 添加计算文件总长度并依据 worker 的不同地址而分别分配不同的任务并且让 worker 打印出分配来顺便检查工作分配是否正确(文件名是否正确接受,文本划分的起止位置是否正确)。

这一步确保能够正确工作之后为 worker 添加字符统计的代码,并分别打印结果来确认统计结果正确。

最后一步为 master 和 worker 分别添加接收统计结果和发送统计结果的代码,并为 master 添加汇总结果的代码。

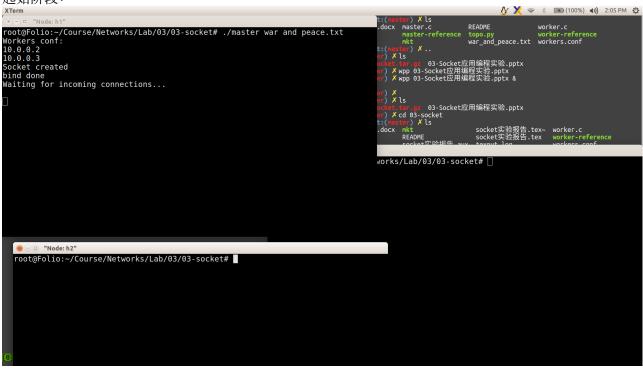
3 实验结果

2

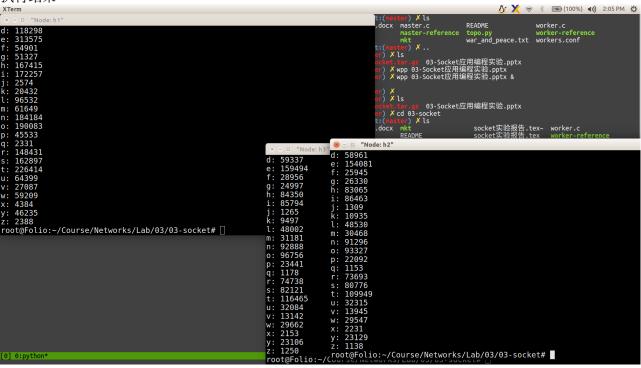
全部完成并却运转正确,和老师的 reference 输出一致之后,重新修改了 master 代码的结构,将一些复杂冗长的操作用函数封装,提高代码的可读性。

3 实验结果

起始阶段:



执行结束:



可以看到两个 worker 成功并且正确地向 master 发送了相关的数据, 而 master 也成功接受并且完成了

4 结果分析 3

最后的结果汇总。

4 结果分析

可以从 master 的主要代码来分析

```
173 void* handle(void* in) {
                                                                                                                 173 V
174
175
176 1
177
                                                                                                                             int cs;
struct sockaddr_in client;
                                                                                                                             accept_and_check(&cs, &client);
                                                                                                                 179
180
                                                                                                                             puts("dispatching...");
dispatch(cs, &client, (char *)in);
                                                                                                                 181
                                                                                                                 182
                                                                                                                             puts("comfirming...");
// receiving confirmation of start of counting
                                                                                                                 183
                                                                                                                 184
                                                                                                                             if(confirm(cs, &client))
    return NULL;
                                                                                                                 185
186
                                                                                                                 187
                                                                                                                             puts("receiving...");
// receiving counting result
if(receive(cs, &client) != 0)
                                                                                                                 188
                                                                                                                 189
35
                return -1;
                                                                                                                 190
                                                                                                                 191
192
193 }
194
                                                                                                                                   return NULL;
          if(get_conf(argv[1]) != 0) {
                                                                                                                             return in;
38
39
                return -1;
40
                                                                                                                 195 int main_work(char *in) {
196    pthread_t p1;
197    pthread_t p2;
198    void *status = NULL;
199    pthread_create(&p1, NUL)
          // setup a server
if(setup_server() != 0) {
   printf("Server setup failed\n");
   return -1;
43
44
                                                                                                                             pthread_create(&p1, NULL, handle, in);
pthread_create(&p2, NULL, handle, in);
45
                                                                                                                 200
201
46
47
                                                                                                                 202
203
204
205
                                                                                                                             pthread_join(p1, &status);
pthread_join(p2, &status);
          // main work
if(main_work(argv[1]) != 0)
48
50
                return -1;
                                                                                                                             if(status == NULL)
                                                                                                                 206
207
                                                                                                                                   return -1;
          show_results();
                                                                                                                             return 0;
          return 0;
                                                                                                                 208 }
```

master 进行了几个阶段的确认,建立 server 时,分配任务时,确认任务分发成功时,最后获得结果时,都需要确认成功才能进行下一步。相应的,worker 也需要进行响应。因此最后的结果如下图所示:

```
Wash (Backeto, Obex | Consect | Co
```