第三周实验报告

Socket 应用编程实验

2015K8009922021

李一苇

**一、实验内容**

实现一个基于socket的分布式字符统计程序，以熟悉BSD Socket API接口。

项目分两部分：master和worker

master负责：

* 读入workers.conf文件，分别建立与对应worker的连接
* 把传入的需统计的文件名，平均分配的统计任务传给各worker
* 将worker回传的统计信息整合后输出屏幕

worker负责：

* 根据传来的文件名和统计起止段，进行字符的统计（全部转为小写字符），将结果回传给master

**二、实验流程**

我分为以下步骤进行试验：

1. 构造能用的能收发消息的客户端和服务器端

这个已经由老师的example代码给出，只是需要明确，master作为主动提供工作请求的一方，是client的角色；worker作为被动处理工作并返回的一方，是server的角色。

1. 将收发消息的数据结构由char\*换成自己的结构体

为了设计的简单，这个结构体既负责master给worker的工作请求，包含待统计文件的文件名、统计起止段，也负责worker给master的工作结果，即26个字母的出现频次。

注意，所有上限均为int，对文件大小和路径名长其实有了隐性要求。

另外，如果视整个结构体为一个整体，则无需对里面的int变量进行htonl、ntohl的转化，本实验为了确保数据在网络传输中的可读性，仍对int变量进行了大小尾端的转化。

typedef struct message {

char path[ADDR\_MAX]; //master -> worker

int startpos; //master -> worker

int endpos; //master -> worker

int count[26]; //worker -> master

} message;

1. 具体的统计算法

在worker中实现， 代码非常简单，线性统计，将统计信息填回msg并发送即可。

pos = msg.startpos;

while (pos<msg.endpos) {

c = fgetc(work\_fp);

if (c>='A' && c<='Z') c = c - 'A' + 'a';

if (c>='a' && c<='z') msg.count[c - 'a']++;

pos++;

}

for (int i = 0; i < 26; i++) {

msg.count[i] = htonl(msg.count[i]);

}

write(clientsock, &msg, sizeof(msg));

1. 处理并发任务请求

课上介绍了两种处理并发任务请求的做法：多线程和IO复用。初做实验时不觉得会用上，因为本实验不涉及服务器对多客户端的问题；但写出来一运行，就发现问题：master是在串行等待各worker回传信息的，如果用上并发技术，可以在理论上将运行时间缩短为原来的nworkers分之一。

本实验选用IO复用，建立fd\_set的方式参考了网络上的资料。通过master一方以类似忙等方式的询问，可以快速得到所有worker的回传信息。

注意，每次需重新建立fd\_set以保证选择的空闲fd是最新的。

int complete\_worker\_num = 0;

while (complete\_worker\_num < work\_number) {

fd\_set rset;

int max\_sock = -1;

FD\_ZERO(&rset);

for (int i = 0; i < work\_number; i++) {

FD\_SET(sock[i], &rset);

if (sock[i] > max\_sock) max\_sock=sock[i];

}

if (select(max\_sock+1, &rset, NULL, NULL, NULL) > 0) {

for (int i = 0; i < work\_number; i++) {

if (FD\_ISSET(sock[i], &rset)) {

if (recv(sock[i], &msg[i], sizeof(message), 0) < 0)

return -1;

else

complete\_worker\_num++;

}

}

}

}

**三、实验结果和分析**

运行方法：（运行worker和master的终端取决于具体的workers.conf配置）

*进行编译*

$ make clean

$ make

*构造虚拟网络环境*

$ python topo.py

$ xterm h1 h2 h3 h4 （数目取决于目标网络环境中host数）

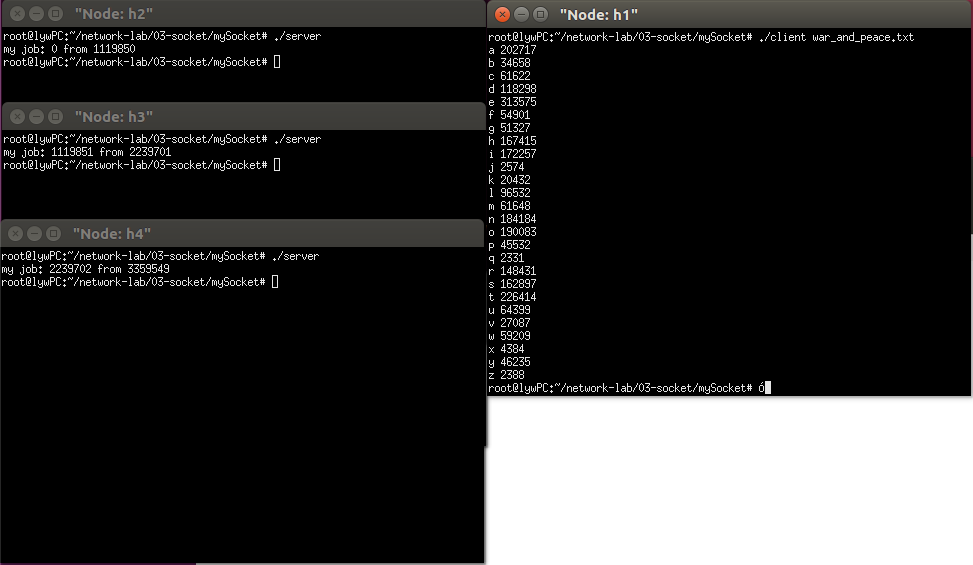
*运行worker，等待任务*

h2/h3/h4# ./server

*运行master，分配任务*

h1# ./client war\_and\_peace.txt

最终得到的结果



和reference进行比对，结果一致。

最后还试图打印等待结果阶段，每次轮询时compeleted worker数目，发现始终都循环了三次，每次加一。也没什么好展示的了。