

计算机网络研讨课实验报告

冯吕 2015K8009929049

2018 年 3 月 30 日

实验题目

Socket 应用编程实验

实验内容

在本次实验中，需要实现一个基于 *Socket* 的字符统计程序。

首先，有两个文件，*workers.conf* 配置文件中存储了所有 *worker* 的 *IP*，*war_and_peace.txt* 文件则是需要统计字符的文件。因此，在本次实验中，需要分别实现 *master* 和 *worker*，首先，*master* 需要读取 *worker* 的配置文件，即获得 *worker* 的 *IP*，然后与 *worker* 建立连接，连接成功之后，将任务分发给各个 *worker*，*worker* 接收到消息后，进行字符统计，然后把统计结果返回给 *master*，然后 *master* 整合统计结果并将结果输出到屏幕上。

实验流程

本次实验需要实现对应的 *master* 和 *worker*，对应的源程序为 *master.c* 和 *worker.c*，它们所实现的功能如下：

1. *master* 读取配置文件，获取 *IP*，*IP* 数为 2，因此，建立两个 *Socket*，分别与一个 *worker* 建立连接。

```
1 // Create socket
2 if ((s1 = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
3     perror("Could not create socket\n");
4     return -1;
5 }
6 if ((s2 = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
7     perror("Could not create socket\n");
8     return -1;
9 }
10 printf("Socket created\n");
11
12 // Read workers' conf
13 FILE *fp;
14 if ( !(fp = fopen("./workers.conf", "r")) ){
```

```

15     printf ( "Open config file failed!\n" );
16     return 1;
17 }
18 if (!(fgets(ip1, bufferSize, fp) && fgets(ip2, bufferSize, fp))){
19     printf ( "Read config file failed!\n" );
20     return 0;
21 }
22 fclose(fp);
23 struct sockaddr_in worker1, worker2;
24
25 worker1.sin_addr.s_addr = inet_addr(ip1);
26 worker1.sin_family = AF_INET;
27 worker1.sin_port = htons(8888);
28 worker2.sin_addr.s_addr = inet_addr(ip2);
29 worker2.sin_family = AF_INET;
30 worker2.sin_port = htons(8888);
31
32 // connect to worker
33 if (connect(s1, (struct sockaddr *)&worker1, sizeof(worker1))<0){
34     perror( "Connect worker1 failed!\n" );
35     return 1;
36 }
37 if (connect(s2, (struct sockaddr *)&worker2, sizeof(worker2))<0){
38     perror( "Connect worker2 failed!\n" );
39     return 1;
40 }
41 printf ("Connected all workers!\n");

```

2. 对于每个 *worker* 来说，首先创建一个 *Socket*，然后进行绑定，绑定完成之后，等待着 *master* 来连接。

```

1 sock = socket(AF_INET, SOCK_STREAM, 0);
2 if (sock == -1) {
3     printf("Could not create socket\n");
4 }
5 printf("Socket created\n");
6
7 // Prepare the sockaddr_in structure
8 worker.sin_family = AF_INET;
9 worker.sin_addr.s_addr = INADDR_ANY;
10 worker.sin_port = htons(8888);
11
12 if ( bind(sock, (struct sockaddr *)&worker, sizeof(worker)) <0 ){
13     perror( "bind failed. Error\n" );
14     return -1;

```

```

15 }
16
17 //waiting for master to connect
18 listen(sock, 3);
19 printf ( "Waiting for incoming connectionos...\n" );

```

3. *master* 与两个 *worker* 建立连接之后，需要分发任务。首先，通过文件操作获取文件中的总字符数目，然后将前一半分给第一个 *worker* 统计，后一半分给另一个。因此，发送给 *worker* 的消息总共占 30 个字节，四个字节为字节数，四个字节为 *worker* 需要统计的起始位置，四个字节为终止位置，剩余的十八个字节为文件名（包括字符串中的最后一个空字符）。

```

1 // open count file
2 int numOfChar = 0;
3 FILE *fpc;
4 if ( !(fpc = fopen("./war_and_peace.txt", "r")) ){
5     printf ( "Open war_and_peace.txt failed!\n" );
6     return 1;
7 }
8 //count all char nums
9 while( (fgetc(fpc)) != EOF ){
10     ++numOfChar;
11 }
12 fclose(fpc);
13 int middle = numOfChar / 2;
14 int total = 30;
15 int zero = 0;
16
17 char loc [] = { 'w', 'a', 'r', '_', 'a', 'n', 'd',
18     '_', 'p', 'e', 'a', 'c', 'e', '.', 't', 'x', 't', '\0' };
19 void *message1 = malloc (30);
20 void *message2 = malloc (30);
21 memcpy((void *)message1, &total, 4);
22 memcpy((void *)((int *)message1 + 1), &zero, 4);
23 memcpy((void *)((int *)message1 + 2), &middle, 4);
24 strncpy(((char *)message1 + 12), loc, 18);
25 memcpy((void *)message2, &total, 4);
26 memcpy((void *)((int *)message2 + 1), &middle, 4);
27 memcpy((void *)((int *)message2 + 2), &numOfChar, 4);
28 strncpy(((char *)message2 + 12), loc, 18);
29
30 //send work to workers
31 if ( send(s1, message1, 30, 0) < 0 ){
32     printf ( "Send to worker1 failed!\n" );
33     return 1;
34 }

```

```

35 if ( send(s2, message2, 30, 0) < 0 ){
36     printf ( "Send to worker2 failed!\n" );
37     return 1;
38 }

```

4. 每个 *worker* 接收到 *master* 传过来的消息之后，就获取到了文件名，然后打开文件，通过 *fseek* 函数定位到起始位置，进行字符统计，统计所有的字母，不区分大小写。统计完成之后，再将统计结果发送回去给 *master*，每个字母的数目用一个 *int* 型四字节存储，因此，返回给 *master* 的消息长度为 104 字节。

```

1  int msg_len = 0;
2  msg_len = recv(cs, msg, 100, 0);
3  if (msg_len <= 0){
4      printf ("recv message error.\n");
5      return 1;
6  }
7  printf ("Recv successful.\n");
8
9  int begin = *((int *)((int *)msg + 1));
10 int end = *((int *)((int *)msg + 2));
11 strncpy(name, ((char *) (msg) + 12), 18);
12 /* count the number of a~z
13  */
14 FILE *fp;
15 if ( !(fp = fopen((char *)name, "r")) ){
16     printf ( "Open file failed.\n" );
17     return 1;
18 }
19
20 fseek (fp, begin, SEEK_SET);
21
22 for ( int i = 0; i < 26; ++i ){
23     ((int *)send_msg)[i] = 0;
24 }
25
26 char ac;
27 for ( int i = 0; i < end - begin && (ac = fgetc(fp)) != EOF; i++ ){
28     if ( ac >= 'a' && ac <= 'z' ){
29         ++((int *)send_msg)[ac - 'a'];
30     }
31     if ( ac >= 'A' && ac <= 'Z' ){
32         ++((int *)send_msg)[ac - 'A'];
33     }
34 }
35

```

```
36 write (cs, send_msg, 104);
```

5. *master* 收到两个 *worker* 返回的消息之后，将两个 *worker* 对应的统计值相加，然后在屏幕上输出 26 个字母的数目统计值。

```
1  if (recv(s1, recv_msg1, 104, 0) < 0){
2      printf ("recv failed!\n");
3      return 1;
4  }
5  if (recv(s2, recv_msg2, 104, 0) < 0){
6      printf ("recv failed!\n");
7      return 1;
8  }
9
10 /* Print the count result*/
11 for (int i = 0; i < 26; ++i){
12     printf ("%c %d\n", 'a' + i,
13         ((int *)recv_msg1)[i] + ((int *)recv_msg2)[i]);
14 }
```

实验结果

运行截图如下，*master* 与 *worker* 成功建立了连接，最终输出了正确统计结果。

```

Node: h1
d 118298
e 313575
f 54901
g 51327
h 167415
i 172257
j 2574
k 20432
l 96532
m 61649
n 184184
o 190083
p 45533
q 2331
r 148431
s 162897
t 226444
u 64399
v 27087
w 59209
x 4384
y 46235
z 2388
12:08 root@segmentfault:03-socket $ 
12:08 fenglv@segmentfault:03-socket $ make clean
12:08 fenglv@segmentfault:03-socket $ make
gcc -Wall -g master.c -o master
gcc -Wall -g worker.c -o worker
12:08 fenglv@segmentfault:03-socket $ sudo ./topo.py
mininet> xterm h1 h2 h3
mininet> 

Node: h3
12:08 root@segmentfault:03-socket $ ./worker
Socket created
Waiting for incoming connections...
Connection accepted.
Recv successful.
12:08 root@segmentfault:03-socket $ 

Node: h2
12:08 root@segmentfault:03-socket $ ./worker
Socket created
Waiting for incoming connections...
Connection accepted.
Recv successful.
12:08 root@segmentfault:03-socket $ 

```

图 1: *Socket* 应用编程实验运行截图

结果分析

在刚开始，由于一些内存错误，导致 *worker* 运行时出现 *segmentation fault*，后经过调试，程序能够按照实验要求正确运行，输出结果与 *reference* 程序一样。