

Socket 应用编程实验

学号：2016K8009908007

姓名：薛峰

一、实验内容

1、Master 分发任务

(1) Master 通过读取 workers.conf 配置文件，获取每个 worker 的 IP 地址，然后分别建立连接；

(2) Master 获取 war_and_peace.txt 文件长度，将统计任务等分到所有的 worker；

(3) Master 给每个 worker 发送消息，包括如下内容：

- 1) 文件名长度（4 个字节）
- 2) 文件所在位置（因为 master 和 worker 在同一主机同一目录，所以给出相对位置即可）
- 3) 需要进行字符统计的起始位置（4 个字节）和终止位置（4 个字节）

2、Worker 计算并返回结果

(1) 每个 worker 收到消息后，进行解析，根据指定统计区间对文件进行统计；

(2) Worker 统计结束后，将每个字符出现的次数以 4 字节整数形式（网络字节序）返回给 Master，因此传输消息长度为 104 字节；

(3) Master 收到所有 worker 的消息后，进行聚合并输出到屏幕。

二、实验流程

1、Master

(1) 读 workers.conf 文件。本次实验读取了 3 个 ip。

```
// read ip
num_worker = 0;
FILE * conf_file;
if( (conf_file = fopen("workers.conf", "r"))==NULL )
{
    printf("Open workers.conf failed!\n");
    exit(1);
}

char str[MAX_WORKER * MAX_IP_SIZE];
fread(str, MAX_WORKER * MAX_IP_SIZE, 1, conf_file);

for(i = 0, k = 0; i < MAX_WORKER; i++)
{
    if( (str[k] != 10 || str[k] != '.') && (str[k] < '0' || str[k] > '9'))
        break;
    else
        for(j = 0; j < MAX_IP_SIZE; j++, k++)
        {
            if(str[k] == 10)
            {
                ip[i][j] = 0;
                k++;
                break;
            }
            else
                ip[i][j] = str[k];
        }
}

num_worker = i;
printf("Reading workers.conf succeeds, nworkers = %d!\n", num_worker);
```

(2) 创建 socket 文件描述符

```
// create socket
for(i = 0; i < num_worker; i++)
    if( (s[i] = socket(AF_INET, SOCK_STREAM, 0)) < 0 )
    {
        perror("Create socket failed");
        return -1;
    }

printf("Socket created!\n");
```

(3) 连接 workers。

```
// connect to workers
for(i = 0; i < num_worker; i++)
{
    worker[i].sin_addr.s_addr = inet_addr(ip[i]);
    worker[i].sin_family = AF_INET;
    worker[i].sin_port = htons(12345);

    if (connect(s[i], (struct sockaddr *)&worker[i], sizeof(worker[i])) < 0)
    {
        perror("connect failed");
        return 1;
    }
}

printf("Connected!\n");
```

(4) 统计 war_and_peace.txt 总字数。获取总字数的目的是为了能够将统计任务平均分为 nworkers 分，从而为每个 worker 分配一个任务。

```
// count the total num
FILE *txt;
int total_num = 0;

if ( !(txt = fopen(argv[1], "r")) )
{
    printf ( "Open war_and_peace.txt failed!\n" );
    return 1;
}

while( (fgetc(txt)) != EOF )
    total_num++;

fclose(txt);
```

(5) 创建子进程。对于每个 worker 都创建一个子进程，用于和对应的 worker 进行通讯。

```
// submit request
int thread_id[MAX_WORKER];
for(i = 0; i < MAX_WORKER; i++)
    thread_id[i] = i;

for(i = 0; i < num_worker; i++)
{
    send_msg[i].filename_length = (int) strlen(argv[1]);

    memset(send_msg[i].locate, 0, sizeof(send_msg[i].locate));
    strcat(send_msg[i].locate, "./");
    strcat(send_msg[i].locate, argv[1]);

    send_msg[i].start = (i == 0)? 0 : total_num/3 * i + 1;
    send_msg[i].end = (i == num_worker - 1)? total_num : total_num/3 * (i + 1);

    int *t = thread_id + i;
    if( pthread_create(&thread[i], NULL, submit_request, (void*) t) != 0)
    {
        printf("Can't create thread[%d]\n", i);
        return 1;
    }
}

for(i = 0; i < num_worker; i++)
    pthread_join(thread[i], NULL);

printf("Send & Recv succeed!\n");
```

(6) 发送和接收数据函数。

```
void* submit_request(void* request)
{
    int i = * (int*)request;
    printf("Now, thread[%d] starts to work!\n", i);

    // send message
    if (send(s[i], (char*) &send_msg[i], sizeof(send_msg[i]), 0) < 0)
        printf("send failed");

    // recv message
    if (recv(s[i], (char*) recv_msg[i], sizeof(recv_msg[i]), 0) < 0)
        printf("recv failed!\n");
}
```

```
typedef struct send_message{
    int filename_length;
    char locate[32];
    int start;
    int end;
} send_message;
```

发送的数据包如上图所示，其中 filename_length 为文件名长度，locate[32]为文件所在位置，start 为需要统计的起始位置，end 统计的末位位置。

(7) 打印并关闭 socket 文件描述符

```
// print result
for(i = 0; i < 26; ++i)
{
    int count = 0;

    for(j = 0; j < num_worker; j++)
        count += recv_msg[j][i];

    printf ("%c: %d\n", 'a' + i, count);
}

for(i = 0; i < num_worker; i++)
    close(s[i]);
```

2、Worker

(1) 创建 socket 文件描述符

```
// create socket
if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("create socket failed");
    return -1;
}
printf("socket created\n");
```

(2) 将 socket 文件描述符与监听地址绑定

```
// prepare the sockaddr_in structure
worker.sin_family = AF_INET;
worker.sin_addr.s_addr = INADDR_ANY;
worker.sin_port = htons(12345);

// bind
if (bind(s, (struct sockaddr *)&worker, sizeof(worker)) < 0)
{
    perror("bind failed");
    return -1;
}
printf("bind done\n");
```

(3) 监听

```
// listen
listen(s, 3);
printf ( "Waiting for incoming connectionos...\n" );
```

(4) 接受连接请求

```
// accept connection from an incoming client
int c = sizeof(struct sockaddr_in);
if ((cs = accept(s, (struct sockaddr *)&master, (socklen_t *)&c)) < 0)
{
    perror("accept failed");
    return -1;
}
printf("connection accepted\n");
```

(5) 接收数据，统计，并返回结果

```
// receive a message from master
int msg_len = 0;
send_message send_msg;

while ((msg_len = recv(cs, (char*) &send_msg, sizeof(send_msg), 0)) > 0)
{
    int i;
    char a;
    int count[26];
    FILE* file = fopen(send_msg.locate, "r");

    for(i = 0; i < 26; i++)
        count[i] = 0;

    fseek (file, send_msg.start, SEEK_SET);

    for (i = 0; i <= send_msg.end - send_msg.start && (a = fgetc(file)) != EOF; i++)
    {
        if ( a >= 'a' && a <= 'z' )
            count[a - 'a']++;
        if ( a >= 'A' && a <= 'Z' )
            count[a - 'A']++;
    }

    write(cs, (char*) count, 104);
}
```

3、运行过程

- \$ gcc -o master master.c -lpthread
- \$ gcc -o worker worker.c
- \$ sudo python topo.py
- mininet> xterm h1 h2 h3 h4
- h4 # ./worker
- h3 # ./worker
- h2 # ./worker
- h1 # ./master war_and_peace.txt

三、实验结果及分析

1、实验结果

```
"Node: h1"
root@feng-VirtualBox:~/2016K8009908007-薛峰-03-Socket/用/程/Code# ./master_war_and_peace.txt
Reading workers.conf succeeds, nworkers = 3!
Socket created!
Connected!
Now, thread[2] starts to work!
Now, thread[1] starts to work!
Now, thread[0] starts to work!
Send & Recv succeed!
a: 202717
b: 34658
c: 61622
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: 226414
u: 64399
v: 27087
w: 59209
x: 4384
y: 46235
z: 2388
root@feng-VirtualBox:~/2016K8009908007-薛峰-03-Socket/用/程/Code#

"Node: h2"
root@feng-VirtualBox:~/2016K8009908007-薛峰-03-Socket/用/程/Code# ./worker
Socket created!
Bind done!
Waiting for incoming connections...
Connection accepted!
Client disconnected!
root@feng-VirtualBox:~/2016K8009908007-薛峰-03-Socket/用/程/Code#

"Node: h3"
root@feng-VirtualBox:~/2016K8009908007-薛峰-03-Socket/用/程/Code# ./worker
Socket created!
Bind done!
Waiting for incoming connections...
Connection accepted!
Client disconnected!
root@feng-VirtualBox:~/2016K8009908007-薛峰-03-Socket/用/程/Code#

"Node: h4"
root@feng-VirtualBox:~/2016K8009908007-薛峰-03-Socket/用/程/Code# ./worker
Socket created!
Bind done!
Waiting for incoming connections...
Connection accepted!
Client disconnected!
root@feng-VirtualBox:~/2016K8009908007-薛峰-03-Socket/用/程/Code#
```

2、结果分析

与标准结果对比，实验结果正确。

四、实验总结

在实验过程中，遇到了一些 bug，其中一个是在发现 master 发出去的数据全部由 worker1 接收，worker2 和 worker3 没有接收到数据。通过调试发现是在建立连接的时候，三次都与 worker1 建立连接，而没有与 worker2 和 worker3 建立连接，从而导致 worker2 和 worker3 收不到数据。

另一个是在创建多个进程的时候，传给子进程的是参数&i，但结果发现每个子进程得到的 i 都是 0。通过查找资料发现，因为每个子进程的参数都是 i 的指针，并且 main 函数的 for 循环执行的快，所以每个子进程的指针指向的值都是一样的。因此改变每个子进程传入的指针即可修正该错误。

通过本次实验，我基本掌握了 Socket 应用编程的基础，理解了发送数据和接收数据的过程，并且重新复习了之前关于 pthread_create 函数的相关知识。