Telnet协议的客户端/服务端程序设计

71118415 叶宏庭

东南大学软件学院

Email: 213182964@seu.edu.cn

May 16, 2021

1 实验目的

了解Telnet协议，掌握Telnet协议的客户端/服务端程序设计。

2 实验环境

2.1 操作系统:

Ubuntu 20.04

2.2 辅助软件:

CTEX(用于编写tex报告)

3 实验内容

3.1 了解Telnet协议:

Telnet协议是TCP/IP协议族中的一员，是Internet远程登录服务的标准协议和主要方式。它为用户提供了在本地计算机上完成远程主机工作的能力。在终端使用者的电脑上使用telnet程序，用它连接到服务器。终端使用者可以在telnet程序中输入命令，这些命令会在服务器上运行，就像直接在服务器的控制台上输入一样。

本实验目的就是实现一个Telnet协议程序，能够远程操作服务器。

3.2 编写程序：

3.2.1 客户端程序client.c

读取终端命令输入、发送至服务端

```
void send_cmd(int sock, int pid) {
        char str[MAX_MSG_LENGTH] = {0};
        printf("> ");
        while (fgets(str, MAX_MSG_LENGTH, stdin) == str) {
                if(strncmp(str, END_STRING, strlen(END_STRING)) == 0) break;
                if(send(sock, str, strlen(str)+1, 0) < 0) perro("send");
        }
        kill(pid, SIGKILL);
        printf("Goodbye.\n");
}
```

接受服务端回馈、打印结果

```
void receive(int sock) {
        char buf[MAX_MSG_LENGTH] = {0};
        int filled = 0;
        while(filled = recv(sock, buf, MAX_MSG_LENGTH-1, 0)) {
                buf[filled] = '\0';
                printf("%s", buf);
                fflush(stdout);
        }
        printf("Server disconnected.\n");
}
```

主函数、定义套接字

```
struct sockaddr_in connection;
connection.sin_family = AF_INET;
memcpy(&connection.sin_addr, &server_addr, sizeof(server_addr));
connection.sin_port = htons(PORT);
```

开创子进程进行通信

```
int pid;
if(pid = fork()) send_cmd(sock, pid);
else receive(sock);
```

3.2.2 服务端程序server.c

配置套接字，地址、端口、绑定、监听

```
name.sin_family = AF_INET;
name.sin_addr.s_addr = INADDR_ANY;
name.sin_port = htons(PORT);
if(bind(sock, (void*) &name, sizeof(name))) perro("binding tcp socket");
if(listen(sock, 1) == -1) perro("listen");
```

程序主体，接受命令，执行命令

```
D("Initializing server...\n");
while(new_socket = accept(sock, &cli_addr, &cli_len)) {
        D("Client connected.\nForking... ");
        if(pid = fork()) D("child pid = %d.\n", pid);
        else {
                pid = getpid();
                if(new_socket < 0) perro("accept");
                if(dup2(new_socket, STDOUT_FILENO) == -1) perro("dup2");
                if(dup2(new_socket, STDERR_FILENO) == -1) perro("dup2");
                /* Processing part.*/
                close(new_socket);
                D("\t[%d] Dying.", pid);
                exit(0);
        }
}
```

本部分代码为建立连接，创建子进程处理请求。后续再Processing part块中加入请求处理代码即可。

请求处理部分代码

```
while(1) {
        int readc = 0, filled = 0;
        while(1) {
                readc = recv(new_socket, buf+filled, MAX_MSG_LENGTH-filled-1, 0);
                if(!readc) break;
                filled += readc;
                if(buf[filled-1] == '\0') break;
        }
        if(!readc) {
                D("\t[%d] Client disconnected.\n", pid);
                break;
        }
        D("\t[%d] Command received: %s", pid, buf);
        system(buf);
        D("\t[%d] Finished executing command.\n", pid);
        send(new_socket, "> ", 3, MSG_NOSIGNAL);
}
```

在recv中接收来自客户端的命令字符，接收完毕后在system()中进行调用，最后将结果通过send()进行返回。

4 实验结果与分析

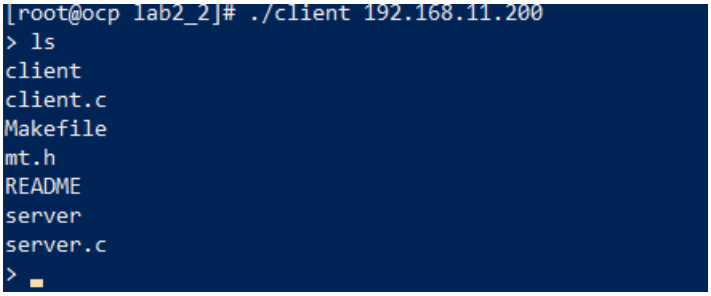
4.1 完整的C/S结构程序:

基于Telnet的C/S结构程序。（详细代码请见附带code文件夹）

4.2 运行结果分析:

4.2.1 客户端终端:

客户端1:



客户端2:

```
[root@ocp lab2_2]# ./client 192.168.11.200
> ls
client
client.c
Makefile
mt.h
README
server
server.c
>
```

从结果图中可以看出，首先本程序的服务器端支持并发，能够同时接受多个客户端的请求。其次，我们执行了ls命令，服务端也争取返回了命令执行的结果，因此该C/S结构程序是有效的。

4.2.2 服务端终端:

服务端：

```
[root@ocp lab2_2]# ./server
Initializing server...
Client connected.
Forking... child pid = 9362.
Client connected.
Forking... child pid = 9367.
        [9367] Command received: ls
        [9367] Finished executing command.
        [9362] Command received: ls
        [9362] Finished executing command.
```

服务端首先初始化服务器，即配置TCP/IP服务，完成初始化后即可开始接受请求，每来一个客户端请求，都会开辟一个新的子进程来处理请求，同时采用不同的端口分别和客户端进行通信，并且显示进程与具体的执行命令，可以记录log日志。