**暨南大学本科实验报告专用纸**

课程名称 计算机网络实验 成绩评定

实验项目名称 基于TCP的远程Bash命令执行协议 指导教师

学生姓名 冯志平 学号 2014051742

学院信息科学技术 系计算机科学 专业 计算机科学与技术

1. **实验目的与背景**

**目的：**

实现一个基于TCP的远程Bash命令执行协议（cmdrunner），并且客户端可以跨平台。

**背景：**

在Linux当中，我们经常需要用到一个Windows转换层WINE来模拟一个子系统，它能解释并运行PE结构的Windows二进制可执行文件。为了能更加与底层系统更好地配合，也经常需要在WINE当中执行一些Linux的Bash命令来开启某些东西，比如说执行Linux应用程序。

有时我不在电脑前面，却想远程执行某些程序的时候，与其下载一个庞大的SSH，或者Telnet（很多Windows系统并不默认提供telnet，而是要另外开启），不如用一个功能稍微逊色，但是微型、可跨平台的客户端。

1. **实验内容**

协议内容：[4字节的命令长度N][N字节的命令]

实现一个客户端，支持设置将连接的服务器的IP和端口，并且支持对命令的转义，以致远程命令的执行正常，命令格式：

client [--host <ip> --port <port> --quote] [<command> ...]

实现一个服务器端，支持设置绑定的端口和IP，并且在每收到一条命令的时候，打出日志。命令格式：

server [--host <ip> --port <port>]

1. **程序代码**

代码1. 客户端代码 client.c

|  |
| --- |
| /\*  \* Copyright: Shihira Fung <fengzhiping@hotmail.com>  \*/  #if defined(\_\_linux\_\_)  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <poll.h>  #elif defined(\_WIN32)  #include <winsock.h>  #endif  #include <stdint.h>  #include <stddef.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <errno.h>  void assert\_error(int err)  {  #if defined(\_\_linux\_\_)  if(err < 0) {  fprintf(stderr, "[OS Error %d] ", errno);  perror("");  abort();  }  #elif defined(\_WIN32)  if(err < 0) {  DWORD dw = GetLastError();  fprintf(stderr, "[OS Error %d]\n", dw);  abort();  }  #endif  }  int main(int argc, char\* argv[])  {  #if defined(\_WIN32)  WSADATA wsaData;  WSAStartup(0x0202, &wsaData);  #endif  int client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);  assert\_error(client\_socket);  int command = 0;  int quote = 0;  struct sockaddr\_in addr;  addr.sin\_family = AF\_INET;  addr.sin\_port = htons(16600);  addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");  for(int i = 1; i < argc; i++) {  if(!strcmp(argv[i], "--host") && i + 1 < argc) {  addr.sin\_addr.s\_addr = inet\_addr(argv[i + 1]);  i += 1;  }  else if(!strcmp(argv[i], "--port") && i + 1 < argc) {  addr.sin\_port = htons(atoi(argv[i + 1]));  i += 1;  } else if(!strcmp(argv[i], "--quote")) {  quote = 1;  } else {  command = i;  break;  }  }  if(!command) {  printf("USAGE:\n %s [--host <ip>] "  "[--port <port>] <command>\n", argv[0]);  exit(255);  }  assert\_error(connect(client\_socket,  (struct sockaddr\*)&addr, sizeof(addr)));  // concat arguments  uint32\_t size = 0;  for(int i = command; i < argc; i++)  size += strlen(argv[i]) + 1;  char\* buf = (char\*) malloc(size \* 2);  char\* pstr = buf;  for(int i = command; i < argc; i++) {  char\* cur\_arg = argv[i];  if(quote)  \*(pstr++) = '\'';  for(; \*cur\_arg != '\0'; cur\_arg++) {  if(quote && \*cur\_arg == '\'') {  pstr[0] = '\'';  pstr[1] = '\"';  pstr[2] = '\'';  pstr[3] = '\"';  pstr[4] = '\'';  pstr += 5;  } else  \*(pstr++) = \*cur\_arg;  }  if(quote)  \*(pstr++) = '\'';  \*(pstr++) = i < argc - 1 ? ' ' : '\0';  }  size = pstr - buf;  assert\_error(send(client\_socket, (char\*) &size, sizeof(size), 0));  assert\_error(send(client\_socket, buf, size, 0));  free(buf);  close(client\_socket);  #if defined(\_WIN32)  WSACleanup();  #endif  } |

代码2. 服务器代码 server.c

|  |
| --- |
| /\*  \* Copyright: Shihira Fung <fengzhiping@hotmail.com>  \*/  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <poll.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <errno.h>  #define MAX\_CONNECTIONS 10  void assert\_error(int e)  {  if(e < 0) {  fprintf(stderr, "[OS Error %d] ", errno);  perror("");  abort();  }  }  struct recv\_stream {  size\_t full\_size;  size\_t recv\_size;  char\* buf;  };  struct conn\_info {  struct pollfd\* fd;  struct recv\_stream stream;  struct sockaddr\_in addr;  };  size\_t ncl = 0;  struct pollfd fds[1 + MAX\_CONNECTIONS];  struct conn\_info clients[MAX\_CONNECTIONS];  struct pollfd\* cfds = fds + 1;  //struct recv\_stream client\_streams[1 + MAX\_CONNECTIONS];  int new\_client(int sock, const struct sockaddr\_in\* client\_addr)  {  clients[ncl].stream.buf = NULL;  clients[ncl].stream.recv\_size = 0;  clients[ncl].stream.full\_size = 0;  clients[ncl].fd = &(cfds[ncl]);  clients[ncl].fd->fd = sock;  clients[ncl].fd->events = POLLIN;  clients[ncl].fd->revents = 0;  memcpy(&(clients[ncl].addr), client\_addr, sizeof(struct sockaddr\_in));  ncl += 1;  return ncl - 1;  }  int read\_client(int i)  {  struct recv\_stream\* cur = &clients[i].stream;  if(!cur->full\_size) {  uint32\_t size;  assert\_error(recv(clients[i].fd->fd, &size, sizeof(size), 0));  cur->full\_size = size;  cur->recv\_size = 0;  cur->buf = (char\*) malloc(size + 1);  cur->buf[size] = 0;  } else {  ssize\_t recv\_size = recv(clients[i].fd->fd,  cur->buf + cur->recv\_size,  cur->full\_size - cur->recv\_size, 0);  assert\_error(recv\_size);  cur->recv\_size += recv\_size;  }  return cur->full\_size - cur->recv\_size;  }  void copy\_cilent(int dest, int src)  {  memcpy(cfds + dest, cfds + src, sizeof(struct pollfd));  memcpy(clients + dest, clients + src, sizeof(struct conn\_info));  clients[dest].fd = cfds + dest;  }  void remove\_client(int idx)  {  if(idx != ncl - 1)  copy\_cilent(idx, ncl - 1);  free(clients[idx].stream.buf);  ncl -= 1;  }  int main(int argc, char\* argv[])  {  int server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);  assert\_error(server\_socket);  struct sockaddr\_in addr;  addr.sin\_family = AF\_INET;  addr.sin\_port = htons(16600);  addr.sin\_addr.s\_addr = inet\_addr("0.0.0.0");  for(int i = 1; i < argc; i++) {  if(!strcmp(argv[i], "--host") && i + 1 < argc) {  addr.sin\_addr.s\_addr = inet\_addr(argv[i + 1]);  i += 1;  }  else if(!strcmp(argv[i], "--port") && i + 1 < argc) {  addr.sin\_port = htons(atoi(argv[i + 1]));  i += 1;  }  }  assert\_error(bind(server\_socket, (struct sockaddr\*)&addr, sizeof(addr)));  assert\_error(listen(server\_socket, 0));  fds->fd = server\_socket;  fds->events = POLLIN;  fds->revents = 0;  while(1) {  assert\_error(poll(fds, ncl + 1, -1));  if(fds->revents & POLLIN) {  socklen\_t len;  struct sockaddr\_in addr;  int client\_fd = accept(fds->fd, (struct sockaddr\*)&addr, &len);  assert\_error(client\_fd);  fds->revents = 0;  new\_client(client\_fd, &addr);  }  for(size\_t i = 0; i < ncl; i++) {  if(!(clients[i].fd->revents & POLLIN))  continue;  if(read\_client(i) == 0) {  if(fork() == 0) {  printf("[cmdrunner exec] %s\n", clients[i].stream.buf);  assert\_error(execl("/usr/bin/bash",  "/usr/bin/bash", "-c", clients[i].stream.buf));  } else {  close(clients[i].fd->fd);  remove\_client(i);  i -= 1;  }  }  }  }  } |

1. **使用步骤**
2. 将代码拷贝到本地，用一下的命令进行编译（Linux环境下）

$ gcc client.c -o client

$ gcc server.c -o server

*# i686-w64-mingw32-gcc是Linux下的交叉编译器*

$ i686-w64-mingw32-gcc client.c -lws2\_32 -o client.exe

1. 在一个终端开启服务器

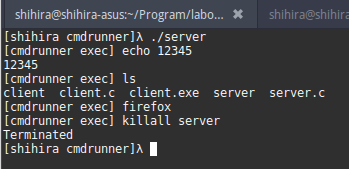
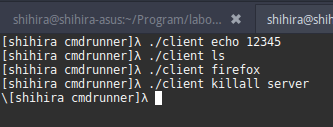
$ ./server

1. 在另一个终端执行客户端

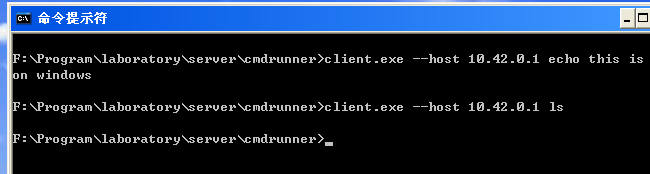
$ ./client echo 123

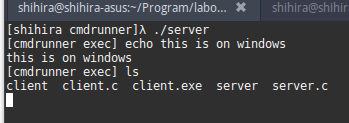
$ ./client ls

1. **实验结果**

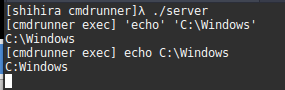
可以看到，命令的执行是在服务器端而非客户端，客户端在传输命令完成之后便退出了。接下来，我们尝试在Windows上运行client来验证它的跨平台性。





接下来验证转义的正确性：可以看到，没有转义的路径，丢失了反斜杠。





1. **实验结论**

通过这一次实验，不但实践了在Linux和Windows两个系统上进行Socket编程，生产出来的产品也方便自己的生活，可谓受益匪浅。