Project I :TCP Scanner

Scott Jin

February 19, 2018

Contents

List of Figures

1 Code Listings

Scott Jin——PortScanner.c

```c
/* PortScanner.c
 *  Created on: Feb 15, 2018
 *  Author: scott Jin
 */
#include <stdio.h>
#include <sys/wait.h>
#include<sys/socket.h>
#include<errno.h>
#include<netdb.h>
#include <arpa/inet.h>
#include<string.h>
#include<stdlib.h>
#include <unistd.h>
#include <ctype.h>
#include <sys/time.h>
#define OUT __stdoutp
#define EXIT_FAIL -1

void port_scanner (char*, int, int);
void waiting(int);
void port_scanner_2 (char*, int, int);
void forkChildren(char*, int, int);
void spawning(char*, int, int);

int main(int argc , char ** argv) {
    //initialization
    int pflag = 0, starting_port = 0, ending_port = 0, opt = 0, match_count = 0;
    char* hostname = 0;
    if (argc != 2 && argc != 4) {
        fprintf(stderr,"ERROR-->Incorrect␣format:%s\n␣Usage:␣PortScanner␣hostname␣[-p␣15:25]\n",argv[0]);
        exit(EXIT_FAIL);
    }
    hostname = argv[1];
    optind = 2;
    while ((opt = getopt(argc, argv, "p:")) != -1) {
        switch (opt) {
            case 'p':
                if (!strcmp ("-p", optarg) || pflag++>0) {
                    fprintf(stderr,"ERROR->Invalid␣argument:␣only␣one␣'-p'␣flag␣can␣be␣accepted.\n");
                    exit(EXIT_FAIL);
                } else if(!strcmp ("-", optarg) || !strcmp ("", optarg)){
                    fprintf(stderr, "ERROR->Void␣argument:␣no␣argument␣provided!\n");
                    exit(EXIT_FAIL);
                } else if((match_count = sscanf(optarg, "%d:%d", &starting_port, &ending_port)) < 2) {
                    fprintf(stderr, "ERROR->Invalid␣argument:␣Two␣ports␣must␣be␣given␣for␣a␣range\n");
                    exit(EXIT_FAIL);
                }
                if(starting_port > ending_port || starting_port < 0) {
                    fprintf(stderr, "ERROR->Invalid␣argument:␣ports␣must␣be␣postive␣and␣range-based\n");
                    exit(EXIT_FAIL);
                }
                break;
            case '?':
                fprintf(stderr,"ERROR-->Invalid␣option(or␣missing␣argument):%c\n␣Usage:␣PortScanner␣hostname␣[-p␣15:25]\n",optopt);
                exit(EXIT_FAIL);
                break;
            default:
                fprintf(stderr,"ERROR-->Incorrect␣format:%s\n␣Usage:␣PortScanner␣hostname␣[-p␣15:25]\n",argv[0]);
                exit(EXIT_FAIL);
        }
    }
    if (pflag == 0) {
        starting_port = 0;
        ending_port = 1024;
        fprintf(stderr, "WARNING->No␣ports␣range␣specified,␣"
                "Using␣Default␣Value:␣starting_port␣=␣%d,␣ending_port␣=␣%d\n", starting_port, ending_port);
        fprintf(stderr,"Please␣be␣Patient␣since␣1024␣Three-HANDSHAKE␣are␣being␣attempted\n");
        port_scanner (hostname, starting_port, ending_port);
        return(0);
    }
    forkChildren(hostname, starting_port, ending_port);
    return(0);
}
void forkChildren (char* hostname, int starting_port, int ending_port) {
    int i;
    pid_t pid;
    for (i = starting_port; i <= ending_port; i++) {
        pid = fork();
        if (pid == -1) {
            perror("fork");
            exit(EXIT_FAIL);
        }
        if (pid == 0) {
            port_scanner (hostname, i, i);
            return;
        }
        if(i == ending_port && pid > 0) {
            waiting(20);
        }
    }
    return;
}
void port_scanner (char* hostname, int starting_port, int ending_port) {
    //Initialise the sockaddr_in structure
    struct hostent *host; struct sockaddr_in si;
    int err, i , sock_num;
    strncpy((char*)&si , "" , sizeof si);
    si.sin_family = AF_INET;

    if(isdigit(hostname[0])) { //direct ip
        fprintf(stderr,"Identifying␣direct␣IP...\n");
        si.sin_addr.s_addr = inet_addr(hostname);
    } else if( (host = gethostbyname(hostname)) != 0) { //translate
        fprintf(stderr,"Retrieving␣direct␣IP...\n");
        strncpy((char*)&si.sin_addr , (char*)host->h_addr , sizeof si.sin_addr);
    } else {
        herror(hostname);
        exit(EXIT_FAIL);
    }
    fprintf(stderr,"Port␣Scanning\n");
    for( i = starting_port ; i < ending_port + 1; i++) {
        waiting(5);
        si.sin_port = htons(i);          //Fill in the port number in network byte order
        sock_num = socket(AF_INET , SOCK_STREAM , 0);          //Create a socket of type internet
        if(sock_num < 0) {
            perror("\nSocket");
            continue;
        }
        //Connect using that socket and sockaddr structure
        err = connect(sock_num , (struct sockaddr*)&si , sizeof si);
        if (err < 0) {          //not connected
```

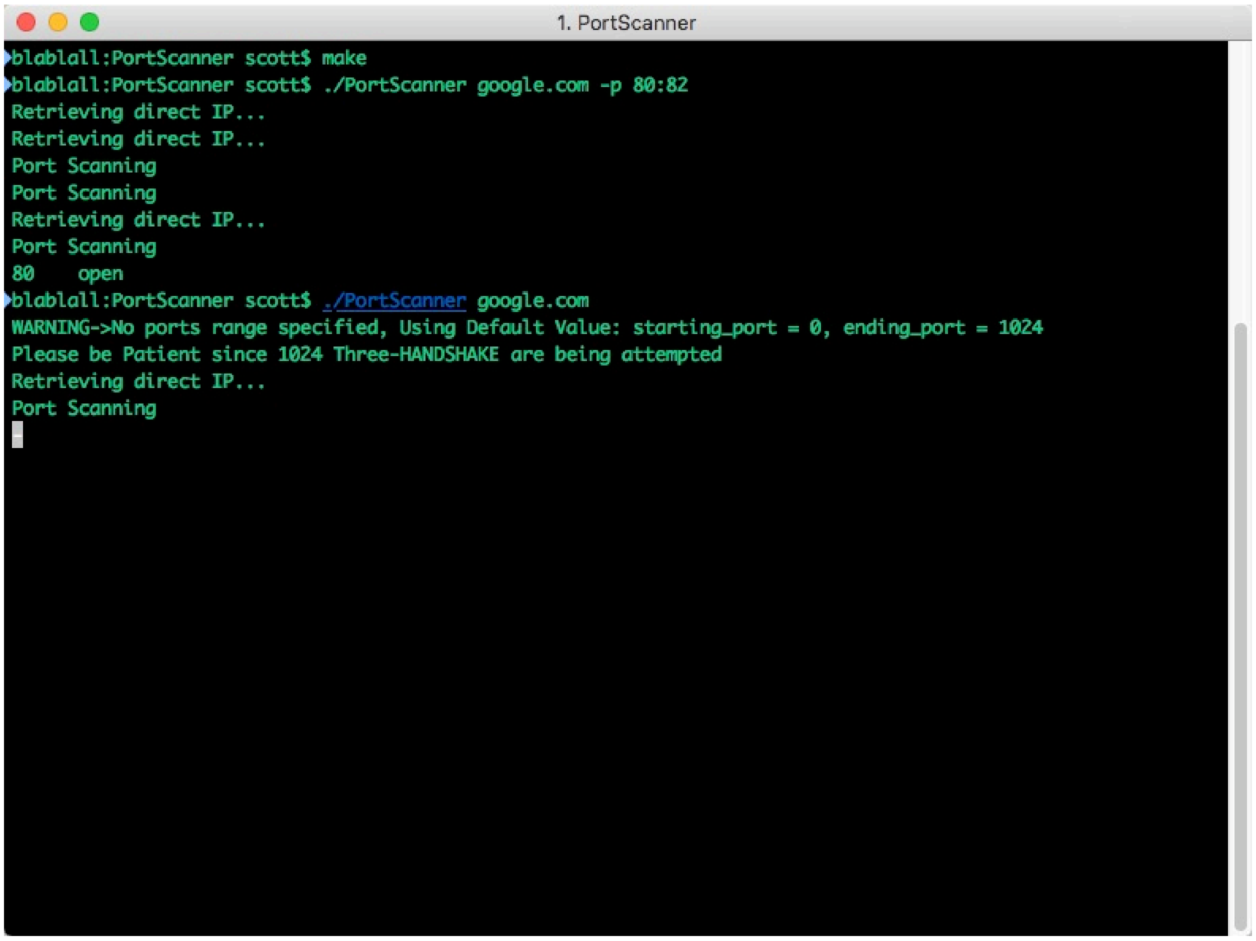
```c
            fflush(OUT);
        } else {
            printf("%-5d␣open\n", i);
        }
        close(sock_num);
    }
    fflush(OUT);
}
void waiting(int a) {
    char chars[] = {'-', '\\', '|', '/'};
    unsigned int i;
    for (i = 0; i < a; ++i) {
        printf("%c\r", chars[i % sizeof(chars)]);
        fflush(stdout);
        usleep(200000);
    }
}
//another implementation using addrinfo struct
void port_scanner_2 (char* hostname, int starting_port, int ending_port) {
    for (int port = starting_port; port <= ending_port; port++) {
        struct addrinfo hints;
        memset(&hints, 0, sizeof(hints));
        hints.ai_family = AF_INET;
        hints.ai_socktype = SOCK_STREAM;
        struct addrinfo *serv_addr=NULL;
        char tport[6]={0};
        sprintf(tport, "%d", port);   // Converts the int to char* type
        if(getaddrinfo(hostname, tport, &hints, &serv_addr)==0) {
            struct addrinfo *temp=NULL;
            int sockfd=0;
            int status=0;
            for(temp= serv_addr; temp != NULL; temp = temp->ai_next) {
                sockfd = socket(temp->ai_family, temp->ai_socktype, temp->ai_protocol);   // Creating a socket
                if (sockfd < 0) {       // socket creation fails.
                    continue;
                }
                status = connect(sockfd, temp->ai_addr, temp->ai_addrlen);    // Try connecting to the socket
                if (status<0) {     //  connection fails
                    close(sockfd);
                    continue;
                }
                printf("Port␣%d␣is␣open.\n", port);
                close(sockfd);
            }
            freeaddrinfo(serv_addr);
        } else {
            freeaddrinfo(serv_addr);
        }
    }
}
```

Scott Jin——makefile

```
ALL:
  @gcc -o PortScanner PortScanner.c

.PHONY:debug,clean
debug:
  @gcc -o Portdebug  PortScanner.c -g
clean:
  rm -f *.dSYM
  rm Portdebug
  rm PortScanner
```

2 Experimental Screenshots



```
blablall:PortScanner scott$ make
blablall:PortScanner scott$ ./PortScanner google.com -p 80:82
Retrieving direct IP...
Retrieving direct IP...
Port Scanning
Port Scanning
Retrieving direct IP...
Port Scanning
80    open
blablall:PortScanner scott$ ./PortScanner google.com
WARNING->No ports range specified, Using Default Value: starting_port = 0, ending_port = 1024
Please be Patient since 1024 Three-HANDSHAKE are being attempted
Retrieving direct IP...
Port Scanning
```

Figure 1: Test result for PortScanner