TCP Port Forwarder

Mario Enriquez

British Columbia Institute of Technology

COMP 8005, 6D

Aman Abdullah

April 4, 2015

Contents

Introduction

In this Assignment I designed a minimum functionality port forwarding server that works as a proxy between 2 machines using the TCP protocol. The goal is to create and observe how my proxy handles the TCP connections between 2 machines. To achieve this I'm going to make some basic functionality tests that will involve simply accessing a protocol in the target machine, while in the second type of tests I will test the capacity for the port forwarder for handling heavy loads of traffic, by forwarding the traffic of multiple clients to one or multiple servers.

Design Work

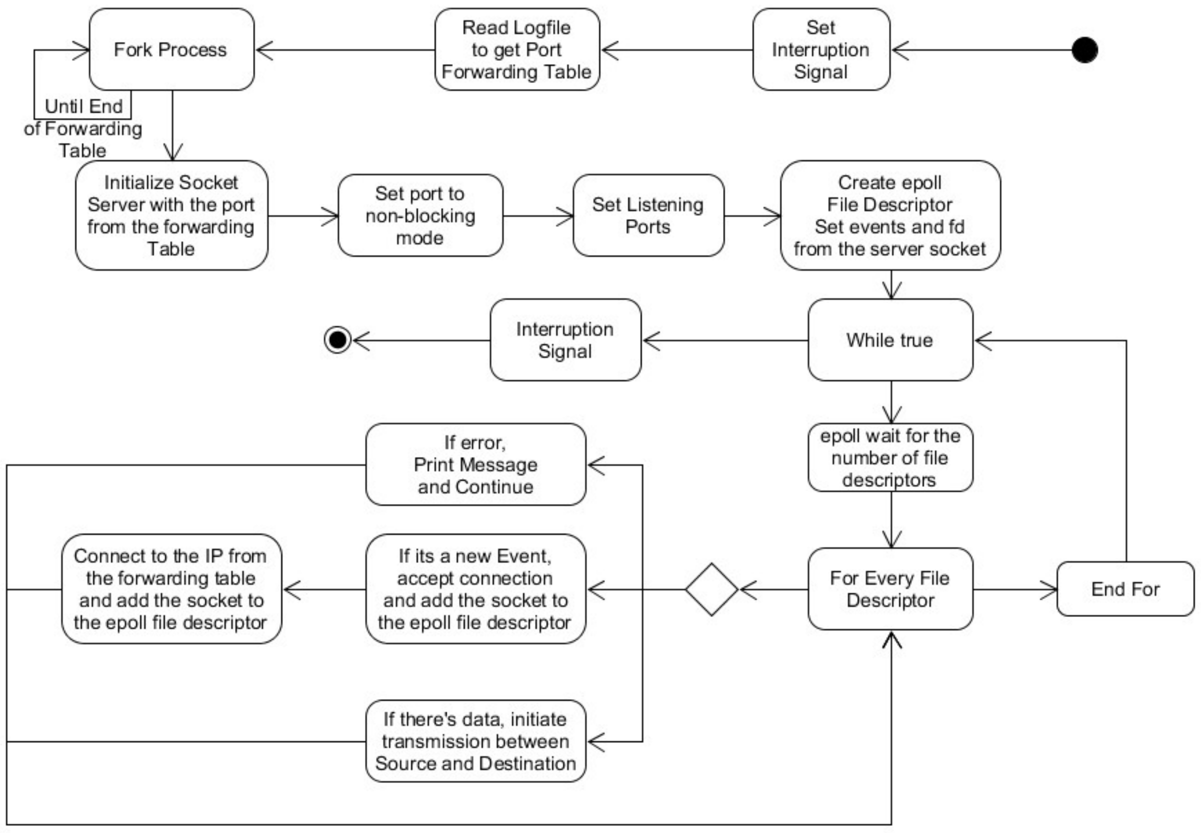
Fork Process
Read Logfile to get Port Forwarding Table
Set Interruption Signal
Until End of Forwarding Table
Initialize Socket Server with the port from the forwarding Table
Set port to non-blocking mode
Set Listening Ports
Create epoll File Descriptor Set events and fd from the server socket
Interruption Signal
While true
If error, Print Message and Continue
epoll wait for the number of file descriptors
Connect to the IP from the forwarding table and add the socket to the epoll file descriptor
If its a new Event, accept connection and add the socket to the epoll file descriptor
For Every File Descriptor
End For
If there's data, initiate transmission between Source and Destination

Figure 1 State Machine Diagram

Instructions

Requirements gcc and g++ installed.

Unix:

- Compile the file proxy.cpp using the command:

  # g++ -Wall –o proxy proxy.cpp

- Execute the program by using the command:

  ./proxy [Forwarding Table File Directory] [Log File Directory]

- For the Forwarding table the format should be as follows:

  "Source Port" "IP" "Destination Port"

Test Cases

Captures for the cases can be found in the Annex or in the case of Wireshark captures or logs in their own directory. They will consist on a copy of the logs, Wireshark captures, and screenshots.

Test	Description	Tool/Command	Expected Results	Actual Results
1	Test SSH connectivity by attempting to connect to the port forwarder	SSH	Port forwarder redirects to the target machine	Success, the Server redirected the SSH connection to another machine
2	Test connectivity to a webserver by connecting to the port forwarder	Web browser, Firefox	We can see the Site mounted in the target machine	Success, the Server redirected to the homepage from another machine
3	Test connectivity to ftp by downloading a file from the port forwarder	FTP server	We can download a file from a remote machine	Success, the client could connect to the vsftpd server from

				the other machine
4	Test a simple Client/Server program	Echo Server, Client	Remote machine will echo the client machine	Success, Connection works well and sends/receives data
5	Test a Multiple Server program that can handle at least 2500 clients consecutively	Multiple server program	Port forwarder maintains the server and clients	Success, Forwarder transmit the information between client-servers
6	Test a Multiple Server program that can handle at least 5000 clients consecutively	Multiple server program	Port forwarder maintains the server and clients	Success, Forwarder transmit the information between client-servers
7	Test a Multiple Server program that can handle at least 7500 clients consecutively	Multiple server program	Port forwarder maintains the server and clients	Success, Forwarder transmit the information between client-servers albeit it has difficulty going forward

Observations

The port forwarding server worked well and was able to communicate two machines successfully. One of the main challenges I had was knowing when one of the sides disconnected to update the fd in my epoll fd, as it caused me a bit of trouble when working with both the protocols and my server-client.

This application with a little tweaking could be useful to create a load balancing server, as it could redirect the traffic inside a network to different servers depending the workload of each one.

Pseudocode

```
proxy.cpp
include the libraries stdio.h, netdb.h, sys/types.h, sys/socket.h, netinet/in.h, errno.h, sys/time.h, stdlib.h, strings.h, string.h, sys/ioctl.h, arpa/inet.h, unistd.h, assert.h, fcntl.h, sys/epoll.h, signal.h, iostream, fstream, vector, ctime
define the static variables IPLIST with a filename, LOGFILE with a filename, FAILURE, SUCCESS, LISTEN_PORTS, BUFLEN, AX_EPOLL_EVENTS_PER_RUN, RUN_TIMEOUT, PROCESSES
reference functions ErrorMessage, close_connection,, set_non_blocking, forwarding, getForwardTable, print_into_file
define global variables fw_server,assignedPort and logfile
define structure ip_port{
    variables srcport, ip and dstport
}
define structure fd_sock{
    variables src, dst, src_ip and dst_ip
}
Main function (Receives arguments)
{
        Define varibles i, pid, portNo, nfds, fw_new, fw_socket, arg, conn, fw_server,
epoll_counter, fd_sockets, forwardList, iplist,ip_remote,message, interrupt, events, event,
epoll_fd,src,dst, remote_addr,port_addr,server_addr, addr_size, host, hostip;
        switch (number of arguments){
            2 arguments:
                iplist is argument 2
                logfile is static variable LOGFILE
            3 arguments:
                iplist is argument 2
                logfile is argument 3
            default:
                iplist is static variable IPLIST
                logfile is static variable LOGFILE
        }
        interrupt.sa_handler = close_connection;
        interrupt.sa_flags = 0;
        if  the interruption signal cannot be set
        {
             Call function ErrorMessage with "Failed to set Interruption Signal."
        }
        if( the function getForwardTable returns failure){
            Call function ErrorMessage with "Forwarding Table is empty."
        }
```

```
Fork based on the port list
Set assignedPort to source port
if server socket cannot be initialized {
        Call function ErrorMessage with "Failure to set the socket server"
}
if  the server socket cannot reuse address {
        Call function ErrorMessage with "Cannot set port to be reusable"
}
Call function set_non_blocking with fw_server
Set server address and port information
if socket cannot bind{
        Call function ErrorMessage with "Cannot bind"
}
if  socket cannot set listening ports {
        Call function ErrorMessage with "Cannot set Listening ports."
}
 Create epoll file descriptor
 if creation of epoll file descriptorfails {
        Call function ErrorMessage with "Cannot create epoll queue"
 }
        Set epoll events EPOLLIN | EPOLLERR | EPOLLHUP;
        Add server data to epoll data
        if epoll file descriptor cannot be set {
```

```
                        Call function ErrorMessage with "epoll_ctl"
                        }
                while ( always){
                        if epoll wait cannot be set{
                                Call function ErrorMessage with "Error in the epoll wait!"
                        }
                        for all the files in nfds{
                                if  there is an EPOLLHUP or EPOLLERR event {
                                        call function print_into_file with "epoll:EPOLLERR\n"
                                        close source and destination sockets
                                        continue;
                                }
                                assert (events[i].events & EPOLLIN);
                                if  server receives new connection {
                                        if server cannot accept new connection{
                                                if  EAGAIN and EWOULDBLOCK flags are not
used {
                                                        print error "accept"
                                                        print "Could not Accept"
                                                }
                                                continue;
                                        }
                                        Get remote client address
```

```
call function set_non_blocking with new connection
set events EPOLLIN | EPOLLERR | EPOLLHUP
add new socket to epoll event data
if epoll file descriptor cannot be set {
        Call function ErrorMessage with "epoll_ctl"
}
epoll_counter++;
if client name cannot be obtained{
        Call function ErrorMessage with "Failed to get port"
}
get client hostname
if remote server socket cannot be initialized{
        Call function ErrorMessage with "Failure to set the socket"
}

if address cannot be reused{
        Call function ErrorMessage with “Cannot set port to be reusable”
}
Set remote server attributes
If port forwarder can connect to remote server{
```

```
                        call function set_non_blocking to remote server
socket
                    } else {
                        Close remote server socket
                    }
                    set events EPOLLIN | EPOLLERR | EPOLLHUP
                    add new socket to epoll event data
                    if epoll file descriptor cannot be set {
                     Call function ErrorMessage with "epoll_ctl"
                    }
                    Add 1 to epoll counter
                    if we cannot obtain the structure of current remote server{
                        Call function ErrorMessage with "Failed to get
port"
                    }
                    Add client and remote server to array fd_sockets
                     Set message with port and ip information
                    Call print_into_file with message
                    continue;
                }
                If forwarding the current file descriptor with an event returns 0 {
                    close source and destination sockets
                    Set message "Closing Connection from source to destiation
```

```
                                        Call print_into_file with message
                                }
                }
        }
        Close fw_server
        Exit with success
}
function set_non_blocking(receives file descriptor)
{
        if  file descriptor cannot be set into nonblocking{
                call ErrorMessage with message "fcntl"
        }
}
function ErrorMessage(receives interruption){
        prints message
        exit with failure
}
function close_connection (receives interruption){
        close fw_server
        call print_into_file with message "Closing Connection..."
        exit with success
}
function getForwardTable(list forwardList, and filename iptable){
        variables result, newIpPort, ip, srcport,dstport;
        while we find srcport, ip and dstport in archive{
            set values to newIpPort
            add newIpPort to forwardList
        }
        if forwardList is not empty{
            return success;
        }
        return failure;
    }
    function forwarding(structure containing source and destination sockets){
        declare variables n, bytes_to_read,total_sent,bp, buf[BUFLEN]
        set bp to buf
        set bytes_to_read as BUFLEN
        set total_sent as 0
        while number of received bytes is higher than 0 {
            add n to bp
            substract n from bytes_to_read
            add n to total_send
            redirect data to destination socket
        }
        if(total_sent is 0){
            return 0
            } else {
                    return 1
            }
        }
        function print_into_file(receives info){
                    get current time
                    open current log file
                    update log with info
}
```