Network Exercise

for Unix network programming

WiLNA LAB[1]

December 7, 2018

[1] https://wilnalab.github.io

Contents

Preface

This is an exercise mannual for advanced C and Unix network programming course. We hope you can learn something in this course. Good Luck!

1

EX NO.1

> *"It seemed the world was divided into good and bad people. The good ones slept better... while the bad ones seemed to enjoy the waking hours much more."*
>
> – Woody Allen, *1935-?*

1.1 Introduction

In this chapter, we will learn the concept of shell, some basic commands, Vim, Git and Make. All those are introduced in following chapters.

1.2 Shell

1.2.1 Introduction

In computing, a shell is a user interface for access to an operating system's services. In general, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI), depending on a computer's role and particular operation. It is named a shell because it is the outermost layer around the operating system kernel.

CLI shells require the user to be familiar with commands and their calling syntax, and to understand concepts about the shell-specific scripting language (for example bash script). They are also more easily operated via refreshable braille display, and provide certain advantages to screen readers.

Graphical shells place a low burden on beginning computer users, and are characterized as being easy to use. Since they also come with certain disadvantages, most GUI-enabled operating systems also provide CLI shells.

1.2.2 Basic command

Here are some basic and useful commands that the new beginners should master.

- cd - change directory
- ls - list directory content
- apt - package manager
- sl, cowsay - interesting command
- man - mannual of a command
- whatis - simple introduction of a command
- other..

1.3 Vim

1.3.1 What is Vim?

Vim (a contraction of Vi IMproved) is a clone, with additions, of Bill Joy's vi text editor program for Unix. It was written by Bram Moolenaar based on source for a port of the Stevie editor to the Amiga and first released publicly in 1991. Vim is designed for use both from a command-line interface and as a standalone application in a graphical user interface. Vim is free and open-source software.

Although it was originally released for the Amiga, Vim has since been developed to be cross-platform, supporting many other platforms. In 2006, it was voted the most popular editor amongst Linux Journal readers; in 2015 the Stack Overflow developer survey found it to be the third most popular text editor, and the fifth most popular development environment in 2018.

1.3.2 Basic operations

Basic operations are listed below. More can be found by search "vim cheat sheet".

- h, j, k, l - move
- i - insert mode
- ^ move head of line
- $ - end of line
- : - input command

- :w - write
- :q - quit
- ESC - return to normal mode

1.4 Git

1.4.1 What is Git

By far, the most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel. A staggering number of software projects rely on Git for version control, including commercial projects as well as open source. Developers who have worked with Git are well represented in the pool of available software development talent and it works well on a wide range of operating systems and IDEs (Integrated Development Environments).

Having a distributed architecture, Git is an example of a DVCS (hence Distributed Version Control System). Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.

In addition to being distributed, Git has been designed with performance, security and flexibility in mind.

1.4.2 How to get Git

There are several ways to get Git. For our experiment, just type ‘su‘ and then ‘pkg install git‘ in your terminal and wait for the end of installation.

If you have intrest on Git, you can go to its offical site for more ditails. `https://Git-scm.com/`

1.4.3 Basic usage

Here we will create our first repo in Github and commit our code.

- register GitHub account
- create remote(online) repo
- Git clone your-repo-url
- do some experiment. For example, create a readme.md file
- Git add .

- Git commit -m “your commit message”
- Git push
- input your account and password
- now your remote code repository is updated

A small work to do is to find how to make “Git push” without password.

1.5 Make

1.5.1 What is Make

In software development, Make is a build automation tool that automatically builds executable programs and libraries from source code by reading files called Makefiles which specify how to derive the target program. Though integrated development environments and language-specific compiler features can also be used to manage a build process, Make remains widely used, especially in Unix and Unix-like operating systems.

Besides building programs, Make can be used to manage any project where some files must be updated automatically from others whenever the others change.

1.5.2 What is a Makefile

A makefile is a file (by default named "Makefile") containing a set of directives used by a make build automation tool to generate a target/goal.

1.5.3 Compile a c program by using gcc

Walk around and find or just write a simple c program. And use “cc yourfile.c -o new-name” to compile and get binary executable file. Run it. Great! You compiled your first program by hand!

Well, what will happen if you have 10 or more source files? Each time that hand-compile will make you headache.

1.5.4 Compile a c program by using Make

We already have the source code of our textbook in our virtual machine. And we will compile one of them.

1.6 Exercises

1.6.1 EX-1

- use basic commands in your terminal and tell their usage.

- create a “test” dir and some c programs, compile and run.
- compile a project by using make

2

EX NO.2

"I have never met a man so ignorant that I couldn't learn something from him."

– Galileo Galilei *1564 - 1642*

2.1 Introduction

In this chapter, we will review what we've learned before, and then learn the use of GitHub, make our first repository and get our C/S code running.

2.2 ctags

2.2.1 Introduction

Ctags is a tool that makes it easy to navigate large source code projects. It provides some of the features that you may be used to using in Eclipse or other IDEs, such as the ability to jump from the current source file to definitions of functions and structures in other files.

2.2.2 Useful Commands

- Generate tags: ctags -R *
- Jump to the tag underneath the cursor: Ctrl-]
- Search for a particular tag: :ts <tag> <RET>
- Go to the next/previous definition for the last tag: :tn/:tp
- Jump back up in the tag stack: Ctrl-t
- Jump back/forward: Ctrl-o/Ctrl-i

2.3 GitHub

2.3.1 What is GitHub?

GitHub is a web-based hosting service for version control using Git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

2.3.2 Practice

- create account
- fork our execise repository

  `https://github.com/ginhton/practice`
- clone the code into your local machine
- edit
- add and commit
- git push

2.3.3 Useful Link

- Ruan Yifeng blog

  `https://blog.csdn.net/syh_486_007/article/details/71159278`
- runoob tutorial

  `http://www.runoob.com/git/git-tutorial.html`

2.4 Network Tools

2.4.1 tcpdump

In most cases you will need root permission to be able to capture packets on an interface. Using tcpdump (with root) to capture the packets and saving them to a file to analyze with Wireshark (using a regular account) is recommended over using Wireshark with a root account to capture packets on an "untrusted" interface.

- See the list of interfaces on which tcpdump can listen:

  tcpdump -D

- Listen on interface eth0:

  tcpdump -i eth0

- Listen on any available interface (cannot be done in promiscuous mode. Requires Linux kernel 2.2 or greater):

  tcpdump -i any

- Be verbose while capturing packets:

  tcpdump -v

- Be more verbose while capturing packets:

  tcpdump -vv

- Be very verbose while capturing packets:

  tcpdump -vvv

- Be verbose and print the data of each packet in both hex and ASCII, excluding the link level header:

  tcpdump -v -X

- Be verbose and print the data of each packet in both hex and ASCII, also including the link level header:

  tcpdump -v -XX

- Be less verbose (than the default) while capturing packets:

  tcpdump -q

- Limit the capture to 100 packets:

  tcpdump -c 100

- Record the packet capture to a file called capture.cap:

  tcpdump -w capture.cap

- Record the packet capture to a file called capture.cap but display on-screen how many packets have been captured in real-time:

  tcpdump -v -w capture.cap

- Display the packets of a file called capture.cap:

  tcpdump -r capture.cap

- Display the packets using maximum detail of a file called capture.cap:

  tcpdump -vvv -r capture.cap

- Display IP addresses and port numbers instead of domain and service names when capturing packets (note: on some systems you need to specify -nn to display port numbers):

  tcpdump -n

- Capture any packets where the destination host is 192.168.1.1. Display IP addresses and port numbers:

  tcpdump -n dst host 192.168.1.1

- Capture any packets where the source host is 192.168.1.1. Display IP addresses and port numbers:

  tcpdump -n src host 192.168.1.1

- Capture any packets where the source or destination host is 192.168.1.1. Display IP addresses and port numbers:

  tcpdump -n host 192.168.1.1

- Capture any packets where the destination network is 192.168.1.0/24. Display IP addresses and port numbers:

  tcpdump -n dst net 192.168.1.0/24

- Capture any packets where the source network is 192.168.1.0/24. Display IP addresses and port numbers:

  tcpdump -n src net 192.168.1.0/24

- Capture any packets where the source or destination network is 192.168.1.0/24. Display IP addresses and port numbers:

  tcpdump -n net 192.168.1.0/24

- Capture any packets where the destination port is 23. Display IP addresses and port numbers:

  tcpdump -n dst port 23

- Capture any packets where the destination port is is between 1 and 1023 inclusive. Display IP addresses and port numbers:

  tcpdump -n dst portrange 1-1023

- Capture only TCP packets where the destination port is is between 1 and 1023 inclusive. Display IP addresses and port numbers:

  tcpdump -n tcp dst portrange 1-1023

- Capture only UDP packets where the destination port is is between 1 and 1023 inclusive. Display IP addresses and port numbers:

  tcpdump -n udp dst portrange 1-1023

- Capture any packets with destination IP 192.168.1.1 and destination port 23. Display IP addresses and port numbers:

  tcpdump -n "dst host 192.168.1.1 and dst port 23"

- Capture any packets with destination IP 192.168.1.1 and destination port 80 or 443. Display IP addresses and port numbers:

  tcpdump -n "dst host 192.168.1.1 and (dst port 80 or dst port 443)"

- Capture any ICMP packets:

  tcpdump -v icmp

- Capture any ARP packets:

  tcpdump -v arp

- Capture either ICMP or ARP packets:

  tcpdump -v "icmp or arp"

- Capture any packets that are broadcast or multicast:

  tcpdump -n "broadcast or multicast"

- Capture 500 bytes of data for each packet rather than the default of 68 bytes:

  tcpdump -s 500

- Capture all bytes of data within the packet:

  tcpdump -s 0

2.4.2 netstat

netstat (network statistics) is a command-line tool that displays network connections (both incoming and outgoing), routing tables, and a number of network interface statistics.

- View only established connection:

  netstat -natu | grep 'ESTABLISHED'

- View only connection is in listen status:

  netstat -an | grep 'LISTEN'

- View port number used by PID:

  netstat -anlp | grep $PID

- Show statistics for all protocols:

  netstat -s

- Show kernel routing information:
  netstat -r
- Show which process using particular port:
  netstat -anlp | grep portnumber
- Show the list of network interfaces:
  netstat -i

2.4.3 lsof

list open files

- List all opened files by processes for a specific user:
  lsof -u user_name
- List all opened files for processes running on a specific port:
  lsof -i :port_number
- Lists open files for TCP port ranges 1-1024:
  lsof -i :1-1024
- List all network connections.:
  lsof -i

2.5 Client/Server Project

2.5.1 Introduction

This is mainly an exercise for students to do. If you have any problem in programming, you can refer to our textbook, code example or ask the teach assistant directly.

2.5.2 Target

- create echo server
- create echo client
- write a Makefile to compile your code
- run successfuly
- commit your code to your GitHub repository

2.5.3 *Optional

- make README.md more beautiful
- server can bind different port
- server reply different content according to client's message
- cross-machine experiment

3

EX NO.3

"If it weren't for my lawyer, I'd still be in prison. It went a lot faster with two people digging."

– Joe Martin

3.1 Introduction

In this chapter, we will talk more about mirrors, Unix network C/S models, functions and small tips about Git. We will start to pramming. Let's get hands dirty!

3.2 Mirrors

3.2.1 What is Mirrors?

Mirror websites or mirrors are replicas of other websites. The main purpose of mirrors is often to reduce network traffic, improve access speed, or to improve availability of the original site. Such websites have different URLs than the original, but host identical content to it. Mirrors can also serve as real-time backups.

3.2.2 Configure our freebsd mirrors

- switch to root user by **su**
- create file **/usr/local/etc/pkg/repos/FreeBSD.conf**
- edit it

```
FreeBSD: {
  url: "pkg+http://mirrors.ustc.edu.cn/freebsd-pkg/${ABI}/quarterly",
}
```

- save and exit editor
- run **pkg update -f**

3.3 Unix Network Client/Server

3.3.1 What is a socket?

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else.

To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

Sockets were first introduced in 2.1BSD and subsequently refined into their current form with 4.2BSD. The sockets feature is now available with most current UNIX system releases.

3.3.2 Where is Socket Used?

A Unix Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.

3.3.3 2-tier and 3-tier architectures

There are two types of client-server architectures.

- **2-tier architecture**: In this architecture, the client directly interacts with the server. This type of architecture may have some security holes and performance problems. Internet Explorer and Web Server work on two-tier architecture. Here security problems are resolved using Secure Socket Layer (SSL).
- **3-tier architectures**: In this architecture, one more software sits in between the client and the server. This middle software is called 'middleware'. Middleware are used to perform all the security checks and load balancing in case of heavy load. A middleware takes all requests from the client and after performing the required authentication, it passes that request to the server. Then the server does the required processing and sends the response back to the middleware and finally

the middleware passes this response back to the client. If you want to implement a 3-tier architecture, then you can keep any middleware like Web Logic or WebSphere software in between your Web Server and Web Browser.

3.3.4 How to Make a Client?

The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. Both the processes establish their own sockets.

The steps involved in establishing a socket on the client side are as follows.

- Create a socket with the socket() system call.
- Connect the socket to the address of the server using the connect() system call.
- Send and receive data. There are a number of ways to do this, but the simplest way is to use the read() and write() system calls.

3.3.5 How to Make a Server?

The steps involved in establishing a socket on the server side are as follows.

- Create a socket with the socket() system call.
- Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the listen() system call.
- Accept a connection with the accept() system call. This call typically blocks the connection until a client connects with the server.
- Send and receive data using the read() and write() system calls.

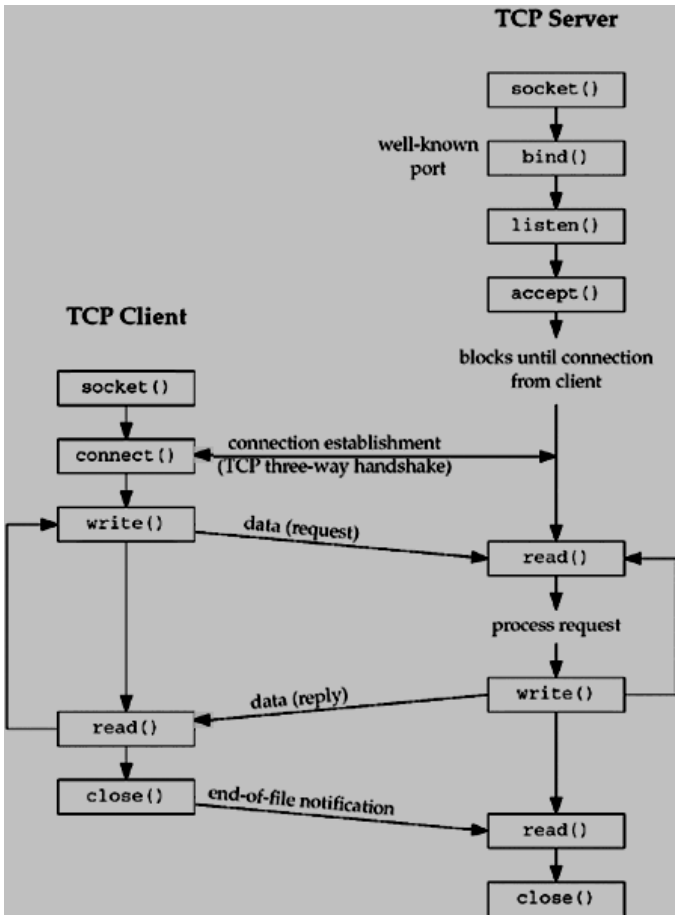
3.3.6 Client and Server Interaction

All the details about functions involved can be find in our textbook.

3.4 More about Git

3.4.1 Git .ignore

.gitignore tells git which files (or patterns) it should ignore. It's usually used to avoid committing transient files from your working directory that aren't


TCP Server
socket()
well-known port
bind()
listen()
accept()
blocks until connection from client
TCP Client
socket()
connect()
connection establishment (TCP three-way handshake)
write()
data (request)
read()
process request
write()
data (reply)
read()
close()
end-of-file notification
read()
close()

useful to other collaborators, such as compilation products, temporary files IDEs create, etc.

3.4.2 Git Tags

A tag is used to label and mark a specific commit in the history. It is usually used to mark release points (eg. v1.0, etc.).

Although a tag may appear similar to branch, a tag, however, does not change. It points directly to a specific commit in the history.

Here is a link include more details about Git tags.

3.5 Intesting Task

Those tasks are optional. Just for fun.

- ren gong zhi neng tian gou - built on C/S echo server
- browser plugin - vimium
- learn Git branching by playing this game