Electrical and Computer Engineering (ECE) Operating Systems and System Programming ECE254 Laboratory Manual

by

Yiqing Huang
Paul A.S. Ward

Electrical and Computer Engineering Department
University of Waterloo

Waterloo, Ontario, Canada, May 5, 2014

Contents

List of Tables

List of Figures

Preface

Two operating systems are used in laboratories. One is the general purpose operating system Linux that supports Intel processors on personal computers. The second is the ARM RL-RTX that supports ARM Cortex-M3 processors on Keil MCB1700 boards.

The Linux computing environment is for practising system programming aspect of the course. The ARM RL-RTX, a real-time operating system library, is for practising the operating system kernel programming aspect of the course.

The first purpose of this document is to provide the descriptions of each laboratory project. The second purpose of this document is a quick reference guide of the relevant development tools for completing laboratory projects.

Who Should Read This Lab Manual?

This lab manual is written for students who are taking Electrical and Computer Engineering (ECE) Operating Systems and System programming course ECE254 in the University of Waterloo.

What is in This Lab Manual?

This manual is divided into three parts.

Part I describes the lab administration policies

Part II is a set of course laboratory projects as follows.

- Lab0A: Introduction to Linux system programming
- Lab0B: Introduction to ARM RL-RTX kernel and application programming
- Lab1: Task management in ARM RL-RTX

- Lab2: Inter-process communication by message passing
- Lab3: Concurrency control

Part III is the reference guide of the development tools for Linux and ARM RL-RTX. The main topics are as follows.

- Software Development Tools on Linux
  - Linux hardware environment
  - Editors
  - Compiler
  - Debugger
  - Utility to automate build
  - Utility for version control
- Keil MCB1700 Development Hardware Environment and Software Tools
  - Keil MCB1700 hardware environment
  - Keil MCB1700 software development Tools
  - Programming MCB1700 with ARM RL-RTX
    - Building an RTX application
    - Building a customized ARM RL-RTX library
    - Creating an application with customized ARM RL-RTX library
  - Programming MCB1700

Acknowledgments

We would like to sincerely thank our students who took ECE254 and MTE241 [1] courses in the past two years. They provided constructive feedback every term to make the manual more useful to address problems that students would encounter when working on each lab assignment.

Special thank goes to Dr. Thomas Reidemeister who shared his prototyping work in SE350 [2] course project on Keil MCB1700 boards with us.

[1]MTE241 is the course number for Introduction to Real-time Systems in the University of Waterloo Mechatronics Engineering program.

[2]SE350 is the course number for Operating Systems in the University of Waterloo Software Engineering Program.

We are grateful to teaching assistants Bo Zhu, Dr. Nabil Drawil, Pei Wang, Shasha Zhu, Zheng Wu and Zack Newsham who provided valuable feedback and improved lab tutorials.

Dr. Ajit Singh, Dr. Rodolfo Pellizzoni, and Douglas W. Harder in the Electrical and Computer Engineering department have provided valuable laboratory project improvement feedback. Dr. Pellizzoni proof-read the entire manual meticulously. Douglas W. Harder also carefully proof-read descriptions of each lab. We warmly acknowledge their contributions.

The project and manual won't be possible without lab facilities. Thank Roger Sanderson for providing us with lab tools and resources. Our gratitude also goes out to Eric Praetzel who sets up the Keil boards in lab and maintains the Keil software on Nexus machines; Laura Winger who managed to customize the boards so that we have the neat plastic cover to protect our hardware. We appreciate that Bernie Roehl and Rasoul Keshavarzi-Valdani have shared their valuable Keil board experiences with us. Bob Boy from ARM always answers our questions in a detailed and timely manner. Thank everyone who has helped.

Part I

Lab Administration

Lab Administration Policy

Group Lab Policy

- **Group Size.** All labs are done in a group of *two*. A size of three is only considered in a lab section that has an odd number of students and only one group is allowed to have a size of three. All group of three requests are processed on a first-come first-served basis. A group size of one is not permitted except that your group is split up. There is no workload reduction if you do the labs individually. Everyone in the group normally gets the same mark. The Learn at URL `http://learn.uwaterloo.ca` is used to signup for groups. *The lab group signup is due by 4:30pm on the First Friday of the academic term.* Late group sign-up incurs a 5% per day final lab mark deduction.

- **Group Split-up.** If you notice workload imbalance, try to solve it as soon as possible within your group or split-up the group as the last resort. Group split-up is only allowed once. You are allowed to join a one member group after the split-up. But you are not allowed to split up from the newly formed group again. There is one grace day deduction penalty to be applied to each member in the old group. We highly recommend everyone to stay with your group members as much as possible, for the ability to do team work will be an important skill in your future career. Please choose your lab partners carefully. A copy of the code and documentation completed before the group split-up will be given to each individual in the group.

- **Group Split-up Deadline.** To split from your group for a particular lab, you need to notify the lab instructor in writing and sign the group slip-up form (see Appendix). Lab*n* (*n=1,2,3,4*) group split-up form needs to be submitted to the lab instructor by 4:30pm Thursday in the week that Lab*n* has scheduled lab sessions. If you are late to submit the split-up form, then you need to finish Lab*n* as a group and submit your split-up form during the week where Lab(*n+1*) has scheduled sessions and split starting from Lab(*n+1*).

Deliverable	Weight	Due Date	File Name
Group Sign-up		4:30pm May. 9th	
LAB1	2%	11:59pm May. 13th	LAB1_Gid.zip
LAB2	2%	11:59pm May. 27th	LAB2_Gid.zip
LAB3	8%	11:59am Jun. 26th	LAB3_Gid.zip
LAB4	4%	11:59pm Jul. 8th	LAB4_Gid.zip
LAB5	6%	11:59pm Jul. 22nd	LAB5_Gid.zip

Table 1: Project Deliverable Weight of the Course Grade and Deadlines. Replace the “id” in “Gid” with the two digit group ID

Lab Assignments Grading and Deadline Policy

Labs are graded by lab TAs based on the rubric listed in each lab. The weight of each lab towards your final course grade is listed in Table 1.

- **Lab Assignment Preparation and Due Dates.** Students are required to prepare the lab well before they come to the schedule lab session. *Pre-lab deliverable for each lab is due before the scheduled lab session starts.* During the scheduled lab session, we either provide in lab help or conduct lab assignment evaluation or do both at the same time. Table 1 lists the Post-lab deliverable deadlines. *Please be advised that lab sessions during the midterm week are cancelled.*
- **Lab Assignment Late Submissions.** There are five grace days [3] that can be used for some Post-lab deliverables late submissions. A Post-lab deliverable that does not accept a late submission will be clearly stated in the lab assignment description. Normally grace days are for lab reports. Labs whose evaluation involves demonstrations do not accept late submissions of the code. A group split-up will consume one grace day. After all grace days are consumed, your late submission will automatically get a grade of zero.

Lab Assignments Solution Internet Policy

It is not permitted to post your lab assignment solution source code or lab report on the internet freely for public to access. For example, it is not acceptable to host a public repository on GitHub that contains your lab assignment solutions. A warning with instructions

[3]Grace days are calendar days. Days in weekends are counted.

to take the lab assignment solutions off the internet will be sent out upon the first offence. If the no action is taken from the offender within twenty-four hours, then a lab grade zero will automatically be assigned to the offender.

Seeking Help Outside Scheduled Lab Hours

- **Discussion Forum.** We recommend students to use the Learn discussion forum to ask the teaching team questions instead of sending individual emails to lab teaching staff. For questions related to lab projects, our target response time is one business day before the deadline of the particular lab in question. [4]. *After the deadline, there is no guarantee on the response time.*
- **Office Hours.** During weeks where there are no scheduled labs, lab teaching staff hold bi-weekly office hours. The Learn system calendar gives the office hour details.
- **Appointments.** Students can also make appointments with lab teaching staff should their problems are not resolved by discussion forum or during office hours. When you request an appointment, please specify three preferred times and roughly how long you would like the appointment to be. On average, an appointment is fifteen minutes per project group.

Lab Facility After Hour Access Policy

After hour access to the lab will be given to the class when we start to use the Keil boards in lab. However please be advised that the after hour access is a privilege. Students are required to keep the lab equipment and furniture in good conditions to maintain this privilege.

No food or drink is allowed in the lab (water is permitted). Please be informed that you may share the lab with other classes. When resources become too tight, certain cooperation is required such as taking turns to use the stations in the lab.

[4]Our past experiences show that the number of questions spike when deadline is close. The teaching staff will not be able to guarantee one business day response time when workload is above average, though we always try our best to provide timely response.

Part II

Lab Projects

Chapter 1

Lab1: Introduction to Linux System Programming

1.1 Objective

This lab is to introduce the general Linux Development Environment at ECE department and basic Linux system programming procedures to students. After finishing this lab, students will have a good understanding of the following:

- How to use the `gcc` compiler on Linux.
- How to use the `make` utility on Linux.
- How to use the `ddd` debugger on Linux.
- How to read Linux manual page.
- How to write a C program to obtain file attributes.

1.2 Starter Files

The starter file is on GitHub at `http://github.com/yqh/ECE254/` under directory `lab1/starter`. It contains three sub-directories where we have example source code to help you started:

- the `cmd_arg` demonstrates how to capture command line input arguments;

- the `ls` demonstrates how to list all files under a directory and obtain owner permission and file types; and
- the `pointer` demonstrates how to use pointers to access C structure.

Using the code in the starter files is permitted and will not be considered as plagiarism.

1.3 Pre-lab Preparation

Read Chapter 6.

1.4 Warm-up Exercises

This exercise is to practice a few basic UNIX commands on Linux.

1. Use the SSH Secure Shell Client to login onto `ecelinux.uwaterloo.ca`. You are now inside the Linux shell.
2. Use the `ls` command to list all files in your current working directory.
3. Read the online manual of the `ls` command by issuing `man ls` command to the shell.
   - Which option of `ls` produces output in long listing format, which includes the file type, permission, ownership, group ownership, size, time of the last modification and file name?
   - What does `-a` option do?
4. Create a directory as the work space of ECE254 labs. Name the newly created directory as `ece254`. Read the man page of the command `mkdir` to see how to do it.
5. Change directory to the newly create directory of `ece254`. The command is `cd`. Read the man page to find out the exact syntax of the command.
6. Clone the entire repository by using the command:
   ```
   git clone https://github.com/yqh/ECE254.git
   ```

1.5 Lab1 Assignment

This assignment is to program a simple `ls` command. Write a program in C to list the following attributes of all files in the a given directory.

1. File name, mode (type and permissions for owner, group and other) and size
2. File access time, last modify time and last status change time
3. File ownership and group ownership

   Hints

   - From the OS point of view, a directory is a file. But it is a special file. Read the man page of `opendir()`, `readdir()`, `closedir()`. Can `open()/close()` system call open/close a directory? Can you read a directory file by the `read()` system call? Read the man page of `perror()`, which prints a system error message. Use `perror()` to print out system error when a call fails is a good programming practice.
   - The `stat()`, `fstat()` and `lstat()` system calls can be used to obtain file attribute information. Read the man page (section 2) of these system calls. Which one of these system calls gives information about a symbolic link file?
   - Execute the Linux "`ls -alt`" and "`stat <filename>`" commands and see the output.
   - Study `ls_fname.c`, `ls_ftype.c` and `ls_fperm.c` files in the `lab0_linux.zip`. The example files show how command line arguments are passed to the `main` function and how to interpret the `st_mode` value in the system defined `struct stat` by using system defined macros.
   - Study the man pages of `strcmp()`, `strcat()`, `strlen()`, `getpwuid()`, `getgrgid()` and `ctime()`.

1.6 Deliverables

1.6.1 Pre-lab Deliverables

There is no pre-lab deliverable.

1.6.2 Post-lab Deliverables

Your Lab assignment (see Section 1.5) source code, Makefile and a README (including build instruction) in a singled compressed archive and name it `lab1_linux.`*ext*, where *ext* can be `zip`, `gz` or `Z`. Submit to the Learn Dropbox.

1.7 Marking Rubric

TA will test your source code on one of the ecelinux machines. Total mark is 10. You will get full mark is all required file attributes are displayed correctly. Partial mark will be given based on the portion of the correctly displayed file attributes if not all required attributes are displayed correctly.

Chapter 2

Lab2: Introduction to ARM RL-RTX Kernel and Application Programming

2.1 Objective

This Lab is to introduce the Keil μ`Vision4` IDE and ARM RL-RTX development. Students will build the ARM RL-RTX from source. After this lab, students will have a good understanding of the following:

- How to build an RL-RTX library from source;
- How to create an application with self-built RTX Library;
- How to use SVC as the gateway to program OS functions.

Chapter 3

Lab3: Task Management in RL-RTX

3.1 Objective

This lab is to learn about, and gain practical experience in ARM RL-RTX kernel programming. In particular, you will add three functions to ARM RL-RTX library.

After this lab, students will have a good understanding of:

- how to program an RTX function to read kernel task control block related data structure;
- how to block and unblock a task by using context switching related kernel functions.

Chapter 4

Lab4: Inter-process Communication by Message Passing

4.1 Objective

This lab is to learn about, and gain practical experience in message passing for the purpose of inter-process communication. In particular, you will use the POSIX message queue facility in a general Linux environment and the mailbox APIs in RL-RTX on the Keil LPC1768 board.

After this lab, students will have a good understanding of, and ability to program with

- the `fork()` and `exec()` system calls, and their use for creating a new child process on the Linux platform;
- the `wait()` family system calls, and their use to obtain the status-change information of a child process;
- the POSIX message queue facility (`<mqueue.h>`) on the Linux platform for inter-process communication; and
- the use of mailbox APIs in RL-RTX for inter-task communication.

Chapter 5

Lab5: Thread Concurrency Control

5.1 Objective

This lab is to learn about, and gain practical experience in concurrency control for the purpose of inter-thread communication by shared memory and compare its performance with inter-process/task communication by message passing. In particular, you will use the POSIX pthread library in a general Linux environment and RL-RTX on the Keil LPC1768 board.

Sharing data among threads in a general Linux environment is straight forward because threads share the same memory [2]. For the Keil RL-RTX tasks, they share the same addressing space, hence behave similar as threads.

The challenging part of sharing memory is to avoid race conditions. In a general Linux environment, pthread semaphore and mutex library calls are to be used for thread mutual exclusion. In RL-RTX, semaphore and mutex management functions are to be used for task communication mutual exclusion.

After this lab, students will have a good understanding of, and ability to program with

- the pthread `pthread_create()` and `pthread_join()` to create and join threads,
- the pthread `sem_int()`, `sem_post()` and `sem_wait()` library calls for inter-thread communication concurrency control in a general Linux environment, and
- the use of semaphore or mutex management APIs in RL-RTX for inter-task communication concurrency control.

Part III

Development Environment Quick Reference Guide

Chapter 6

Introduction to ECE Linux Programming Environment

6.1 Linux Hardware Environment

There are ten Linux servers that are open to ECE undergraduate students. They are `ecelinux1.uwaterloo.ca` - `ecelinux4.uwaterloo.ca` and `ecelinux6.uwaterloo.ca` - `ecelinux11.uwaterloo.ca`. The `ecelinux1-4` can be accessed off-campus as well as on campus. The rest of these machines are only accessible on-campus and consoles are located at E5-5038 whose door code is printed in the welcome message when you login onto any one of the Linux workstations. Linux CentOS is installed on all these machines.

6.2 How to Connect to Linux Servers

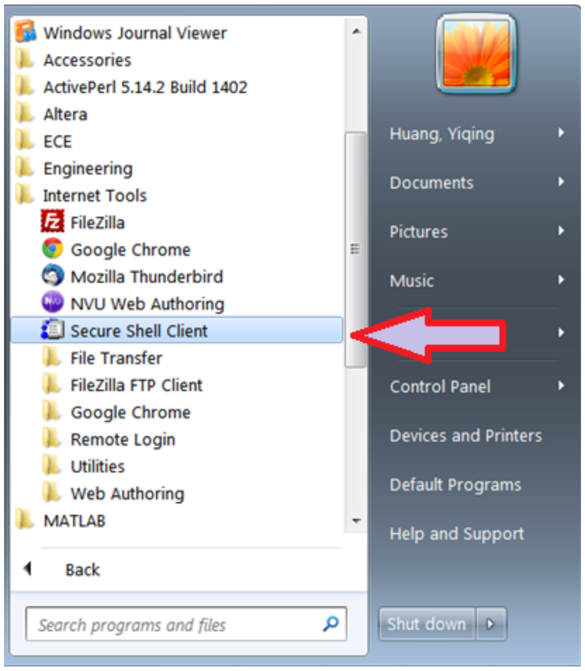
The servers are accessed by remote login. You will need to have a terminal client that supports secure shell (ssh) installed before you can log onto one of the ecelinux servers. Two popular terminal clients are:

- Windows secure shell client and
- PuTTY .

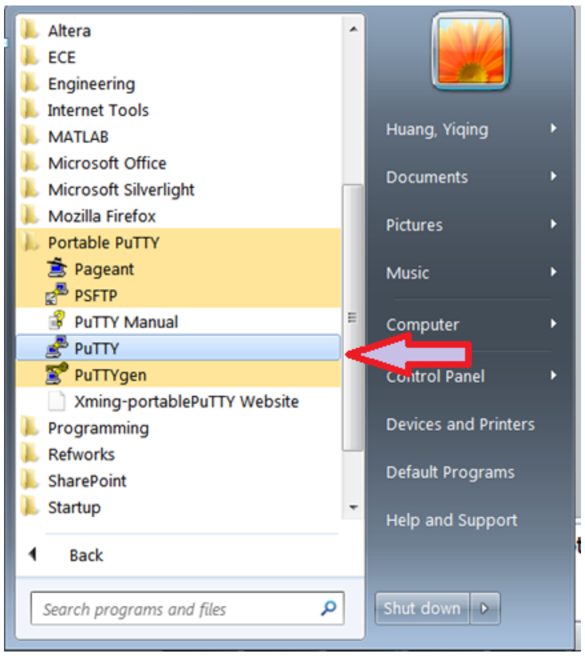
Both terminal clients are installed on ECE Nexus machines. Use your WatIAM credential to login onto these machines.

To use the SSH Secure shell windows client, click Start → All Programs → Internet Tools → **Secure shell client**. Figure 6.1(a) is a screen shot taken on a Nexus computer.

To use the PuTTY, click Start → All Programs → Portable PuTTY → **PuTTY**. Figure 6.1(b) is a screen shot taken on a Nexus computer.



(a) SSH Secure Shell Client



(b) PuTTY

Figure 6.1: Invoking Terminal Clients on an ECE Nexus PC

6.3 Work Environment Setup

6.3.1 Setting up Remote Linux Graphic Support

After you login, you will notice that you are in a command line shell environment, one where GUI applications cannot be run. For example the `ddd` debugger is an important GUI application you will most likely want to use. You will need to configure your environment to support GUI application to be able to display graphics on your terminal.

First start the X server on your local machine. Xming is a popular and free X server which is installed on ECE Nexus computers. Start Xming by clicking All Programs → Xming (see Figure 6.2).

The second step is to configure the terminal client so that X11 forwarding is enabled.

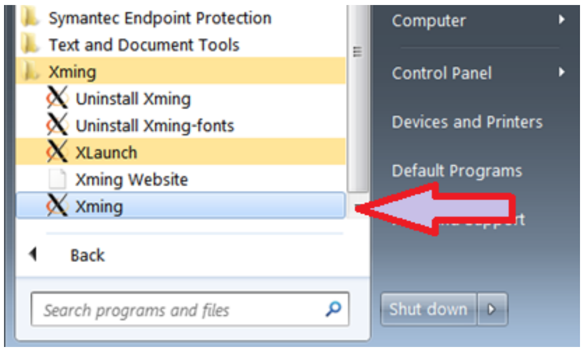
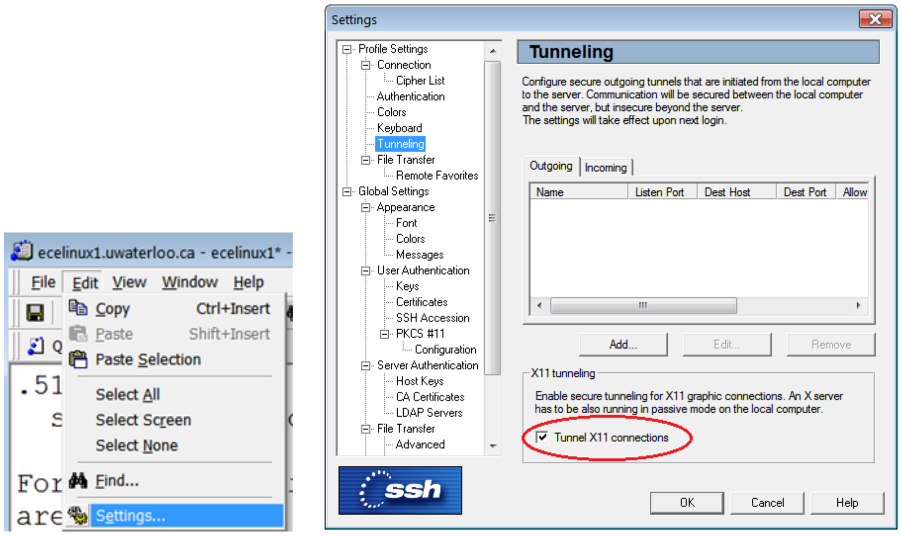


Figure 6.2: Invoking Xming on an ECE Nexus Computer



Configure secure outgoing tunnels that are initiated from the local computer to the server. Communication will be secured between the local computer and the server, but insecure beyond the server.
The settings will take effect upon next login.

Enable secure tunneling for X11 graphic connections. An X server has to be also running in passive mode on the local computer.

[✓] Tunnel X11 connections

(a) Setting (b) X11 Tunneling Setting

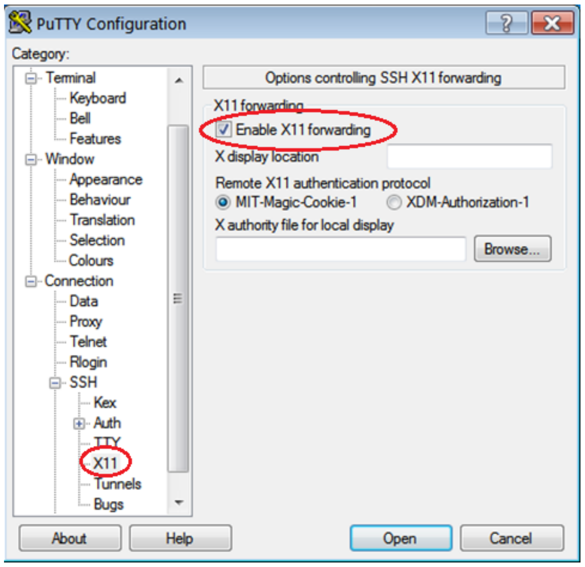
Figure 6.3: SSH Secure Shell Client X11 Setting

For SSH secure shell client, select Edit → Settings to bring up the setting dialog window (see Figure 6.3(a)). Go to Profile Settings → Tunneling and put a check mark beside the Tunnel X11 connection item (see Figure 6.3(b)).

For PuTTY, go to Connection → SSH → X11 and put a check mark beside the Enable X11 forwarding item (see Figure 6.4).

6.3.2 Mapping Linux Account on Nexus

Your ECE Linux files can be access through network drive mapping on Nexus machines. Open My Computer → Tools → Map Network Drive (see Figure 6.5(a)). Under Driver,



Options controlling SSH X11 forwarding

X11 forwarding

[✓] Enable X11 forwarding

X display location

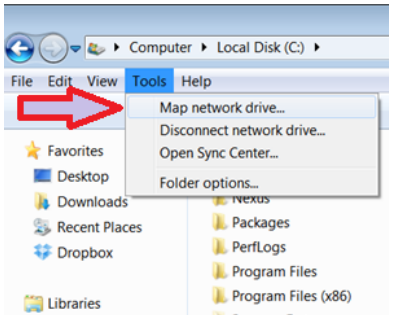
Remote X11 authentication protocol

(●) MIT-Magic-Cookie-1 ( ) XDM-Authorization-1

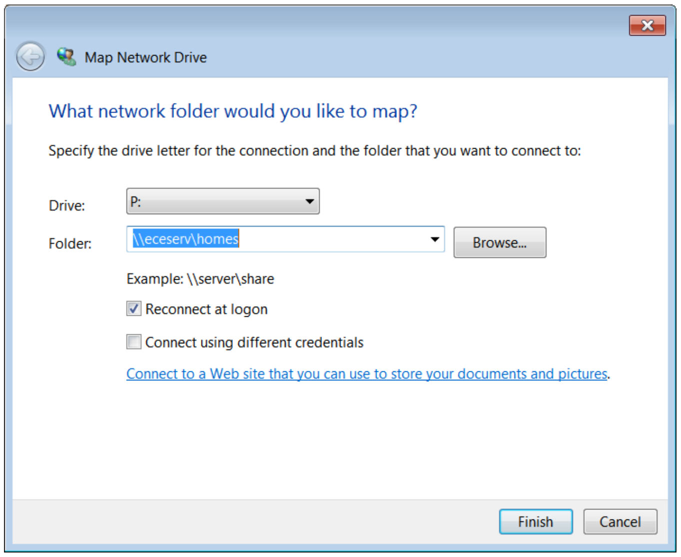
X authority file for local display

Figure 6.4: PuTTY X11 Forwarding Setting

pick a drive letter, say P. Under Folder, type `\\eceserv\homes`. Put a check mark beside Reconnect at logon item and click Finish (see Figure 6.5(b)). You will use your WatIAM credential to authenticate yourself.



(a) Prepare to Map a Network Drive



What network folder would you like to map?

Specify the drive letter for the connection and the folder that you want to connect to:

Drive: P:

Folder: \\eceserv\homes

Example: \\server\share

[✓] Reconnect at logon

[ ] Connect using different credentials

Connect to a Web site that you can use to store your documents and pictures.

(b) Configuring Network Drive Mapping

Figure 6.5: SSH Secure Shell Client X11 Setting

6.4 Basic Software Development Tools

To develop a program, there are three important steps. First, a program is started from source code written by programmers. Second, the source code is then compiled into object code, which is a binary. Non-trivial project normally contains more than one source file. Each source file is compiled into one object code and the linker would finally link all the object code to generate the final target, which is the executable that runs. People refer to compiling and linking as building a target. It is very rare that the target will run perfectly the first time it is built. Most of time you need to fix defects and bugs in your code and the this is the third step. The debugger is a tool to help you identify the bug and fix it. Table 6.1 shows the key steps in programming work flow, the corresponding tools are needed and some example tools provided by a general purpose Linux operating system.

Task	Tool	Examples
Editing the source code	Editor	vi, emacs
Compiling the source code	Compiler	gcc
Debugging the program	Debugger	gdb, ddd

Table 6.1: Programming Steps and Tools

At each development step, you will have a choice of tools to get the work done. Most of you probably are more familiar with a certain Integrated Development Environment (IDE) which integrates all these tools into a single environment. For example Eclipse and Visual Studio. However another choice is that you pick your favorite tool in each programming step and build your own tool chain. Many seasoned Linux programmers build their own tool chains. A few popular tools are introduced in the following subsections.

6.4.1 Editor

Some editors are designed to better suit programmers' needs than others. The *vi* (*vim* and *gvim* belong to the vi family) and *emacs* (*xemacs* belongs to emacs family) are the two most popular editors for programming purposes.

Two simple notepad editors *pico* and *nano* are also available for a simple editing job. These editors are not designed for serious programming activity. To use one of them to write your first *Hello World* program is fine though.

After you finish editing the C source code, give the file name an extension of .c. Listing 6.1 is the source code of printing "Hello World!" to the screen.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
  printf("Hello World!\n");
  exit(0);
}
```

Listing 6.1: HelloWorld C source Code

Next we will compile and execute the program.

6.4.2 C Compiler

The executable*gcc* is the GNU project C and C++ compiler. To compile the HelloWorld source code in Listing 6.1, type the following command at the prompt:

```
gcc helloworld.c
```

You will notice that a new file named `a.out` is generated. This is the executable generated from the source code. To run it, type the following command at the prompt and hit Enter.

```
./a.out
```

The result is“Hello World!” appearing on the screen.

You can also instruct the compiler to name the executable another name instead of the default `a.out`. The `-o` option in gcc allows one to name the executable a name. For example, the following command will generate an executable named “`helloworld.out`”.

```
gcc helloworld.c -o helloworld.out
```

although there is no requirement that the name ends in `.out`.

6.4.3 Debugger

The GNU debugger gdb is a command line debugger. Many GUI debugger uses gdb as the back-end engine. One GNU GUI debugger is *ddd*. It has a powerful data display functionality.

GDB needs to read debugging information from the binary in order to be able to help one to debug the code. The `-g` option in gcc tells the compiler to produce such debugging information in the generated executable. In order to use gdb to debug our simple HelloWorld program, we need to compile it with the following command:

```
gcc -g helloworld.c -o helloworld.out
```

The following command calls gdb to debug the helloworld.out

```
gdb helloworld.out
```

This starts a gdb session. At the `(gdb)` prompt, you can issue gdb command such as `b main` to set up a break point at the entry point of main function. The `l` lists source code. The `n` steps to the next statement in the same function. The `s` steps into a function. The `p` prints a variable value provided you supply the name of the variable. Type `h` to see more gdb commands.

Compared to gdb command line interface, the ddd GUI interface is more user friendly and easy to use. To start a ddd session, type the command

```
ddd
```

and click File → Open Program to open an executable such as helloworld.out. You will then see gdb console in the bottom window with the source window on top of the gdb console window. You could see the value of variables of the program through the data window, which is on top of the source code window. Select View → to toggle all these three windows.

6.5 More on Development Tools

For any non-trivial software project, it normally contains multiple source code files. Developers need tools to manage the project build process. Also project normally are done by several developers. A version control tool is also needed.

6.5.1 How to Automate Build

Make is an utility to automate the build process. Compilation is a cpu-intensive job and one only wants to re-compile the file that has been changed when you build a target instead of re-compile all source file regardless. The `make` utility uses a Makefile to specify the dependency of object files and automatically recompile files that has been modified after the last target is built.

In a Makefile, one specifies the targets to be built, what prerequisites the target depends on and what commands are used to build the target given these prerequisites. These are the *rules* contained in Makefile. The Makefile has its own syntax and is a good reference on Make. The general form of a Makefile rule is:

```
target ...: prerequisites ...
        recipe
        ...
        ...
```

One important note is that each recipe line starts with a TAB key rather than white spaces.

Listing 6.2 is our first attempt to write a very simple Makefile.

```
helloworld.out: helloworld.c
        gcc -o helloworld.out helloworld.c
```

Listing 6.2: Hello World Makefile: First Attempt

Our second attempt is to break the single line gcc command into two steps. First is to *compile* the source code into object code .o file. Second is to *link* the object code to one final executable binary. Listing 6.3 is our second attempted version of Makefile.

```
helloworld.out: helloworld.o
        gcc -o helloworld.out helloworld.o

helloworld.o: helloworld.c
        gcc -c helloworld.c
```

Listing 6.3: Hello World Makefile: Second Attempt

When a project contains multiple files, separating object code compilation and linking stages would give a clear dependency relationship among code. Assume that we now need to build a project that contains two source files `src1.c` and `src2.c` and we want the final executable to be named as `app.out`. Listing 6.4 is a typical example Makefile that is closer to what you will see in the real world.

```
all: app.out

app.out: src1.o src2.o
        gcc -o app.out src1.o src2.o

src1.o: src1.c
        gcc -c src1.c
```

```
src2.o: src2.c
        gcc -c src2.c

clean:
        rm *.o app.out
```

Listing 6.4: A More Real Makefile: First Attempt

We also have added a target named *clean* so that `make clean` will clean the build.

So far we have seen the Makefile contains *explicit rules*. Makefile can also contain *implicit rules*, *variable definitions*, *directives* and *comments*. Listing 6.5 is a Makefile that is used in the real world.

```
 1 # Makefile to build app.out
 2 CC=gcc
 3 CFLAGS=-Wall -g
 4 LD=gcc
 5 LDFLAGS=-g

 7 OBJS=src1.o src2.o

 9 all: app.out
10 app.out: $(OBJS)
11         $(LD) $(CFLAGS) $(LDFLAGS) -o $@ $(OBJS)
12 .c.o:
13         $(CC) $(CFLAGS) -c $<
14 .PHONY: clean
15 clean:
16         rm -f *.o *.out
```

Listing 6.5: A Real World Makefile

Line 1 is a comment. Lines $2-7$ are variable definitions. Line 12 is an implicit rule to generate .o file for each .c file. See `http://www.gnu.org/software/make/manual/make.html` if you want to explore more of makefile.

6.5.2 Version Control Software

ECE Linux has the following version control software installed.

- CVS
- SVN
- Git

Choose your favorite one. Any one of them will be able to help you manage ECE254 code repository. Git is getting more and more popularity these days. If you decides to use GitHub to host your repository, please make sure it is a private one. Go to `http://github.com/edu` to see how to obtain five private repositories for two years on GitHub.

6.5.3 Integrated Development Environment

Eclipse with C/C++ Plug-in has been installed on all ECE Linux servers. You will need to first set up the X Window support properly (see section 6.3.1, then type the following command to bring up the eclipse frontend.

```
/opt/eclipse64/eclipse
```

This eclipse is not the same as the default eclipse under `/usr/bin` directory. You may find running eclipse over network performs poorly at home though. It depends on how fast your network speed is.

If you have Linux operating system installed on your own personal computer, then you can download the eclipse with C/C++ plugin from the eclipse web site and then run it from your own local computer. However you should always make sure the program will also work on ecelinux machines, which is the environment TAs would be using to test your code.

6.6 Man Page

Linux provides manual pages. You can use the command `man` followed by the specific command or function you are interested in to obtain detailed information.

Mange pages are grouped into sections. We list frequently used sections here:

- Section 1 contains user commands.
- Section 2 contains system calls
- Section 3 contains library functions
- Section 7 covers conventions and miscellany.

To specify which section you want to see, provide the section number after the `man` command. For example,

```
    man 2 stat
```

shows the system call `stat` man page. If you omit the `2` in the command, then it will return the command `stat` man page.

You can also use `man -k` or `apropos` followed by a string to obtain a list of man pages that contain the string. The Whatis database is searched and now run `man whatis` to see more details of `Whatis`.

Appendix A

Forms

Lab administration related forms are given in this appendix.

ECE254/MTE241 Request to Leave a Project Group Form

Name:	
Quest ID:	
Student ID:	
Lab Assignment ID	
Group ID:	
Name of Other Group Members:	

Provide the reason for leaving the project group here:

Signature ______________________ Date ______________________

Bibliography

[1] AD Marshall. Programming in c unix system calls and subroutines using c. *Available on-line at http://www.cs.cf.ac.uk/Dave/C/CE.html*, 1999.

[2] M. Mitchell, J. Oldham, and A. Samuel. Advanced linux programming. *Available on-line at http://advancedlinuxprogramming.com*, 2001. 13

[3] W. Stallings. *Operating systems: internals and design principles.* Prentice Hall, seventh edition, 2011.