Politechnika Świętokrzyska
Kielce University of Technology

Operating systems 1

Introduction to the vim editor, gcc compiler, **make** program, **man** manual and the **Linux** shell

Tanmay Nandanikar | Lab report 1 | 10/9/20

dr inz Karol Tomaszewski

INTRODUCTION

This lab primarily focused on common Linux shell commands, a brief introduction to the VIM text editor, and using the gcc compiler.

1 BRIEF OVERVIEW OF THEORY

1.1 LINUX SHELL COMMANDS

Text interpreters on linux and other unix based systems are called shells. We learnt a few shell commands for essential tasks such as creating a directory (mkdir *name*), changing shell directory (cd *name*), deleting a directory (rmdir *name*), deleting a file (rm *name*), display system manual pages (man *command_name*), etc.

1.2 VIM EDITOR

Vim has various modes. The command mode, which is the default, is used to input various vim commands. The insert and replace modes can be entered through commands from the command mode. In insertion mode new characters are added to existing ones and in replacement mode they replace existing characters. Vim can allow users to quickly perform tasks such as formatting long lists (file names of a tv series for example) to look more orderly if they are proficient at using it. It can just as easily be used as a normal text editor if one so wishes.

1.3 THE GCC COMPILER AND THE MAKE COMMAND

The gcc compiler can be used through the terminal to compile a .c file. A make program allows you to automate the compilation process which proves necessary for large programs containing several files. A configuration file called a makefile is used to describe the compilation rules. The gdb debugger can be used if the -g flag is used during compilation.

2 EXERCISES

2.1 PRINTING NUMBERS FROM 1 TO 100 USING A VIM MACRO

To demonstrate understanding of vim, I used a macro to print numbers from 1 to 100 in vim. The commands used can be found in the source section (here). I could not include the actual 'code' since I only used commands within the text editor.

2.1.1 A snippet of part of the output

```
1
2
3
4
5
6
7
8
9
10
.
.
```

2.2 HELLO WORLD

Source code for hello world can be found in the source section (here). I did not encounter any warnings while debugging.

2.2.1 Compilation and output in terminal

```
sprices@pop-os:~/Documents/Coding/Operating systems$ vim hello.c

sprices@pop-os:~/Documents/Coding/Operating systems$ gcc -Wall -o hello
hello.c

sprices@pop-os:~/Documents/Coding/Operating systems$ ./hello

Hello World!!!
```

2.3 MAKEFILE USING CUSTOM RULES

I made a program containing a C file and a header file. One containing a function to print 'Hello again, world!', and another containing a call to that function. I compiled it using a makefile. The makefile and the C files can be found in the source section (here). The 'make exec' command wasn't working as intended.

2.3.1 make command and output in terminal

```
sprices@pop-os:~/Documents/Coding/Operating systems$ make

gcc -Wall -o output1 hello1.c

sprices@pop-os:~/Documents/Coding/Operating systems$ ./output1

Hello again, world!
```

2.4 GDB DEBUGGER WITH PROGRAM CONTAINING A FOR LOOP

I tried using the GDB debugger for a program containing a for loop. I used the help function and all of the commands listed in the lab guide. The text from the help function and the C source is listed in the source section (here).

3 SOURCE

3.1 VIM COMMANDS FOR EXERCISE ONE

```
qw to register the macro

yy to copy '1'

insert, down arrow key, Esc to go to the next line and enter command
mode again

P to paste

Ctrl+A to increment

q to stop recording

99@w  to run macro 99 times
```

3.2 SOURCE CODE FOR EXERCISE TWO

```
#include<stdio.h>


int main(){

printf("Hello World!!!\n");

return 0;

}
```

3.3 SOURCE CODE FOR EXERCISE THREE

Makefile

```
#this is a makefile
CC = gcc
CFLAGS = -Wall
program: hello1.c
	$(CC) $(CFLAGS) -o output1 hello1.c
exec:
	./output1
clean:
	rm -f output1
```

hello.c

```
#include "hello1.h"

int main(){
hellofn();
return 0;
}
```

hello.h

```
#include<stdio.h>

hellofn(){
printf("Hello again, world!");
}
```

3.4 SOURCE FILE TEXT FOR EXERCISE FOUR

```
C source
#include<stdio.h>

void loopthis(){
      int i;
      for(i=0;i<100;i++){
            printf("%d \n",i);
      }
}

int main(){
loopthis();
return 0;
}
Help command
List of classes of commands:

aliases -- Aliases of other commands.
breakpoints -- Making program stop at certain points.
data -- Examining data.
files -- Specifying and examining files.
internals -- Maintenance commands.
obscure -- Obscure features.
running -- Running the program.
stack -- Examining the stack.
status -- Status inquiries.
support -- Support facilities.
tracepoints -- Tracing of program execution without stopping the
program.
user-defined -- User-defined commands.

Type "help" followed by a class name for a list of commands in that
class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Type "apropos -v word" for full documentation of commands related to
"word".
Command name abbreviations are allowed if unambiguous.
```