

OS TUTORIAL 1

Specification go-through, design hints &
system calls

Zhiming Huang



About Me

- **Tutorial Instructor:**
 - Zhiming Huang, PhD candidate
 - Department of Computer Science, University of Victoria
- **How to reach me?**
 - Teams



About the Course Tutorial (1)

- What **I** can do for you:
 - Help you **understand** the assignments
 - Provide **required knowledge** to complete the assignments
 - Give **hints/tips** at key points of assignments
- It is **your** responsibility to:
 - Prepare solution codes of assignments
 - Debug your programs
 - Pay attention to the due time



About the Course Tutorial (2)

Date	Tutorial	Milestones by the Friday that week
Sept 12/13/15	T1: P1 spec go-thru, design hints, system calls	design and code skeleton
Sept 19/20/22	T2: code design & implementation	code design/implementation done
Sept 26/27/29	T3: code implementation & testing; submission issues	Code implementation/testing done



Outline

- Linux Basic and C Programming Basic
- Specification of Assignment 1 go-through
- Design hints & system calls



Linux Basic (1)

- **Local machine:**
 - Your laptop with Linux OS or VM in Windows
- **Remote access via SSH**
 - Evaluation will be done on `linux.csc.uvic.ca`
 - Mac OS X and Linux users:
 - `ssh NetlinkID@linux.csc.uvic.ca`
 - `ssh -l NetlinkID linux.csc.uvic.ca`
 - Windows:
 - PuTTY, MobaXterm, etc.
- **More on <https://itsupport.cs.uvic.ca/>**



Linux Basic (2)

- Remote copy file:
 - SCP:
 - Copy files/folders from local to host:
 - `scp local_file remote_username@remote_ip:remote_folder`
 - `scp -r local_folder remote_username@remote_ip:remote_folder`
 - Copy files from host to local:
 - Change the order of the arguments in the above commands
 - SFTP
 - `SFTP NetlinkID@linux.csc.uvic.ca`
 - `cd [pathname]`
 - `put [user@]SRC_HOST:]file1`



Linux Basic (2) - Setup

Compress your coding assignments using **tar** before submitting to Brightspace:

- Command to create an archive:

```
$ tar -czvf filename.tar.gz path
```

- Command to decompress from an archive:

```
$ tar -zxvf filename.tar.gz
```



Linux Basic (3) - Basic Command Line Operations

- **man**: system's manual pager, useful for referencing help/manual pages
- **ls**: list directory contents
- **pwd**: print name of current/working directory
- **cd**: change directory
- **cp**: copy files and directories
- **mv**: move (rename) files
- **mkdir**: make directories
- **rmdir**: remove empty directories
- **rm**: remove files or directories
- **chmod**: change file mode bits, permissions
- **chown**: change file owner and group

and many more...



When reading a Linux manual, you often encounter paragraphs like this:

*The child process and the parent process run in separate memory spaces. At the time of **fork()** both memory spaces have the same content. Memory writes, file mappings (**mmap(2)**), and unmappings (**munmap(2)**) performed by one of the processes do not affect the other.*

*After a **fork()** in a multithreaded program, the child can safely call only async-signal-safe functions (see **signal-safety(7)**) until such time as it calls **execve(2)**.*

What does these numbers mean?

<https://man7.org/linux/man-pages/man2/fork.2.html>



University
of Victoria

Linux Basic (4) - Manual Pages

\$ man [section] page...

- Each page argument given to **man** is normally the name of a program or function. (e.g., *ssh(1)*, *printf(3)*)
- The manual page associated with each of these arguments is then found and displayed. (It can accept multiple page argument)
- A section, if provided, will direct **man** to look only in that section of the manual.
- The default action is to search in all of the available sections following a pre-defined order, and to show only the first page found, even if page exists in several sections. (e.g., *printf(1)*, *printf(3)*)



`$ man [section] page`

Also check out **man man**

Section number

- 1 Executable programs or shell commands (e.g., `ls`, `ssh`...)
- 2 System calls (functions provided by the kernel, e.g., `fork`, `waitpid`)
- 3 Library calls (functions within program libraries, e.g., `printf`)
- 4 Special files (usually found in `/dev`)
- 5 File formats and conventions, e.g., `/etc/passwd`
- 6 Games
- 7 Miscellaneous (including macro packages and conventions)
- 8 System administration commands (usually only for root)
- 9 Kernel routines [Nonstandard]

Examples

`man 1 ls`
`man 2 fork`
`man 3 printf`

Recommend reading:

man 1 intro
man 2 intro

Demo

List all the pages:

<https://www.kernel.org/doc/man-pages/>



Linux Basic (5) - Shell

Shell

We type commands to the shell, the command **interpreter**. It is just a program that you can change. Everybody has their own favorite one. The standard one is called sh(dash).

See also ash(1), **bash**(1), chsh(1), csh(1), dash(1), ksh(1), zsh(1) and list all the shells currently installed using **cat /etc/shells**

```
$ cat /etc/shells
```

```
/bin/bash  
/bin/csh  
/bin/dash  
/bin/ksh  
/bin/sh  
/bin/tcsh  
/bin/zsh
```



Why C language?

- Better control of low-level operations
- Better performance
- Other languages, like Java and Python, hide many details for OS level interaction and coding
 - Process management
 - Memory management
 - Error detection



System Call

- System call is how a program requests a service from an operating system's kernel.
- This may include hardware-related services (for example, accessing a hard disk drive), creation and execution of new processes, and communication with internal kernel services e.g. process scheduling.



Assignment Spec Go-through

Implementing a **S**imple **S**hell **I**nterpreter (**SSI**)

- Basic Execution (5 marks) *fork ()*, *execvp ()*
 - Changing Directories (5 marks) *getcwd ()*, *chdir ()*
 - Background Execution (5 marks) *bg*, *bglist*, *waitpid ()*
- *SSI shell must indicate to the user after background jobs have terminated*
- *Use GNU **Readline** Library to modify input string*



Assignment Evaluation

- **Basic Execution (5 marks)**

- Show correct prompt => (2)
- Be able to execute arbitrary commands => (3)

- **Changing Directories (5 marks)**

- `cd (~)` => (1)
- `cd ..` => (1)
- `cd path`
 - absolute path => (1.5)
 - relative path => (1.5)

- **Background Execution (5 marks)**

- `bg`, `bglist`
- 1 bg process => (2.5)
- Multi bg processes => (2.5)

➤ *SSI shell must indicate to the user after background jobs have terminated*



Structure

- loop
 - print: (use getcwd, getlogin and gethostname)
username@hostname: /home/user >
- read a line from terminal
- execute the input line by:
 - fork
 - execvp



See you next week

- **Contributors:**

- Cheng Chen
- Dawood Sajjadi
- Huan Wang
- Jingrong Wang
- Zehui Zheng
- Kaiyang Liu
- Zhiming Huang



C Programming under Linux (1)

- **What you need:** + C standard library
 - **Editor:**
 - Command line editor: vi, **vim**
 - GUI editor: **gedit**
 - **Compiler:**
 - GNU Compiler Collection (**GCC**)
 - `$gcc hello.c -o hello`
 - `$/hello`
 - **Debugger:**
 - **gdb**



C Programming under Linux (2)

- 1. **Create** and **Edit** Source Files
 - Using editors mentioned before: vim, gedit or emacs etc.
 - An example: **\$ vim hello.c**
- 2. **Compile** Single Source File
 - \$ gcc hello.c -o hello**
 - Preprocess - > compile - > assemble - > link
 - Warning info.: **\$ gcc -Wall hello.c -o hello**
- 3. **Execute** Output
 - \$./hello**



C Programming under Linux (3)

- 4. Compile **Multiple** Source Files

- \$ gcc -c main.c -o main.o

- \$ gcc -c add.c -o add.o

- \$ gcc main.o add.o -o test

- What if you have more source files?



C Programming under Linux (4)

- 5. **Makefile** for multiple source files

- Basic Syntax:

Target: [dependencies]

[TAB] <command>

<command 2> ...

- Example:

- Command:

- **\$ make**

- use **-f** to specify a Makefile : **\$ make -f myMakefile**

- Tutorials:

- <http://www.gnu.org/software/make/manual/make.html#Introduction>

- <http://mrbook.org/blog/tutorials/make/>

- <http://www.cprogramming.com/tutorial/makefiles.html>

```
test: main.o add.o
```

```
    gcc -o test main.o add.o
```

```
main.o: main.c add.h
```

```
    gcc -c main.c
```

```
add.o: add.c add.h
```

```
    gcc -c add.c
```

```
clean:
```

```
    rm -f test
```

```
    rm -f *.o
```



C Programming under Linux (4)

- 6. **Debug** Programs

- GDB:

- \$ gcc -g hello.c -o hello

- \$ gdb hello

- Official doc.:

- <http://www.gnu.org/software/gdb/documentation/>

