

Lecture Note 2.

Programming Environment

September 11, 2023

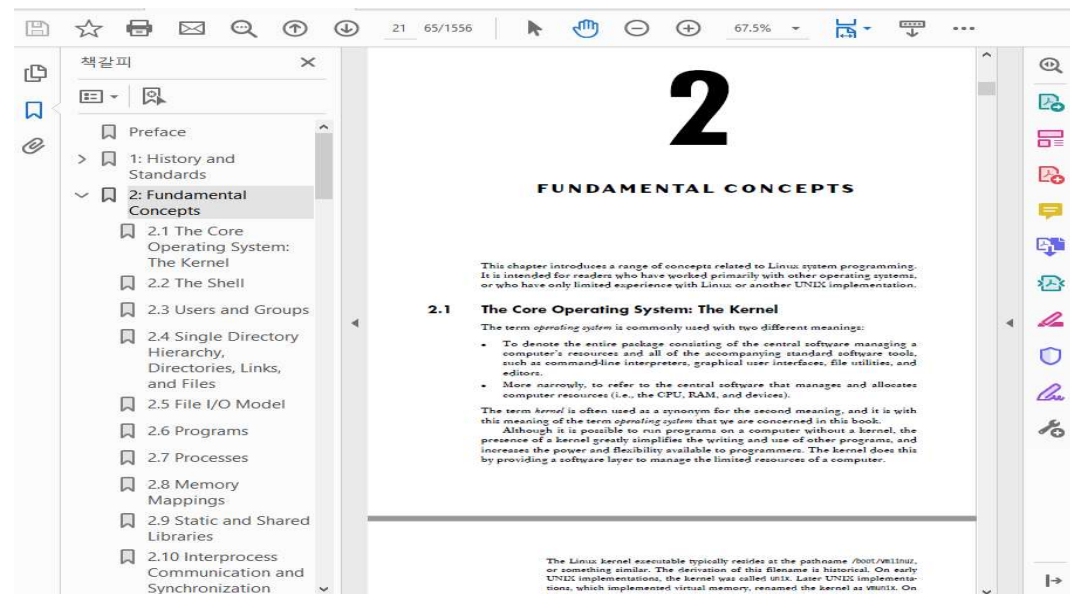
Jongmoo Choi
Dept. of Software
Dankook University

<http://embedded.dankook.ac.kr/~choijm>

(Copyright © 2023 by Jongmoo Choi, All Rights Reserved. Distribution requires permission.)

Contents

- Introduce Linux
 - Discuss fundamental concepts of Linux
 - Learn how to access Linux
 - Learn how to use commands in Linux
 - Learn how to make programs in Linux
-
- Refer to Chapter 1, 2 in the LPI



Linux Introduction (1/8)

■ Operating System

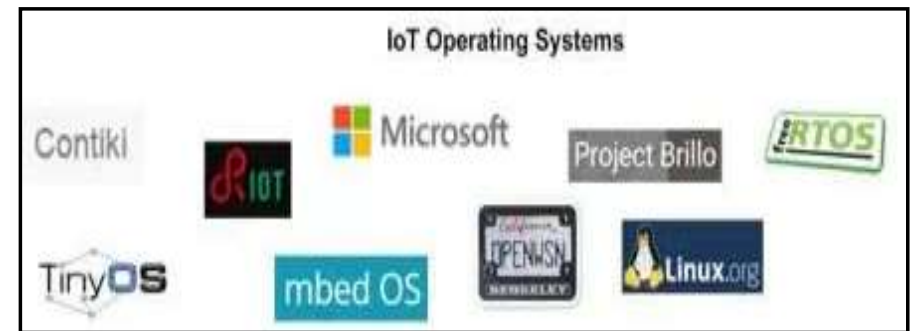
- ✓ Definition: Resource Manager
- ✓ Examples: Linux, Windows, OS X and so on.



(Source: IEEE Spectrum, 2001)



(source: <https://www.deviantart.com/nick-os/art/Os-war-choose-your-poison-110510677>)

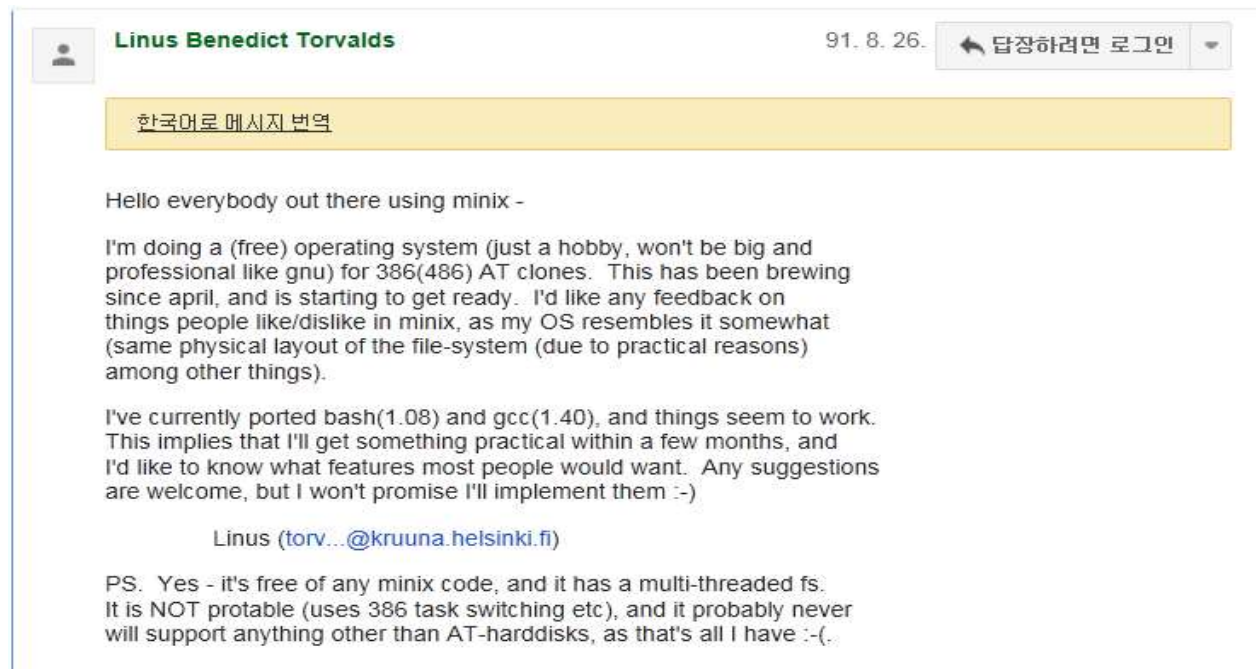


(source: <https://maxhemingway.com/2015/10/21/iot-device-security-considerations-and-security-layers-operating-system/>)

Linux Introduction (2/8)

■ Linux Definition

- ✓ Linux is a clone of the **UNIX Operating System**
- ✓ Written from scratch by **Linus B. Torvalds**, with assistance from a loosely-knit team of **Developers across the Network**



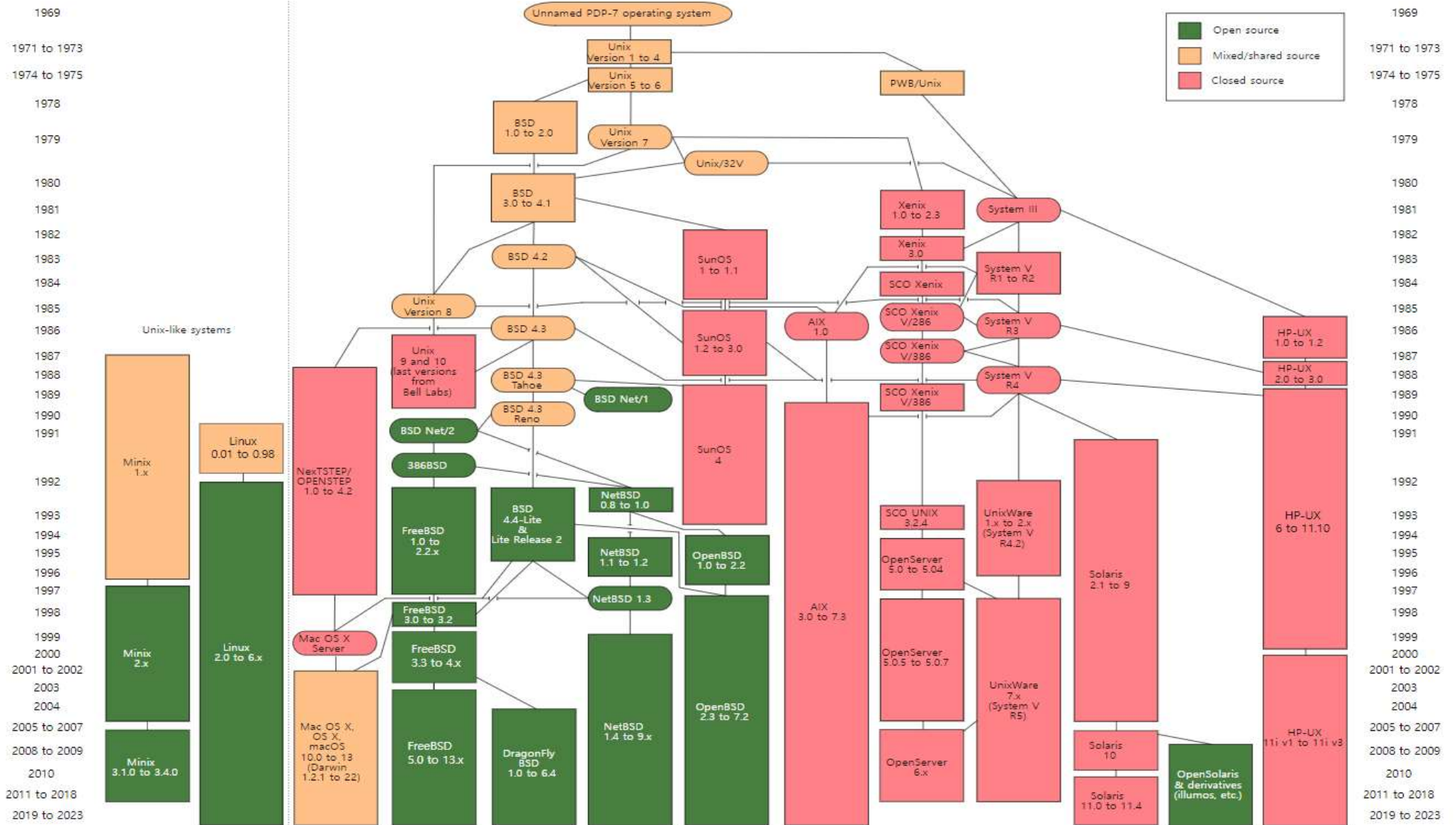
- ✓ Univ. of Helsinki in Finland
- ✓ May, 1991: Release 0.0.1 version
- ✓ 11. September, 2023: Release 6.5.2 (refer to <https://www.kernel.org/>)



Linux Introduction (3/8)

Unix-like OSes

(Source: wikipedia.org)



Linux Introduction (4/8)

■ Ken and Dennis

W Ken Thompson - Wikipedia

en.wikipedia.org/wiki/Ken_Thompson

Ken Thompson

From Wikipedia, the free encyclopedia

For other people named Ken Thompson, see Ken Thompson (disambiguation).

Kenneth Lane Thompson (born February 4, 1943) is an American pioneer of computer science. Thompson worked at Bell Labs for most of his career where he designed and implemented the original Unix operating system. He also invented the B programming language, the direct predecessor to the C programming language, and was one of the creators and early developers of the Plan 9 operating system. Since 2006, Thompson has worked at Google, where he co-developed the Go programming language.

Other notable contributions included his work on regular expressions and early computer text editors QED and ed, the definition of the UTF-8 encoding, and his work on computer chess that included the creation of endgame tablebases and the chess machine Belle. He won the Turing Award in 1983 with his long-term colleague Dennis Ritchie.

Early life and education

Thompson was born in New Orleans, Louisiana. When asked how he learned to program, Thompson stated, "I was always fascinated with logic and even in grade school I'd work on arithmetic problems in binary, stuff like that. Just because I was fascinated."^[3]

Thompson received a Bachelor of Science in 1965 and a master's degree in 1966, both in Electrical Engineering and Computer Sciences, from the University of California, Berkeley, where his master's thesis advisor was Elwyn Berlekamp.^[4]

Career and research

Thompson was hired by Bell Labs in 1966.^[5] In the 1960s at Bell Labs, Thompson and Dennis Ritchie worked on the Multics operating system. While writing Multics, Thompson created the Bon programming language.^{[6][7]} He also created a video game called *Space Travel*. Later, Bell Labs withdrew from the MULTICS project.^[8] In order to go on playing the game, Thompson found an old PDP-7 machine and rewrote *Space Travel* on it.^[9] Eventually, the tools developed by Thompson became the Unix operating system: Working on a PDP-7, a team of Bell Labs researchers led by Thompson and Ritchie, and including Rudd Canaday, developed a hierarchical file system, the concepts of computer processes and device files, a command-line interpreter, pipes for easy inter-process communication, and some small utility programs. In 1970, Brian Kernighan suggested the name "Unix", in a pun on the name "Multics".^[10] After initial work on Unix, Thompson decided that Unix needed a system programming language and created B, a precursor to Ritchie's C.^[11]

In the 1960s, Thompson also began work on regular expressions. Thompson had developed the CTSS version of the editor QED, which included regular expressions for searching text. QED and Thompson's later editor ed (the standard text editor on Unix) contributed greatly to the eventual popularity of regular

Ken Thompson

Thompson in 2019

Born Kenneth Lane Thompson
February 4, 1943 (age 80)
New Orleans, Louisiana, U.S.

Nationality American

Alma mater University of California, Berkeley (B.S., 1965; M.S., 1966)

Known for Multics
Unix
B (programming language)
Belle (chess machine)
UTF-8
Plan 9 from Bell Labs
Inferno (operating system)
grep
Endgame tablebase
Go

Awards IEEE Emanuel R. Piore Award (1982)^[1]
Turing Award (1983)
Member of the National Academy of Sciences (1985)^[2]
IEEE Richard W. Hamming Medal (1990)
Computer Pioneer Award (1994)
National Medal of Technology (1998)

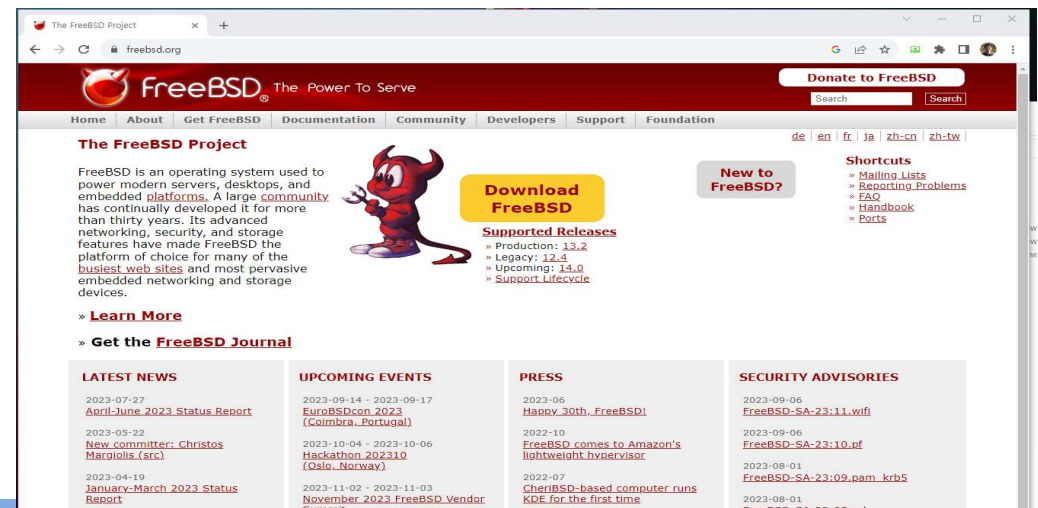
Thompson (sitting) and Ritchie working together at a PDP-11



Linux Introduction (5/8)

Contributors

- ✓ GNU (www.gnu.org)
 - Richard M. Stallman (rms)
 - Free software
- ✓ Minix
 - Andrew Tanenbaum
- ✓ BSD
 - Bill Joy (cofounder of Sun Microsystems), FFS, TCP/IP, ...
 - Linus Torvalds has said that if 386BSD had been available at the time, he probably would not have created Linux



Linux Introduction (6/8)

- Applications



(Source: images at google)



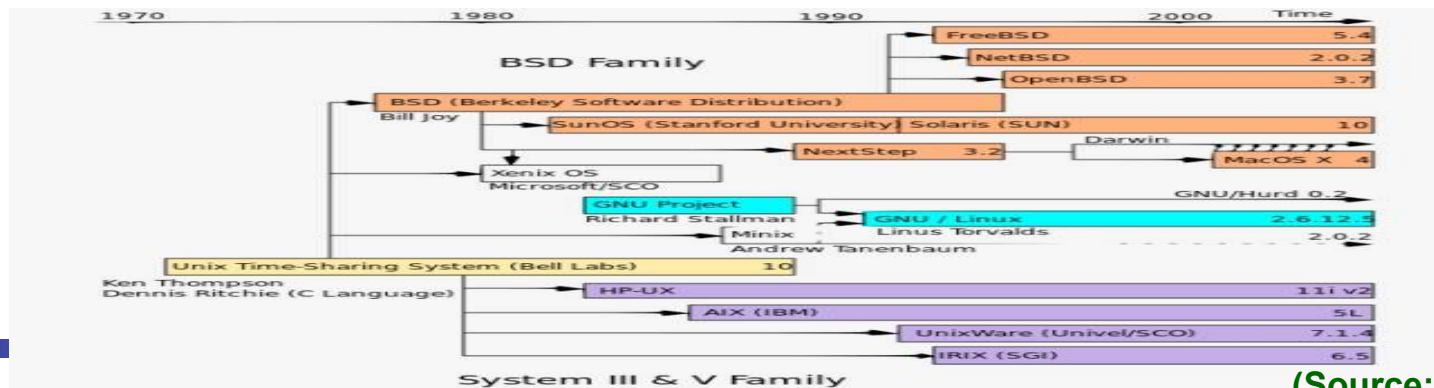
Linux Introduction (7/8)

■ Some notes about UNIX and Linux (From LPI Chapter 1)

✓ Linux is a member of the UNIX family

✓ History

- 1969~ : **UNIX** Invented by Ken and Dennis, UNIX 1~7 edition at AT&T
- 1975~ : popularly used at universities include Berkeley, MIT and CMU.
- 1979~ : **BSD** and new features (FFS, TCP/IP, C shell, ...)
- 1981~ : System III & System V from AT&T
- 1985~ : UNIX golden ages (IBM, HP, Sun, NeXTStep, SCO, ...) → UNIX War
- 1984~ : **GNU** by R. Stallman (gcc, Emacs, bash, ...), GPL (General Public License)
- 1990~ : Standardization (**POSIX**, FIPS, X/Open, SUS (Single UNIX Spec.))
- 1991~ : **Linux** by L. Torvalds, Minix + Intel optimization, GNU incorporation
- 2023: Three representative OSES + Vendor proprietary OSES + New OSES



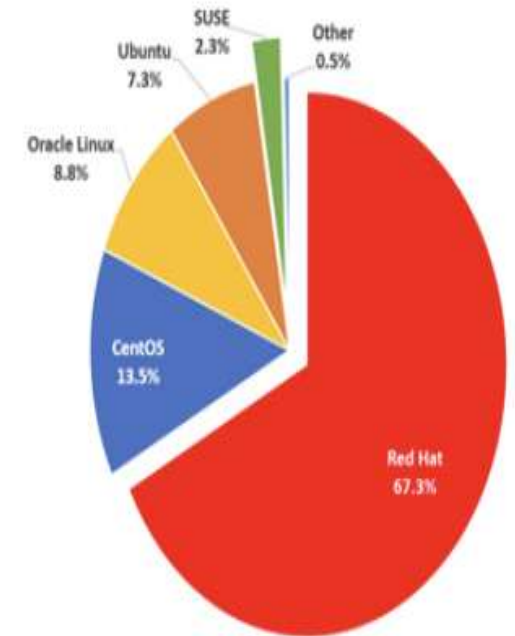
(Source: www.pngkey.com/)

Linux Introduction (8/8)

■ Linux vs. Distribution

- ✓ Linux: Kernel
- ✓ Distribution: Kernel + Packages + Frameworks + ...

구분	Red Hat	CentOS	Oracle	SUSE	Canonical
설립 연도	1993년	2004년	1977년 Oracle Linux (Since 2006년)	1992년	2004년
본사	Raleigh, North Carolina, U.S.	Raleigh, North Carolina, U.S.	Redwood City, California, U.S.	Nuremberg, Germany	London, United Kingdom
매출 규모	US \$3.4 Billion (2018년)	N/A	US \$39.5 Billion (2019년)	US \$303.4 Million (2017년)	US \$110 Million (2018년)
직원수 (가변적)	12,600명 (한국레드햇: 100명 이상)	Community Project Members	136,000명 (한국Oracle: Linux 담당 10명 미만)	1,750명 (수세코리아: 10명 미만)	443명 (캐노니컬: 10명 미만)
모회사	IBM (2018년)	Red Hat (2014년)		Marcel BidCo GmbH (2018년)	



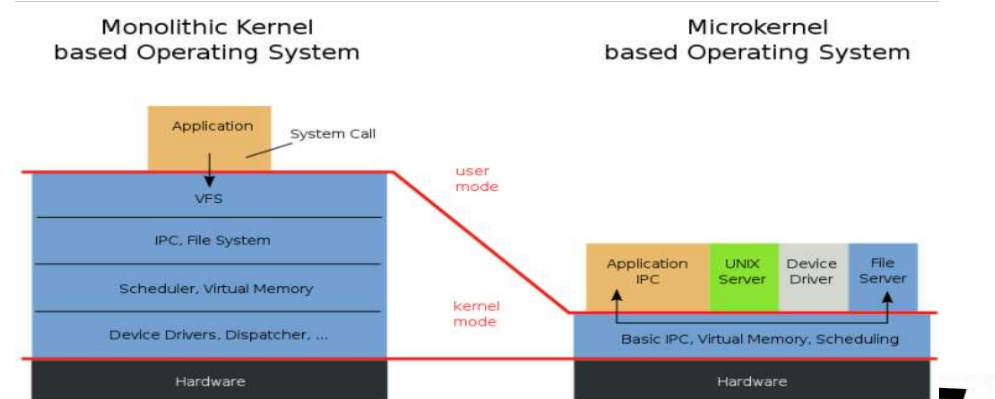
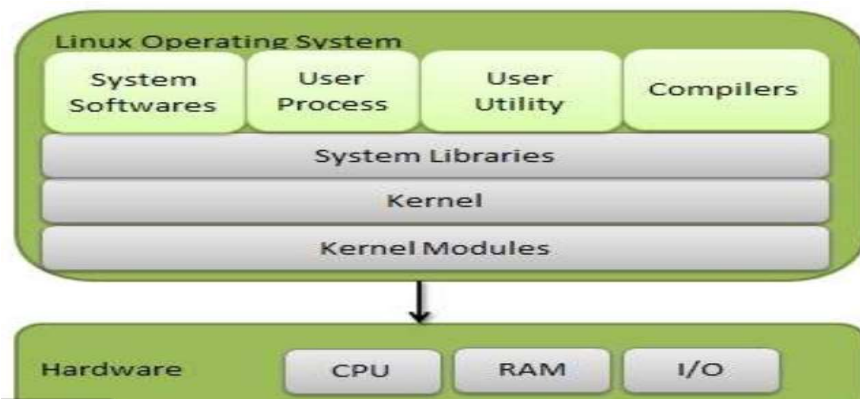
[그림 3] 'A' 기업 내 Linux 배포본 사용 현황

(Source: https://www.samsungsds.com/kr/insights/linux_distribution.html)



Fundamental Concepts of Linux (1/7)

- From [LPI Chapter 2](#)
- OS vs. Kernel
 - ✓ OS: Computing environments
 - Kernel + System SW (GUI, GCC, Packages, Util., ...)
 - ✓ Kernel: Central part of OS
 - Run in **kernel mode** (protected mode, supervisor mode) vs. user mode
 - To protect kernel from applications
 - Role: 1) Process mgmt., 2) VM, 3) FS, 4) Device driver, 5) Networking, 6) HAL, 7) system call, 8) multi-user support, ...
 - ✓ Kernel architecture
 - Monolithic kernel vs. Microkernel (u-kernel)
 - Dynamic kernel module: dynamic loadable SW runs in kernel mode



Fundamental Concepts of Linux (2/7)

■ The shell

- ✓ Special-purpose program designed to read commands typed by a user and execute them → command interpreter
- ✓ Examples: Bourne shell (Bell Lab.), C shell (BSD), Korn Shell (AT&T), bash (GNU)

■ Users and Groups

- ✓ 3 categories: user, group, others
- ✓ Superuser: has special privileges (User ID: 0, login name: root)

■ Unix Shell application comparison table

Application	sh	csh	ksh	bash	tcsh
Job control	N	Y	Y	Y	Y
Aliases	N	Y	Y	Y	Y
Input/Output redirection	Y	N	Y	Y	N
Command history	N	Y	Y	Y	Y
Command line editing	N	N	Y	Y	Y
Vi Command line editing	N	N	Y	Y	Y
Underlying Syntax	sh	csh	ksh	sh	csh

(Source: <https://stackoverflow.com/questions/5725296/difference-between-sh-and-bash>)



Fundamental Concepts of Linux (3/7)

■ File and directory

- ✓ **file types**: regular, directory, link, device, ... (everything is file)
- ✓ directory: a set of related file, support hierarchical structure
- ✓ **Home directory**, root directory, current directory

■ File I/O Model

- ✓ stdio library: fopen(), fread(), fwrite(), fclose(), printf(), scanf(), ...
- ✓ system call: open(), read(), write(), close(), ... → LN3
- ✓ After open(): file name → file **descriptor**

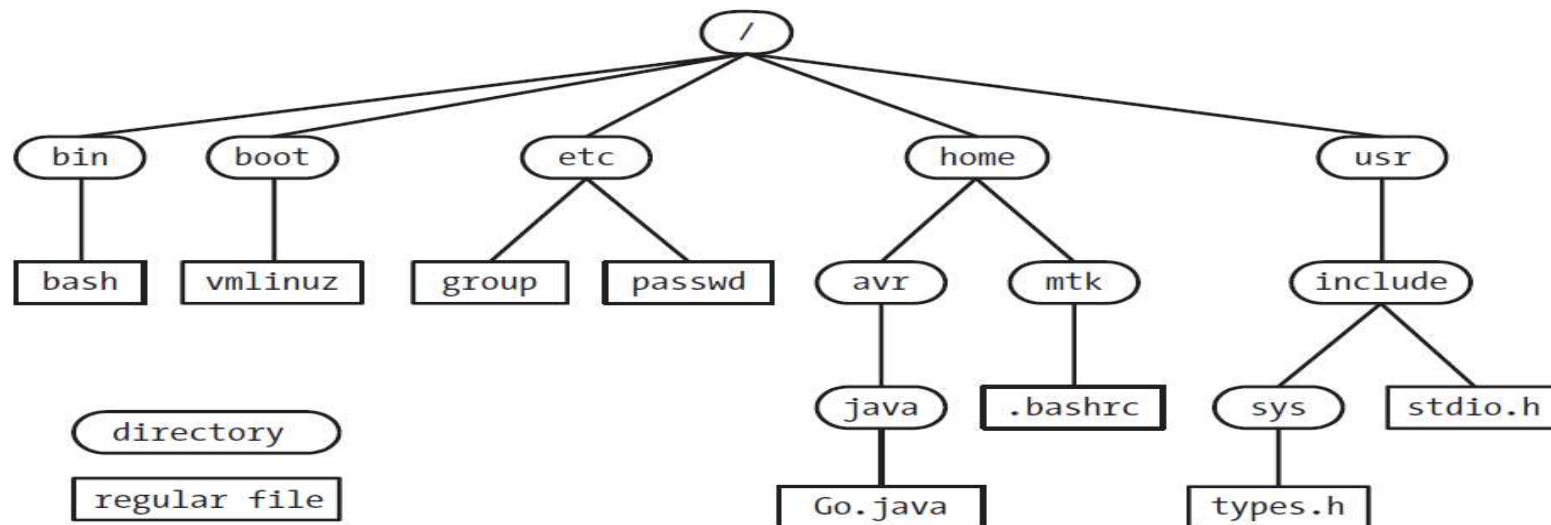


Figure 2-1: Subset of the Linux single directory hierarchy



Fundamental Concepts of Linux (4/7)

■ Program and Process

✓ Program

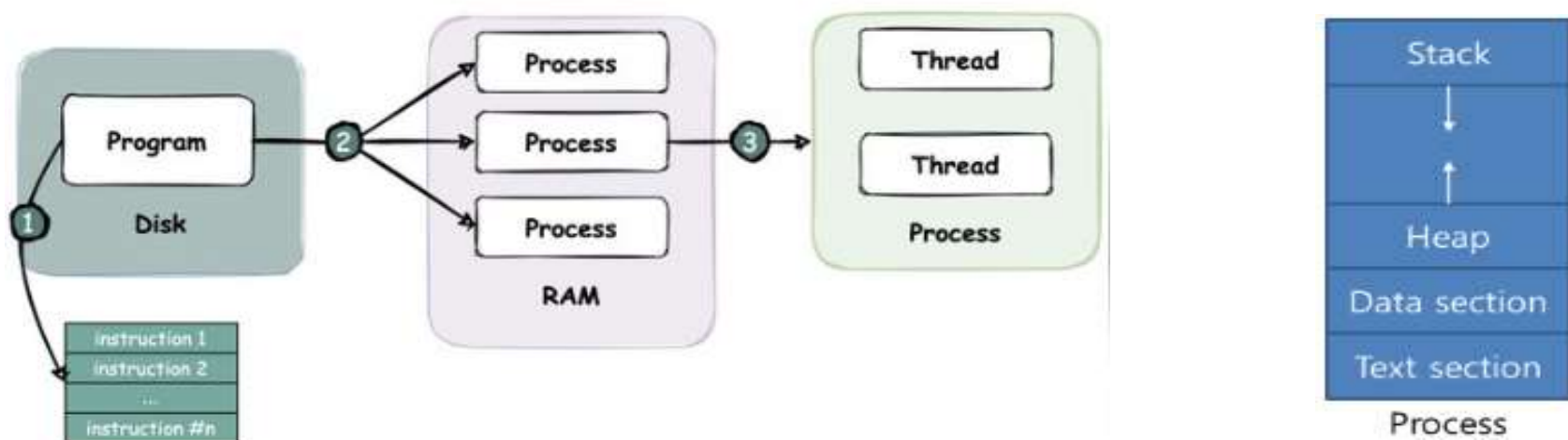
- A set of instructions that describes how to perform a specific task
- Two forms: source code, binary (machine language)

✓ Processes

- An instance of an **executing program** → LN4, 5
- Has its own virtual memory (layout: text, data, heap, stack, map)

■ Thread

- ✓ **A flow control** in a process (threads share virtual memory) → LN5

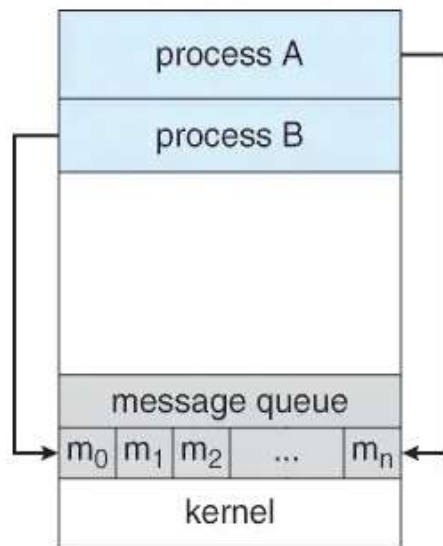


(Source: <https://twitter.com/alexsubyte/status/1518615214316425216>)

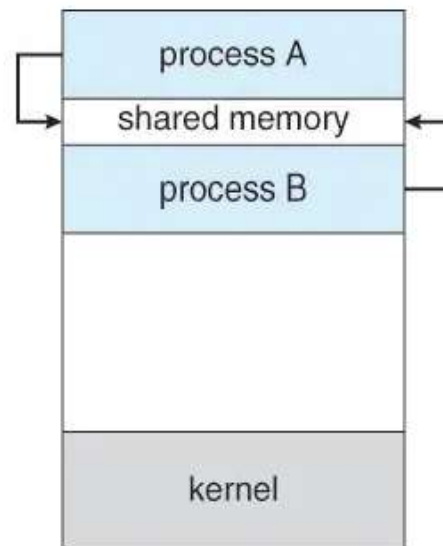


Fundamental Concepts of Linux (5/7)

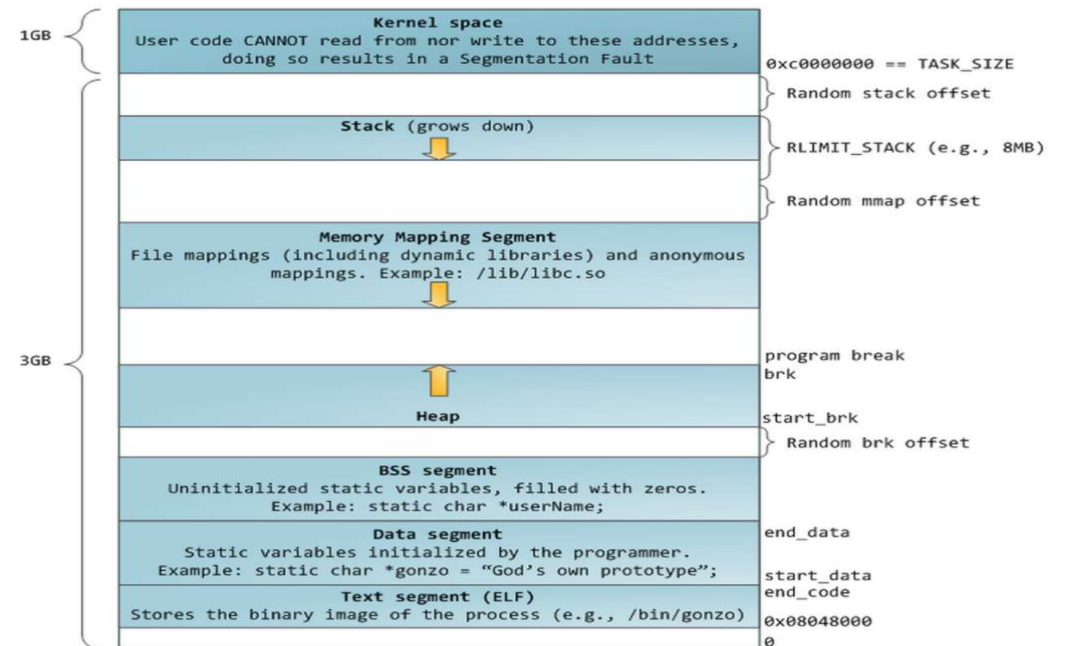
- IPC (Inter Process Communication) and Synchronization
 - ✓ For communication among processes and Process orchestration
 - ✓ Examples: signal, pipe, socket, message queue, shared memory, semaphore, ...
- Memory Mappings
 - ✓ mmap(): maps a file into the calling process's virtual memory
 - Access file using a pointer instead of open()/read()/write()



(a) Message Passing



(b) Shared Memory



(Source: <https://networkencyclopedia.com/interprocess-communication-ipc/> and brunch.co.kr/@alden/13)



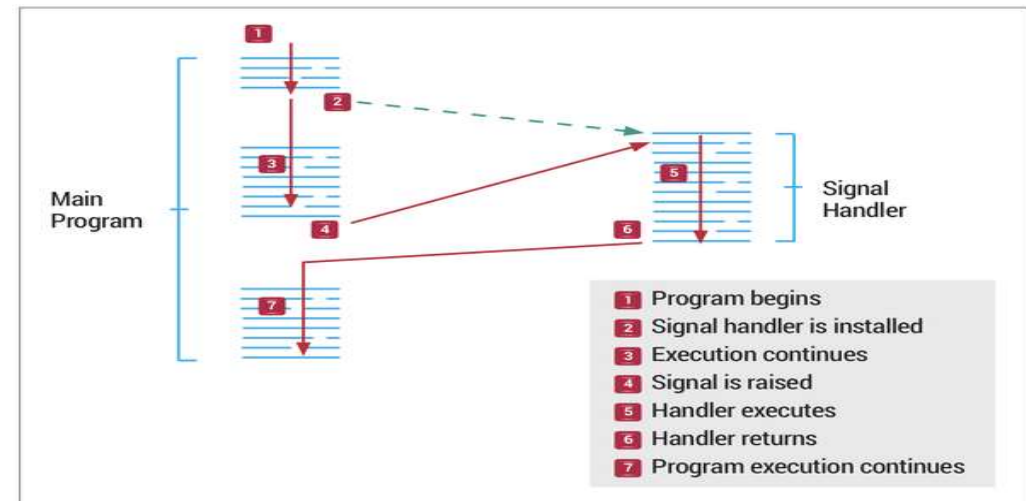
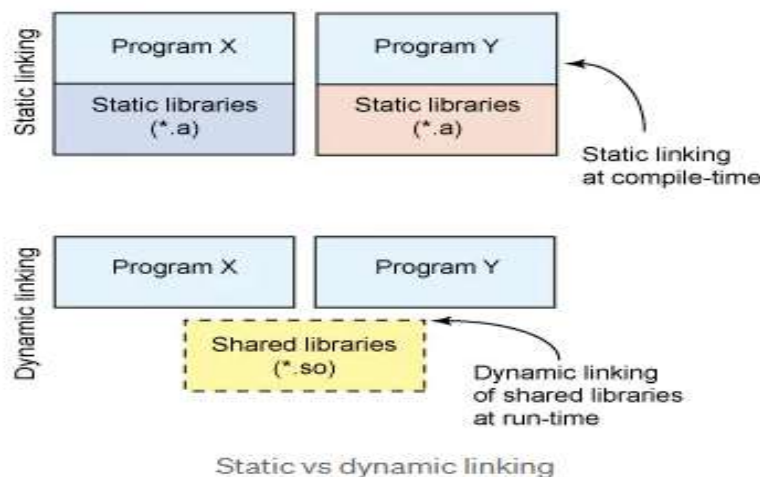
Fundamental Concepts of Linux (6/7)

■ Static and Shared Libraries

- ✓ Compiled objects (relocatable objects)
- ✓ Static libraries (also called as archive): compile-time linking
 - extracts copies of the required object modules from the library and copies these into an executable file
- ✓ Shared libraries: run-time linking
 - instead of copying object modules from library into executable, just write a reference, which allows shared libraries to be linked on-demand

■ Signal

- ✓ User-level interrupt: inform to a process (^C)
- ✓ c.f.) Interrupt: a mechanism to inform an event to kernel



(Source: medium.com/@birnbera/static-vs-dynamic-libraries-5912efe9bf52 and devopedia.org/linux-signals)



Fundamental Concepts of Linux (7/7)

■ Process group (Job control)

- ✓ Allows the user to simultaneously execute and manipulate multiple commands or pipelines.

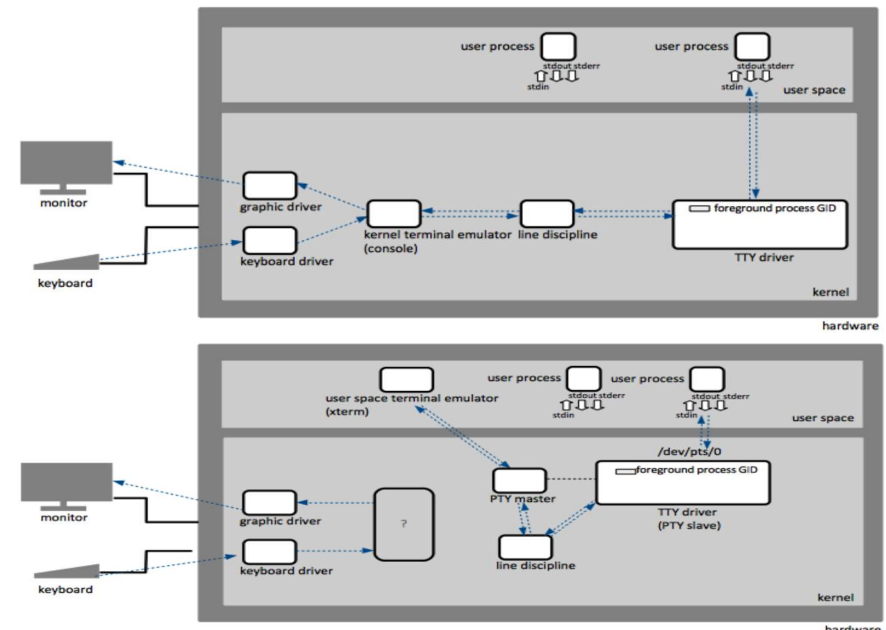
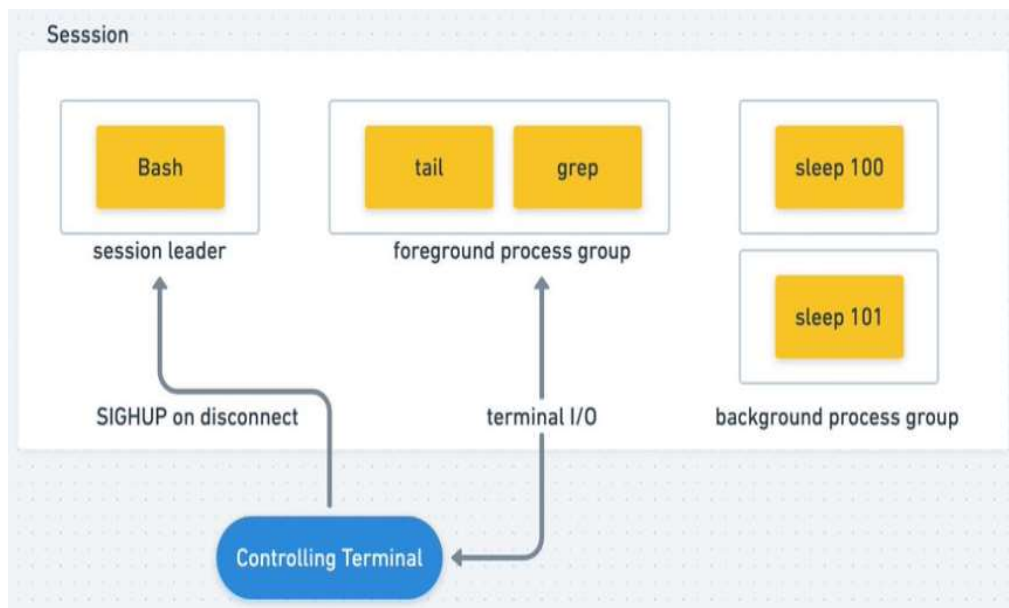
```
$ ls -l | sort -k5n | less
```

■ Session

- ✓ A session is a collection of process groups (jobs).
- ✓ Related with a terminal (controlling terminal, usually login terminal)
 - One **foreground job** and multiple background jobs

■ Pseudo-terminal

- ✓ Connected **virtual devices** (e.g. terminal emulator)

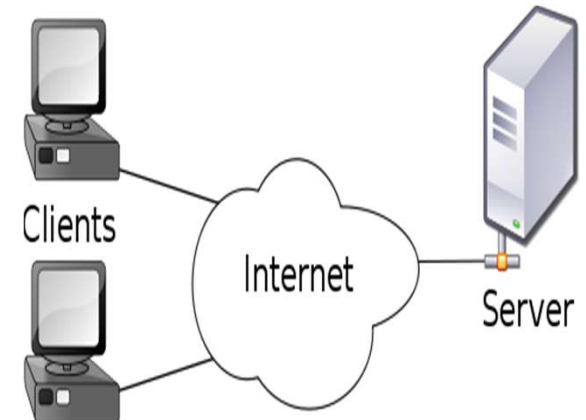
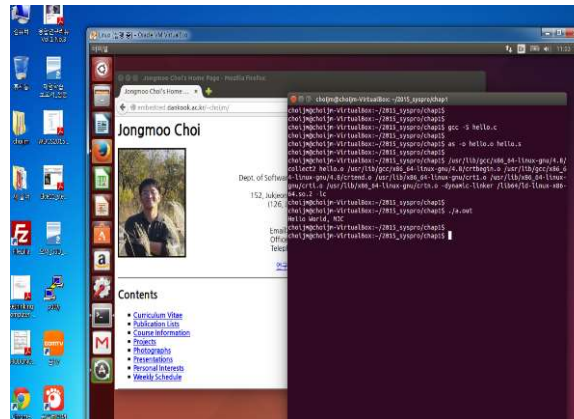


(Source: twitter.com/igor_sarcevic/status/1157349076809191425 and kb.novaordis.com/index.php/Linux_TTY)



How to access Linux (1/4)

- 1) Standalone (usually with multi-boot)
- 2) Virtualization (or WSL)
- 3) Client-Server



✓ In our course

- Client: terminal emulator (telnet/ssh client, putty, ...)
- Server: Linux system (PC)
 - IP: 220.149.236.2 (primary), 220.149.236.4 (secondary)
- Alternative: Amazon EC2, Google Cloud, MS Azure or ToastCloud



How to access Linux (2/4)

■ Client

- ✓ telnet, ssh, ping, ...
- ✓ putty, SecureCRT, powershell, ...

```
choijm@embedded: ~
PS C:\Users\waeran>
PS C:\Users\waeran> ping 220.149.236.2

Ping 220.149.236.2 32바이트 데이터 사용:
220.149.236.2의 응답: 바이트=32 시간=6ms TTL=54
220.149.236.2의 응답: 바이트=32 시간=6ms TTL=54
220.149.236.2의 응답: 바이트=32 시간=6ms TTL=54
220.149.236.2의 응답: 바이트=32 시간=6ms TTL=54

220.149.236.2에 대한 Ping 통계:
패킷: 보낸 = 4, 받은 = 4, 손실 = 0 (0% 손실),
왕복 시간(밀리초):
최소 = 6ms, 최대 = 6ms, 평균 = 6ms
PS C:\Users\waeran>
PS C:\Users\waeran> ping 220.149.236.4

Ping 220.149.236.4 32바이트 데이터 사용:
220.149.236.4의 응답: 바이트=32 시간=7ms TTL=54
220.149.236.4의 응답: 바이트=32 시간=7ms TTL=54
220.149.236.4의 응답: 바이트=32 시간=7ms TTL=54
220.149.236.4의 응답: 바이트=32 시간=7ms TTL=54

220.149.236.4에 대한 Ping 통계:
패킷: 보낸 = 4, 받은 = 4, 손실 = 0 (0% 손실),
왕복 시간(밀리초):
최소 = 7ms, 최대 = 8ms, 평균 = 7ms
PS C:\Users\waeran>
PS C:\Users\waeran> ssh 220.149.236.2 -l choijm
choijm@220.149.236.2's password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.15.0-91-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

146 packages can be updated.
0 updates are security updates.
```

Download PuTTY: latest release (0.79)

[Home](#) | [FAQ](#) | [Feedback](#) | [Licence](#) | [Updates](#) | [Mirrors](#) | [Keys](#) | [List](#)

Download: [Stable](#) · [Snapshot](#) | [Docs](#) | [Changes](#) | [Wishlist](#)

This page contains download links for the latest released version of PuTTY. Currently this is 0.79, released on 2023-08-26.

When new releases come out, this page will update to contain the latest, so this is a good page to bookmark or link to. Alternatively, here is a [permanent link to the 0.79 release](#).

Release versions of PuTTY are versions we think are reasonably likely to work well. However, they are often not the most up-to-date version of the code available. If you have a problem with this release, then it might be worth trying out the [development snapshots](#), to see if the problem has already been fixed in those versions.

Package files

You probably want one of these. They include versions of all the PuTTY utilities (except the new and slightly experimental Windows pterm).

(Not sure whether you want the 32-bit or the 64-bit version? Read the [FAQ entry](#).)

We also publish the latest PuTTY installers for all Windows architectures as a free-of-charge download at the [Microsoft Store](#); they usually take a few days to appear there after we release them.

MSI ("Windows Installer")

64-bit x86:	putty-64bit-0.79-installer.msi	(signature)
64-bit Arm:	putty-arm64-0.79-installer.msi	(signature)
32-bit x86:	putty-0.79-installer.msi	(signature)

Unix source archive

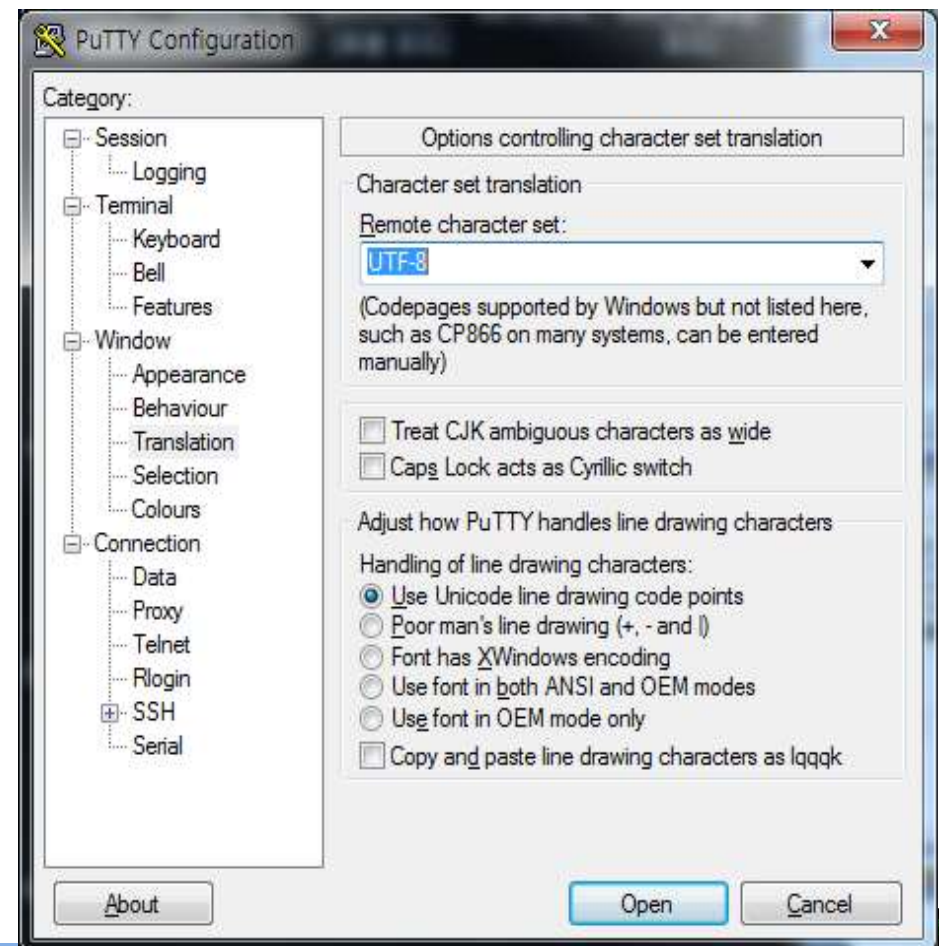
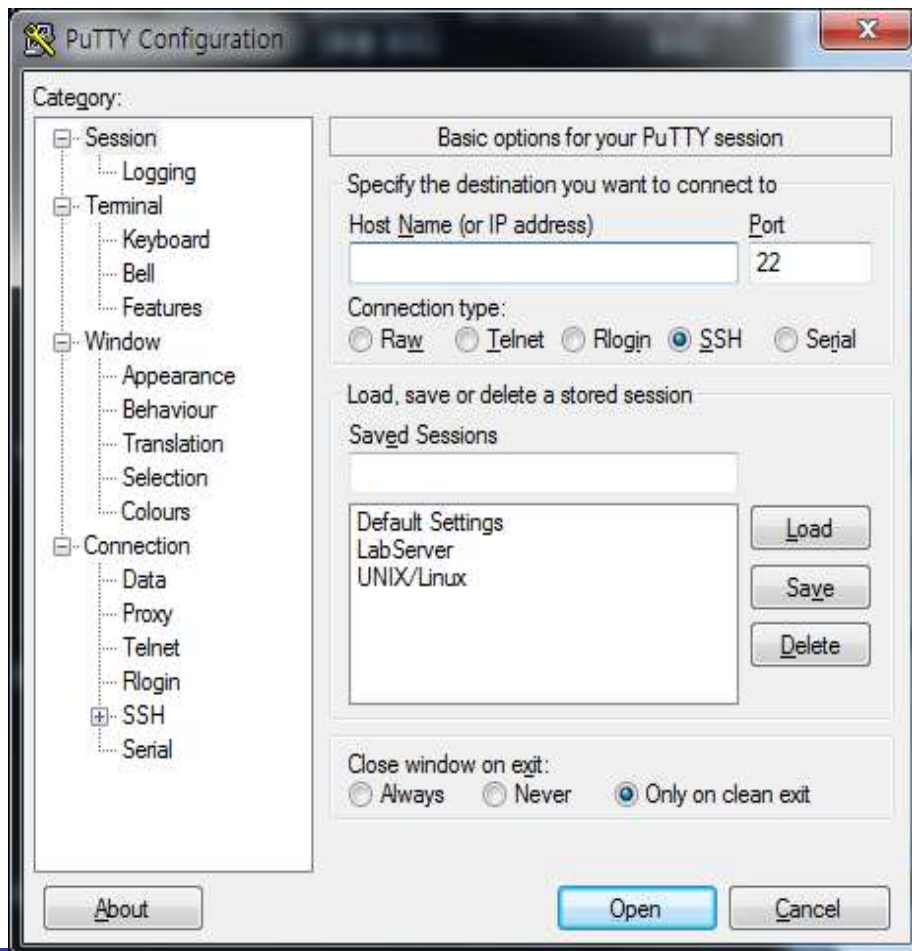
.tar.gz:	putty-0.79.tar.gz	(signature)
----------	-----------------------------------	-----------------------------

Alternative binary files

How to access Linux (3/4)

■ Putty with ssh

- ✓ IP: 220.149.236.2 (check that “type is ssh” and “port is 22”)
- ✓ Colours: click “Use system colours
- ✓ Translation: choose “UTF-8”



How to access Linux (4/4)

■ Login and shell

```
220.149.236.2 - PuTTY
login as: █
```

```
choijm@embedded: ~
login as: choijm
choijm@220.149.236.2's password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.15.0-142-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

136 packages can be updated.
0 updates are security updates.

Last login: Thu Sep  7 16:54:40 2023 from 172.23.14.164
choijm@embedded:~$
choijm@embedded:~$ ls
Desktop          OSTEP  test   test.o  virt_cpu
examples.desktop Public test1.c test.out virt_cpu.c
man_pipe_output.txt Syspro test.c  test.s
choijm@embedded:~$
choijm@embedded:~$ passwd
Changing password for choijm.
(current) UNIX password:
Enter new UNIX password: █
```

- ✓ ID: sys학번 (8 numbers of Student ID)
- ✓ Default passwd: sys***** (change using the “passwd” command)



How to use commands in Linux (1/12)

■ UNIX

- ✓ Two key objects in UNIX: file as a “**place**” and process (task) as a “**life**” (by M. Bach, The Design of the UNIX Operating Systems)

■ File

- ✓ **Array of bytes**, stream of character (attributes: start, size, current offset)
- ✓ Associated with disk blocks
- ✓ Supports a variety of objects using file concept (eg. device, network, memory, and even process)

■ Process (Task)

- ✓ **Program in execution**
- ✓ Associated with CPUs (Scheduling entity)
- ✓ Having context such as memory space and CPU registers



How to use commands in Linux (2/12)

■ file related command

- ✓ create
 - vi, gcc, mknod, ...
- ✓ copy/move
 - cp, mv, ln, ...
- ✓ delete
 - rm
- ✓ listing
 - ls
- ✓ file content view
 - **cat**, **more**, less, head, tail, objdump, hexdump
- ✓ file attributes manipulation
 - chmod, chown, chgrp, touch
- ✓ redirection
 - >

```
choijm@embedded-desktop: ~  
choijm@embedded-desktop:~$ ls  
choijm@embedded-desktop:~$ vi hello.c  
choijm@embedded-desktop:~$ gcc hello.c  
choijm@embedded-desktop:~$ ls  
s.out hello.c  
choijm@embedded-desktop:~$ ./a.out  
Hello System Programming  
choijm@embedded-desktop:~$ more hello.c  
#include <stdio.h>  
  
int main()  
{  
    printf("Hello System Programming\n");  
}  
choijm@embedded-desktop:~$  
choijm@embedded-desktop:~$ cp hello.c hello_new.c  
choijm@embedded-desktop:~$ ls  
s.out hello.c hello_new.c  
choijm@embedded-desktop:~$ rm hello_new.c  
choijm@embedded-desktop:~$ ls  
s.out hello.c  
choijm@embedded-desktop:~$ man ls
```

```
choijm@embedded-desktop: ~  
LS (1) User Commands LS (1)  
NAME  
ls - list directory contents  
SYNOPSIS  
ls [OPTION]... [FILE]...  
DESCRIPTION  
List information about the FILES (the current  
directory by default). Sort entries alphabetically  
if none of -oftuvSUX nor --sort is specified.  
Mandatory arguments to long options are mandatory  
for short options too.  
-a, --all  
do not ignore entries starting with .  
-A, --almost-all  
do not list implied . and ..  
--author  
with -l, print the author of each file  
-b, --escape  
print C-style escapes for nongraphic charac  
ters  
--block-size=SIZE  
Manual page ls(1) line 1 (press h for help or q to quit)
```



How to use commands in Linux (3/12)

■ directory

- ✓ a set of files
- ✓ provide hierarchical structure of files
- ✓ home directory, root directory, current directory
- ✓ relative path, absolute path

■ directory related command

- ✓ create
 - mkdir
- ✓ change
 - cd
- ✓ delete
 - rmdir
- ✓ current position
 - pwd

```
choijm@embedded: ~  
choijm@embedded:~$ pwd  
/home/choijm  
choijm@embedded:~$ ls  
examples.desktop  README  syspro18  
choijm@embedded:~$  
choijm@embedded:~$ mkdir programming  
choijm@embedded:~$ mkdir music  
choijm@embedded:~$  
choijm@embedded:~$ cd programming/  
choijm@embedded:~/programming$ vi hello.c  
choijm@embedded:~/programming$ gcc hello.c  
choijm@embedded:~/programming$ ./a.out  
Hello DKU World  
choijm@embedded:~/programming$  
choijm@embedded:~/programming$ ls  
a.out hello.c  
choijm@embedded:~/programming$ pwd  
/home/choijm/programming  
choijm@embedded:~/programming$  
choijm@embedded:~/programming$ ls .  
a.out hello.c  
choijm@embedded:~/programming$ ls ..  
examples.desktop  music  programming  README  syspro18  
choijm@embedded:~/programming$  
choijm@embedded:~/programming$ cp ../README .  
choijm@embedded:~/programming$ ls  
a.out hello.c README  
choijm@embedded:~/programming$ cp /home/choijm/README README_new  
choijm@embedded:~/programming$ ls  
a.out hello.c README README_new  
choijm@embedded:~/programming$ cd ..  
choijm@embedded:~$
```


How to use commands in Linux (4/12)

■ file attribute manipulation

- ✓ Permission and owner
- ✓ cf. [Command format](#): 1) command, 2) option, 3) argument

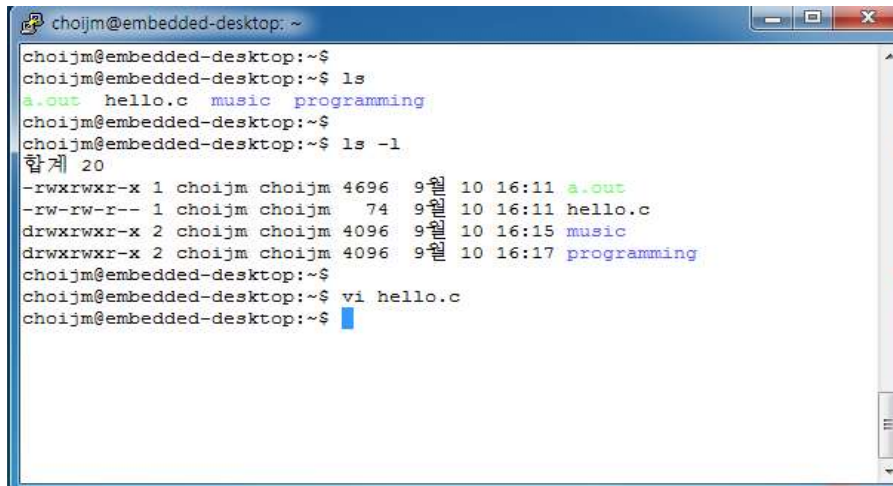
```
choijm@embedded-desktop: ~  
choijm@embedded-desktop:~$  
choijm@embedded-desktop:~$ ls  
a.out hello.c music programming  
choijm@embedded-desktop:~$  
choijm@embedded-desktop:~$ ls -l  
합계 20  
-rwxrwxr-x 1 choijm choijm 4696  9월 10 16:11 a.out  
-rw-rw-r-- 1 choijm choijm   74  9월 10 16:20 hello.c  
drwxrwxr-x 2 choijm choijm 4096  9월 10 16:15 music  
drwxrwxr-x 2 choijm choijm 4096  9월 10 16:17 programming  
choijm@embedded-desktop:~$  
choijm@embedded-desktop:~$ chmod o+w hello.c  
choijm@embedded-desktop:~$  
choijm@embedded-desktop:~$ ls -l  
합계 20  
-rwxrwxr-x 1 choijm choijm 4696  9월 10 16:11 a.out  
-rw-rw-rw- 1 choijm choijm   74  9월 10 16:20 hello.c  
drwxrwxr-x 2 choijm choijm 4096  9월 10 16:15 music  
drwxrwxr-x 2 choijm choijm 4096  9월 10 16:17 programming  
choijm@embedded-desktop:~$  
choijm@embedded-desktop:~$ chmod g-x a.out  
choijm@embedded-desktop:~$  
choijm@embedded-desktop:~$ ls -l  
합계 20  
-rwxrw-r-x 1 choijm choijm 4696  9월 10 16:11 a.out  
-rw-rw-rw- 1 choijm choijm   74  9월 10 16:20 hello.c  
drwxrwxr-x 2 choijm choijm 4096  9월 10 16:15 music  
drwxrwxr-x 2 choijm choijm 4096  9월 10 16:17 programming  
choijm@embedded-desktop:~$
```



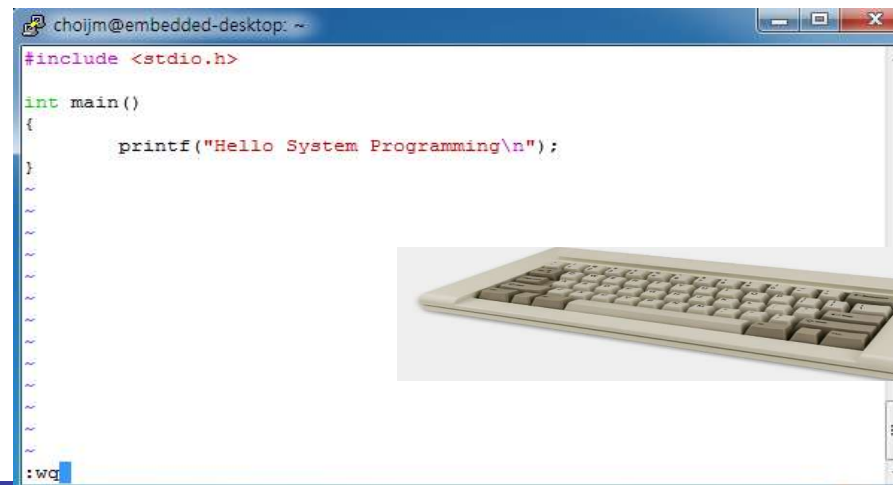
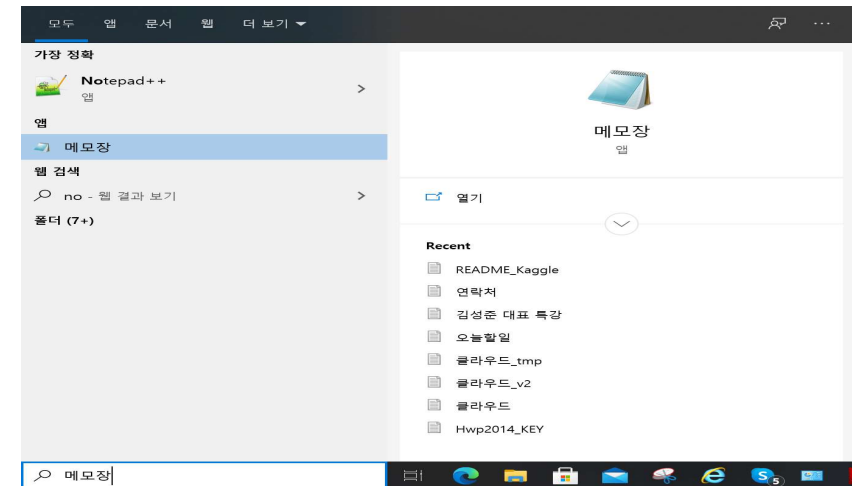
How to use commands in Linux (5/12)

■ vi editor (vim)

- ✓ What are the differences between vi and notepad (or VS code)
 - Explicit input mode vs. Instant editable
 - No “파일” or “편집” button → need line command mode



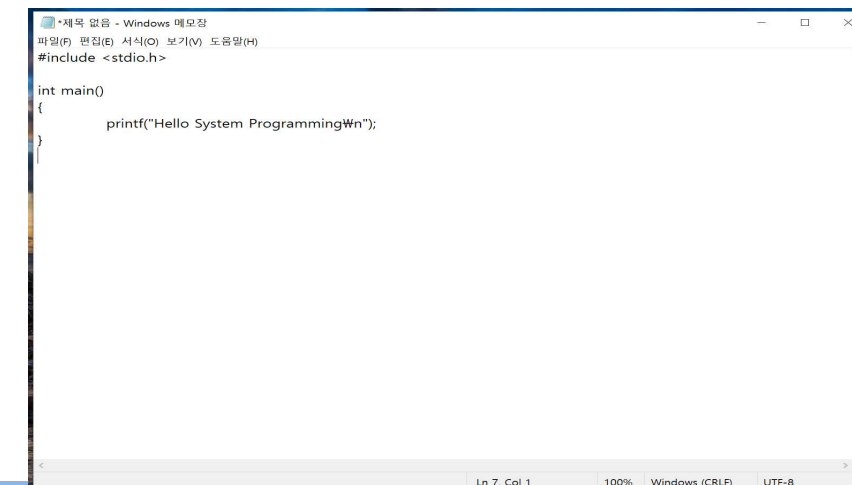
```
choijm@embedded-desktop: ~$
choijm@embedded-desktop:~$ ls
a.out hello.c music programming
choijm@embedded-desktop:~$
choijm@embedded-desktop:~$ ls -l
합계 20
-rwxrwxr-x 1 choijm choijm 4696  9월 10 16:11 a.out
-rw-rw-r-- 1 choijm choijm   74  9월 10 16:11 hello.c
drwxrwxr-x 2 choijm choijm 4096  9월 10 16:15 music
drwxrwxr-x 2 choijm choijm 4096  9월 10 16:17 programming
choijm@embedded-desktop:~$
choijm@embedded-desktop:~$ vi hello.c
choijm@embedded-desktop:~$
```



```
choijm@embedded-desktop: ~$
#include <stdio.h>

int main()
{
    printf("Hello System Programming\n");
}

:bw
```



```
*제목 없음 - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
#include <stdio.h>

int main()
{
    printf("Hello System Programming\n");
}

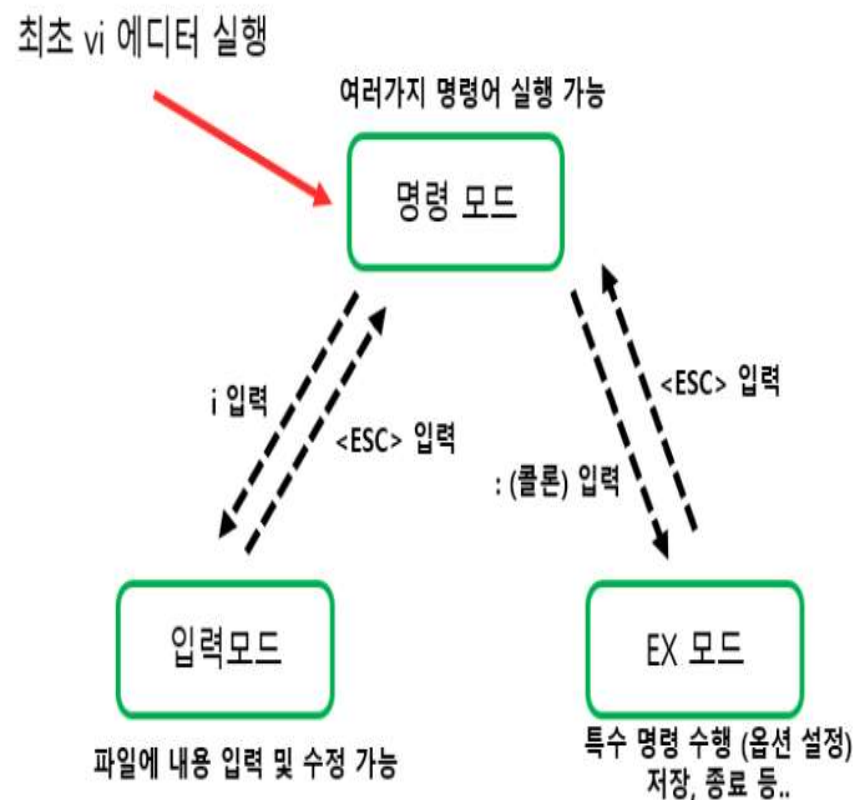
Ln 7, Col 1 100% Windows (CRLF) UTF-8
```

How to use commands in Linux (6/12)

■ vi editor (vim)

✓ 3 modes

- command/input/line command(a.k.a. execution mode)
- ✓ At first (just before loading vi): command mode
- ✓ Switch to the input mode
 - a (append), i (insert), o, r, ...
- ✓ Switch to the command mode
 - ESC
- ✓ Switch to the line command mode
 - : at command mode
- ✓ Switch to the command mode
 - Enter or ESC



(Source: <https://dololak.tistory.com/379>)



How to use commands in Linux (7/12)

■ vi editor (vim)

- ✓ Actions allowed at the command/line command mode
 - Navigation (cursor movement): up/down, begin/end of word/line, ...
 - File management: save, quit (e.g. :wq or :q), open, ...
 - Editing: delete, change, substitute, transpose, ...
 - Multiple windows, files, shell interaction, ...

Vim: Navigation

Keystroke	Function
B/b	Move cursor to bottom of page *
E/e	Move cursor to end of word *
0 (Zero) /	Move cursor to beginning of line *
\$	Move cursor to end of line
)	Move cursor to beginning of next sentence
(Move cursor to beginning of current sentence
G	Move cursor to end of file *
%	Move cursor to the matching bracket; Place cursor on {}>()
' (Apostrophe dot)	Move cursor to previously modified line
'a (Apostrophe a)	Move cursor to line mark "a" generated by marking "ma"

Advanced editing: Multiple Windows This is a Vim only feature

Command	Function
:sp	Split current window horizontally in two
:vsp	Split current window vertically into two
vim -O [n files...]	Opens n windows, files split vertically
:new	Open a new blank window
:on	Make current window the only window
:q	Quit current window
:qa	Quit all windows
:xa	Save and quit all windows
[Ctrl+w]+/-	Increase/decrease window size
[Ctrl+w] [Ctrl+w]	Toggle between windows

Pattern Substitutions

- General format of substitution
:[.[\$|%]s/s1/s1[switches] or :n1,n2s/s1/s2/[switches]
- [switches] are: **g|c|i|l** meaning
global/confirmation/ignore-case/no-ignore-case

Some interesting examples of pattern substitutions

Command	Function
:1,\$s##/g	Globally remove #
:3,10s/^#!/	Insert # at the beginning of line 3 to 10
:\$s\$/;/	Insert a ; at the end of last line
:%s/abc/xyz/gc	Globally replace abc by xyz interactively
:1,\$s/include/<&/g	Globally replace include by <include>

```
choijm@embedded: ~
#include <stdio.h>

int a, b, c;

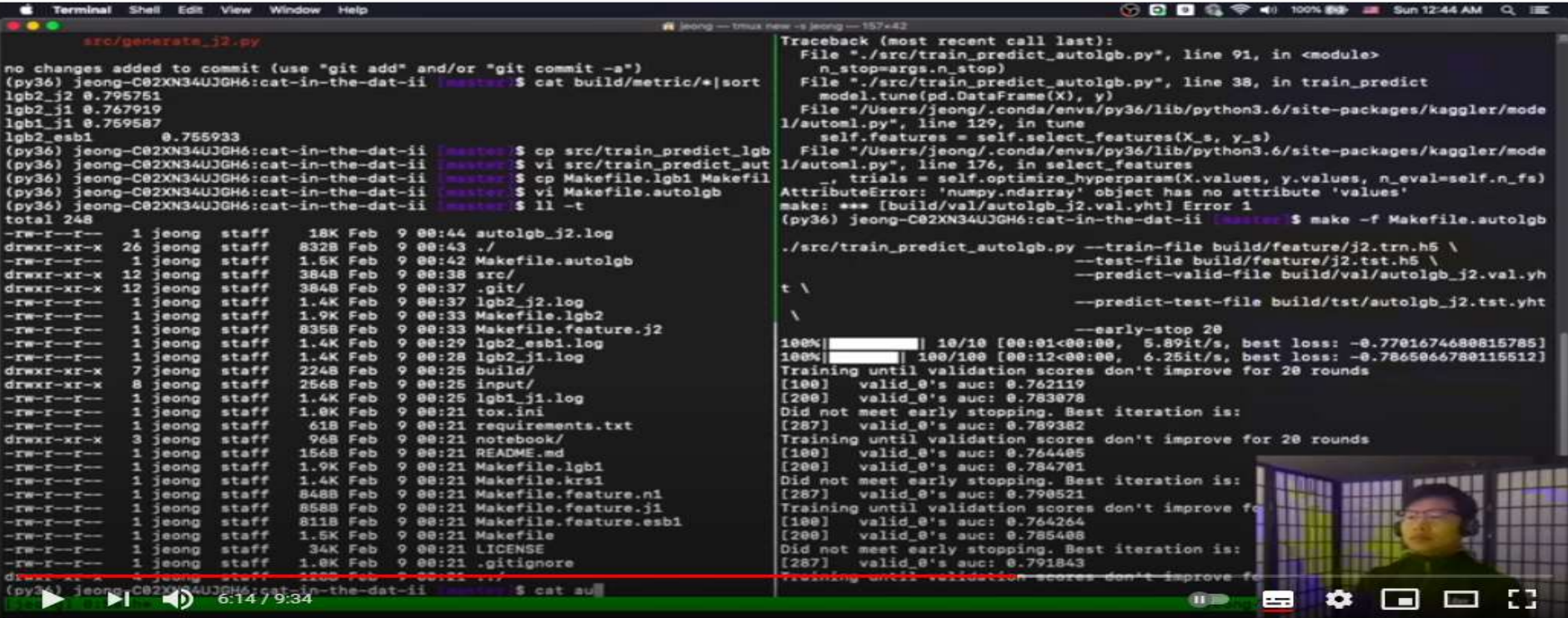
main()
{
    a = 10;
    b = 20;
    c = a + b;
    printf("Hello\n");
}

test.c 5,1 All
#include <stdio.h>
int a, b, c;
main()
{
    a = 10;
    b = 20;
    c = a + b;
    printf("Hello\n");
}
test.c 3,4 All
```



How to use commands in Linux (8/12)

- Reference: Dr. Jeong-Yoon Lee's Kaggle demo ([terminal mode](https://www.youtube.com/watch?v=861NAO5-XJo))



The screenshot shows a terminal window with the following content:

```
src/generate_j2.py
no changes added to commit (use "git add" and/or "git commit -a")
(py36) jeong-C02XN34UJGH6:cat-in-the-dat-ii [master] $ cat build/metric/*|sort
lgb2_j2 0.795751
lgb2_j1 0.767919
lgb1_j1 0.759587
lgb2_esb1 0.755933
(py36) jeong-C02XN34UJGH6:cat-in-the-dat-ii [master] $ cp src/train_predict_lgb
(py36) jeong-C02XN34UJGH6:cat-in-the-dat-ii [master] $ vi src/train_predict_aut
(py36) jeong-C02XN34UJGH6:cat-in-the-dat-ii [master] $ cp Makefile.lgb1 Makefil
(py36) jeong-C02XN34UJGH6:cat-in-the-dat-ii [master] $ vi Makefile.autolgb
(py36) jeong-C02XN34UJGH6:cat-in-the-dat-ii [master] $ ll -t
total 248
-rw-r--r-- 1 jeong staff 18K Feb 9 00:44 autolgb_j2.log
drwxr-xr-x 26 jeong staff 832B Feb 9 00:43 ./
-rw-r--r-- 1 jeong staff 1.5K Feb 9 00:42 Makefile.autolgb
drwxr-xr-x 12 jeong staff 384B Feb 9 00:38 src/
drwxr-xr-x 12 jeong staff 384B Feb 9 00:37 .git/
-rw-r--r-- 1 jeong staff 1.4K Feb 9 00:37 lgb2_j2.log
-rw-r--r-- 1 jeong staff 1.9K Feb 9 00:33 Makefile.lgb2
-rw-r--r-- 1 jeong staff 835B Feb 9 00:33 Makefile.feature.j2
-rw-r--r-- 1 jeong staff 1.4K Feb 9 00:29 lgb2_esb1.log
-rw-r--r-- 1 jeong staff 1.4K Feb 9 00:28 lgb2_j1.log
drwxr-xr-x 7 jeong staff 224B Feb 9 00:25 build/
drwxr-xr-x 8 jeong staff 256B Feb 9 00:25 input/
-rw-r--r-- 1 jeong staff 1.4K Feb 9 00:25 lgb1_j1.log
-rw-r--r-- 1 jeong staff 1.0K Feb 9 00:21 tox.ini
-rw-r--r-- 1 jeong staff 61B Feb 9 00:21 requirements.txt
drwxr-xr-x 3 jeong staff 96B Feb 9 00:21 notebook/
-rw-r--r-- 1 jeong staff 156B Feb 9 00:21 README.md
-rw-r--r-- 1 jeong staff 1.9K Feb 9 00:21 Makefile.lgb1
-rw-r--r-- 1 jeong staff 1.4K Feb 9 00:21 Makefile.krs1
-rw-r--r-- 1 jeong staff 848B Feb 9 00:21 Makefile.feature.n1
-rw-r--r-- 1 jeong staff 858B Feb 9 00:21 Makefile.feature.j1
-rw-r--r-- 1 jeong staff 811B Feb 9 00:21 Makefile.feature.esb1
-rw-r--r-- 1 jeong staff 1.5K Feb 9 00:21 Makefile
-rw-r--r-- 1 jeong staff 34K Feb 9 00:21 LICENSE
-rw-r--r-- 1 jeong staff 1.0K Feb 9 00:21 .gitignore
drwxr-xr-x 1 jeong staff 400B Feb 9 00:21 .
(py36) jeong-C02XN34UJGH6:cat-in-the-dat-ii [master] $ cat au/
```

The terminal output shows the following training progress:

```
./src/train_predict_autolgb.py --train-file build/feature/j2.trn.h5 \
--test-file build/feature/j2.tst.h5 \
--predict-valid-file build/val/autolgb_j2.val.yh
t \
--predict-test-file build/tst/autolgb_j2.tst.yht
\
--early-stop 20
100%|██████████| 10/10 [00:01<00:00, 5.89it/s, best loss: -0.7701674680815785]
100%|██████████| 100/100 [00:12<00:00, 6.25it/s, best loss: -0.7865066780115512]
Training until validation scores don't improve for 20 rounds
[100] valid_0's auc: 0.762119
[200] valid_0's auc: 0.783078
Did not meet early stopping. Best iteration is:
[287] valid_0's auc: 0.789382
Training until validation scores don't improve for 20 rounds
[100] valid_0's auc: 0.764405
[200] valid_0's auc: 0.784701
Did not meet early stopping. Best iteration is:
[287] valid_0's auc: 0.790521
Training until validation scores don't improve for 20 rounds
[100] valid_0's auc: 0.764264
[200] valid_0's auc: 0.785408
Did not meet early stopping. Best iteration is:
[287] valid_0's auc: 0.791843
Training until validation scores don't improve for 20 rounds
```

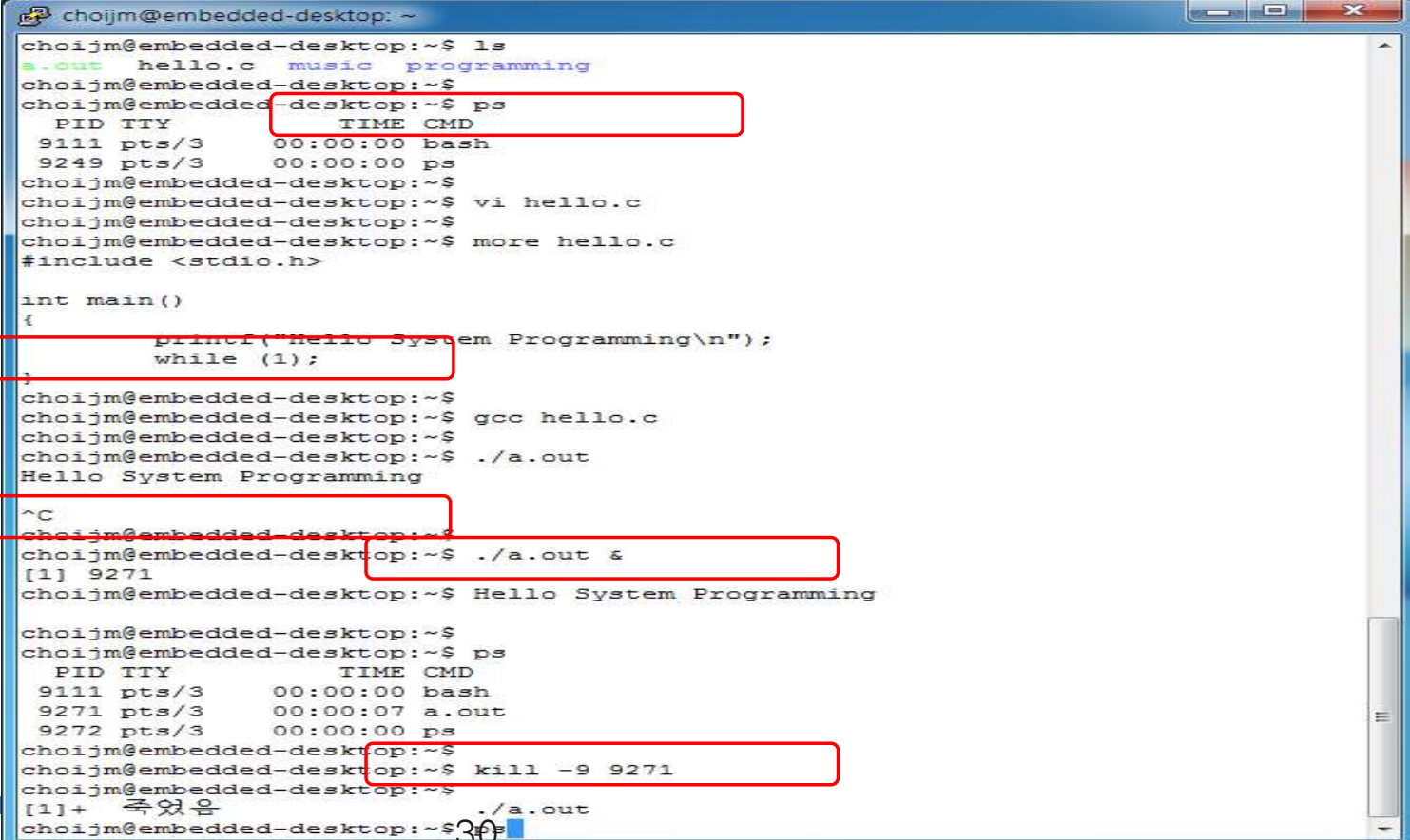
(Source: <https://www.youtube.com/watch?v=861NAO5-XJo>)



How to use commands in Linux (9/12)

■ process related commands

- ✓ process status
 - ps, pstree, top, /proc
- ✓ Creation and deletion
 - Implicitly: using shell (fork(), execve()) and exit() internally)
 - Explicitly: signal, kill command



```
choijm@embedded-desktop: ~
choijm@embedded-desktop:~$ ls
a.out hello.c music programming
choijm@embedded-desktop:~$ ps
  PID TTY          TIME CMD
  9111 pts/3        00:00:00 bash
  9249 pts/3        00:00:00 ps
choijm@embedded-desktop:~$ vi hello.c
choijm@embedded-desktop:~$ more hello.c
#include <stdio.h>

int main()
{
    printf("Hello System Programming\n");
    while (1);
}
choijm@embedded-desktop:~$ gcc hello.c
choijm@embedded-desktop:~$ ./a.out
Hello System Programming
^C
choijm@embedded-desktop:~$ ./a.out &
[1] 9271
choijm@embedded-desktop:~$ ps
  PID TTY          TIME CMD
  9111 pts/3        00:00:00 bash
  9271 pts/3        00:00:07 a.out
  9272 pts/3        00:00:00 ps
choijm@embedded-desktop:~$ kill -9 9271
[1]+  죽었음 ./a.out
choijm@embedded-desktop:~$
```

How to use commands in Linux (10/12)

■ Advanced commands: pipe

```
choijm@embedded: ~
choijm@embedded:~$ pwd
/home/choijm
choijm@embedded:~$ ls -l
total 56
-rwxrwxr-x 1 choijm choijm 4676 11월 19 2018 a.out
drwxr-xr-x 9 choijm choijm 4096 9월 5 11:51 Desktop
-rw-r--r-- 1 choijm choijm 8980 8월 31 2018 examples.desktop
drwxrwxr-x 2 choijm choijm 4096 3월 13 09:22 OSTEP
drwxr-xr-x 2 choijm choijm 4096 9월 5 2019 Public
drwxrwxr-x 10 choijm choijm 4096 11월 20 2019 Syspro
-rw-rw-r-- 1 choijm choijm 95 9월 17 2019 test.c
-rw-rw-r-- 1 choijm choijm 517 9월 17 2019 test.s
-rwxrwxr-x 1 choijm choijm 4880 3월 13 09:27 virt_cpu
-rw-rw-r-- 1 choijm choijm 269 3월 13 09:27 virt_cpu.c
choijm@embedded:~$
choijm@embedded:~$ ls -l | sort
drwxrwxr-x 10 choijm choijm 4096 11월 20 2019 Syspro
drwxrwxr-x 2 choijm choijm 4096 3월 13 09:22 OSTEP
drwxr-xr-x 2 choijm choijm 4096 9월 5 2019 Public
drwxr-xr-x 9 choijm choijm 4096 9월 5 11:51 Desktop
-rw-r--r-- 1 choijm choijm 8980 8월 31 2018 examples.desktop
-rw-rw-r-- 1 choijm choijm 269 3월 13 09:27 virt_cpu.c
-rw-rw-r-- 1 choijm choijm 517 9월 17 2019 test.s
-rw-rw-r-- 1 choijm choijm 95 9월 17 2019 test.c
-rwxrwxr-x 1 choijm choijm 4676 11월 19 2018 a.out
-rwxrwxr-x 1 choijm choijm 4880 3월 13 09:27 virt_cpu
total 56
choijm@embedded:~$ ls -l | sort -k5n
total 56
-rw-rw-r-- 1 choijm choijm 95 9월 17 2019 test.c
-rw-rw-r-- 1 choijm choijm 269 3월 13 09:27 virt_cpu.c
-rw-rw-r-- 1 choijm choijm 517 9월 17 2019 test.s
drwxrwxr-x 10 choijm choijm 4096 11월 20 2019 Syspro
drwxrwxr-x 2 choijm choijm 4096 3월 13 09:22 OSTEP
drwxr-xr-x 2 choijm choijm 4096 9월 5 2019 Public
drwxr-xr-x 9 choijm choijm 4096 9월 5 11:51 Desktop
-rwxrwxr-x 1 choijm choijm 4676 11월 19 2018 a.out
-rwxrwxr-x 1 choijm choijm 4880 3월 13 09:27 virt_cpu
-rw-r--r-- 1 choijm choijm 8980 8월 31 2018 examples.desktop
choijm@embedded:~$
choijm@embedded:~$ ls -l | sort -k5n | wc -l
11
choijm@embedded:~$
```



How to use commands in Linux (11/12)

Advanced commands: pipe, redirection and background

```
choijm@embedded: ~
choijm@embedded:~$ ls
a.out      examples.desktop  Public  test.c  virt_cpu
Desktop   OSTEP             Syspro  test.s  virt_cpu.c
choijm@embedded:~$
choijm@embedded:~$ man pipe
choijm@embedded:~$
choijm@embedded:~$ man pipe > man_pipe_output.txt
choijm@embedded:~$
choijm@embedded:~$ ls
a.out      examples.desktop  OSTEP  Syspro  test.s  virt_cpu.c
Desktop   man_pipe_output.txt  Public  test.c  virt_cpu
choijm@embedded:~$
choijm@embedded:~$ grep -o process man_pipe_output.txt | wc -l
4
choijm@embedded:~$ grep -o file man_pipe_output.txt | wc -l
7
choijm@embedded:~$ grep -o O_NONBLOCK man_pipe_output.txt | wc -l
2
choijm@embedded:~$
choijm@embedded:~$ grep -o process man_pipe_output.txt | wc -l &
[1] 4283
choijm@embedded:~$ 4

[1]+  Done                  grep --color=auto -o process man_pipe_output.txt |
wc -l
choijm@embedded:~$
choijm@embedded:~$ (grep -o process man_pipe_output.txt | wc -l) & (grep -o file
man_pipe_output.txt | wc -l) & (grep -o O_NONBLOCK man_pipe_output.txt | wc -l)
&
[1] 4290
[2] 4291
[3] 4292
choijm@embedded:~$ 4
7
2

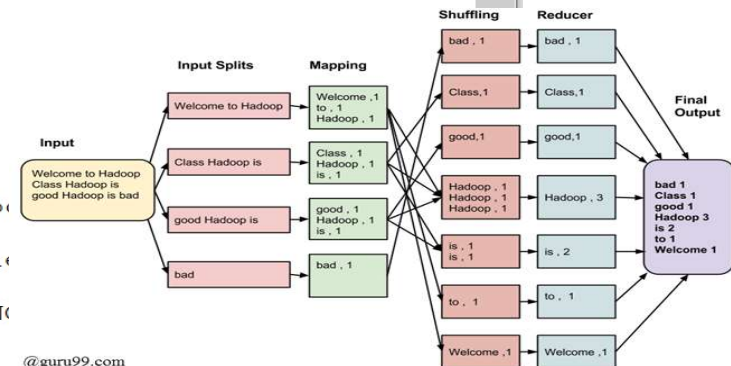
[1] Done
| wc -l )
[2]- Done
wc -l )
[3]+ Done
txt | wc -l )
choijm@embedded:~$
```

```
choijm@embedded: ~
PIPE(2)      Linux Programmer's Manual      PIPE(2)
NAME
pipe, pipe2 - create pipe
SYNOPSIS
#include <unistd.h>

int pipe(int pipefd[2]);

#define _GNU_SOURCE /* See feature_test_macros(7) */
#include <fcntl.h> /* Obtain O_* constant definitions */
#include <unistd.h>

int pipe2(int pipefd[2], int flags);
DESCRIPTION
pipe() creates a pipe, a unidirectional data channel that can be used
for interprocess communication. The array pipefd is used to return two
file descriptors referring to the ends of the pipe. pipefd[0] refers
to the read end of the pipe. pipefd[1] refers to the write end of the
pipe. Data written to the write end of the pipe is buffered by the
kernel until it is read from the read end of the pipe. For further
Manual page pipe(2) line 1 (press h for help or q to quit)
```



@guru99.com

How to use commands in Linux (12/12)

■ Generalization of file concept

- ✓ Treat device, socket, IPC as a file

```
choijm@embedded: ~  
choijm@embedded:~$ ps  
  PID TTY          TIME CMD  
22492 pts/9    00:00:00 bash  
22532 pts/9    00:00:00 ps  
choijm@embedded:~$ #include <stdio.h>  
  
main()  
{  
    printf("Hello DKU World\n");  
}
```

```
choijm@embedded: ~/programming  
choijm@embedded:~/programming$ ps  
  PID TTY          TIME CMD  
22561 pts/8    00:00:00 bash  
22610 pts/8    00:00:00 ps  
choijm@embedded:~/programming$ ls  
a.out hello.c README README new  
choijm@embedded:~/programming$ cat hello.c  
#include <stdio.h>  
  
main()  
{  
    printf("Hello DKU World\n");  
}  
choijm@embedded:~/programming$ cat hello.c > hello_backup.c  
choijm@embedded:~/programming$ more hello_backup.c  
#include <stdio.h>  
  
main()  
{  
    printf("Hello DKU World\n");  
}  
choijm@embedded:~/programming$ cat hello.c > /dev/pts/9  
choijm@embedded:~/programming$
```



How to make and run a program in Linux (1/7)

Overall

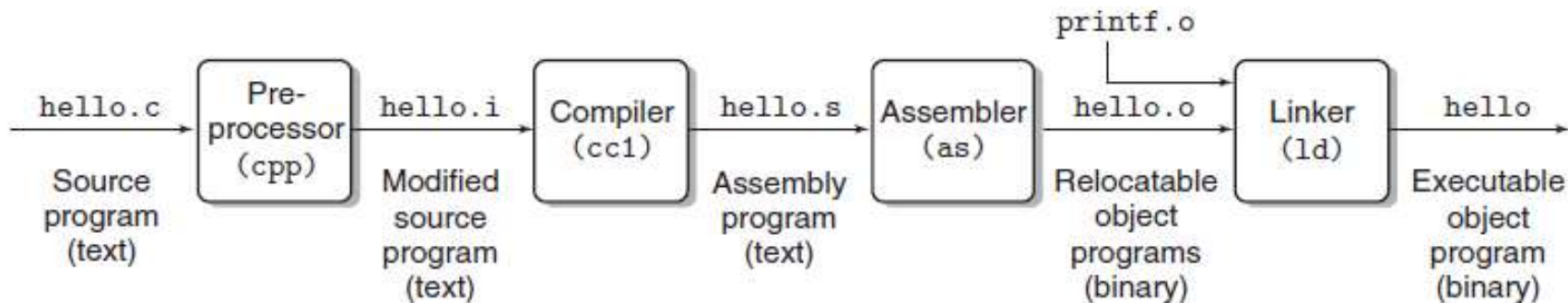


Figure 1.3 The compilation system.

(Source: computer systems: a programmer perspective, Figure 1.3)

```
choijm@embedded-desktop: ~/syspro/chap2
choijm@embedded-desktop:~/syspro/chap2$ vi test.c
choijm@embedded-desktop:~/syspro/chap2$
choijm@embedded-desktop:~/syspro/chap2$ ls
test.c
choijm@embedded-desktop:~/syspro/chap2$ more test.c
#include <stdio.h>

int a, b, c;

int main()
{
    a = 10;
    b = 20;
    c = a + b;
    printf("C = %d\n", c);
}
choijm@embedded-desktop:~/syspro/chap2$ gcc test.c
choijm@embedded-desktop:~/syspro/chap2$ ./a.out
C = 30
choijm@embedded-desktop:~/syspro/chap2$ gcc -o test.out test.c
choijm@embedded-desktop:~/syspro/chap2$ ./test.out
C = 30
choijm@embedded-desktop:~/syspro/chap2$

choijm@sungmin-Samsung-DeskTop-System: ~/syspro/chap2
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap2$ ls
test.c
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap2$ gcc -S test.c
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap2$ ls
test.c test.s
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap2$ as -o test.o test.s
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap2$ ls
test.c test.o test.s
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap2$ /usr/lib/gcc/i486-linux-gn
u/3.4.6/collect2 /usr/lib/i386-linux-gnu/crt1.o /usr/lib/i386-linux-gnu/crti.o /
usr/lib/i386-linux-gnu/crtn.o /usr/lib/gcc/i486-linux-gnu/3.4.6/crtbegin.o /usr
/lib/gcc//i486-linux-gnu/3.4.6/crtend.o test.o -lc -dynamic-linker /lib/ld-linux
.so.2
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap2$ ls
a.out test.c test.o test.s
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap2$ ./a.out
C = 30
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap2$
```

How to make and run a program in Linux (2/7)

■ Assembly code

```
choijm@embedded: ~/syspro18/chap2
choijm@embedded:~/syspro18/chap2$ gcc -S test.c
choijm@embedded:~/syspro18/chap2$ more test.s
.file "test.c"
.section .rodata
.LC0:
.string "C = %d\n"
.text
.globl main
.type main, @function
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $8, %esp
    andl   $-16, %esp
    movl   $0, %eax
    addl   $15, %eax
    addl   $15, %eax
    shrl   $4, %eax
    sall   $4, %eax
    subl   %eax, %esp
    movl   $10, a
    movl   $20, b
    movl   b, %eax
    addl   a, %eax
    movl   %eax, c
    movl   c, %eax
    movl   %eax, 4(%esp)
    movl   $.LC0, (%esp)
    call   printf
    leave
    ret
.size   main, .-main
.comm   a, 4, 4
.comm   b, 4, 4
.comm   c, 4, 4
.section .note.GNU-stack,"",@progbits
.ident "GCC: (GNU) 3.4.6 (Debian 3.4.6-5)"
choijm@embedded:~/syspro18/chap2$
```

```
choijm@DESKTOP-7SHQTVH: ~/Syspro23
.section .rodata
.LC0:
.string "C = %d\n"
.text
.globl main
.type main, @function
main:
.LFBO:
.cfi_startproc
endbr64
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6
movl    $10, a(%rip)
movl    $20, b(%rip)
movl    a(%rip), %edx
movl    b(%rip), %eax
addl    %edx, %eax
movl    %eax, c(%rip)
movl    c(%rip), %eax
movl    %eax, %esi
leaq   .LC0(%rip), %rdi
movl   $0, %eax
call   printf@PLT
movl   $0, %eax
popq   %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size   main, .-main
.ident "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"
.section .note.GNU-stack,"",@progbits
.section .note.gnu.property,"a"
```

C = A + B;

```
...
movl 0x8049388, %eax
addl 0x8049384, %eax
movl %eax, 0x804946c
...
```

```
...
00a1 8893 0408
0305 8493 0408
00a3 6c94 0408
...
```

(Language hierarchy)

Can be different based on the version of kernel and compiler

How to make and run a program in Linux (3/7)

■ Relocatable code

- ✓ Hexdump (or xxd), objdump

```
choijm@embedded-desktop: ~/syspro/chap2
choijm@embedded-desktop:~/syspro/chap2$ ls
S.OBJ test.c test.o test.s
choijm@embedded-desktop:~/syspro/chap2$
choijm@embedded-desktop:~/syspro/chap2$ more test.o

***** test.o: Not a text file *****

choijm@embedded-desktop:~/syspro/chap2$
choijm@embedded-desktop:~/syspro/chap2$ hexdump test.o
00000000 457f 464c 0101 0001 0000 0000 0000 0000
00000010 0001 0003 0001 0000 0000 0000 0000 0000
00000020 0110 0000 0000 0000 0034 0000 0000 0028
00000030 000b 0008 8955 83e5 08ec e483 b8f0 0000
00000040 0000 c083 830f 0fc0 e8c1 c104 04e0 c429
00000050 05c7 0000 0000 000a 0000 05c7 0000 0000
00000060 0014 0000 00a1 0000 0300 0005 0000 a300
00000070 0000 0000 00a1 0000 8900 2444 c704 2404
00000080 0000 0000 fce8 ffff c9ff 00c3 2043 203d
00000090 6425 000a 4700 4343 203a 4728 554e 2029
000000a0 2e33 2e34 2036 5528 7562 746e 2075 2e33
000000b0 2e34 2d36 7536 7562 746e 3575 0029 2e00
000000c0 7973 746d 6261 2e00 7473 7472 6261 2e00
000000d0 6873 7473 7472 6261 2e00 6572 2e6c 6574
000000e0 7478 2e00 6164 6174 2e00 7362 0073 722e
000000f0 646f 7461 0061 6e2e 746f 2e65 4e47 2d55
00001000 7473 6361 006b 632e 6d6f 656d 746e 0000
00001100 0000 0000 0000 0000 0000 0000 0000 0000
*
00001130 0000 0000 0000 0000 001f 0000 0001 0000
00001140 0006 0000 0000 0000 0034 0000 0057 0000
00001150 0000 0000 0000 0000 0004 0000 0000 0000
00001160 001b 0000 0009 0000 0000 0000 0000 0000
00001170 03b4 0000 0040 0000 0009 0000 0001 0000
00001180 0004 0000 0008 0000 0025 0000 0001 0000
```

```
choijm@embedded-desktop: ~/syspro/chap2
S.OBJ test.c test.o test.s
choijm@embedded-desktop:~/syspro/chap2$ objdump -f test.o

test.o:      file format elf32-i386
architecture: i386, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000

choijm@embedded-desktop:~/syspro/chap2$ objdump -d test.o

test.o:      file format elf32-i386

Disassembly of section .text:

00000000 <main>:
 0: 55                push   %ebp
 1: 89 e5             mov    %esp,%ebp
 3: 83 ec 08         sub   $0x8,%esp
 6: 83 e4 f0         and   $0xfffffff0,%esp
 9: b8 00 00 00 00   mov   $0x0,%eax
 e: 83 c0 0f         add   $0xf,%eax
11: 83 c0 0f         add   $0xf,%eax
14: c1 e8 04         shr   $0x4,%eax
17: c1 e0 04         shl   $0x4,%eax
1a: 29 c4            sub   %eax,%esp
1c: c7 05 00 00 00 0a movl  $0xa,0x0
23: 00 00 00         movl  0x0,0x0
26: c7 05 00 00 00 14 movl  $0x14,0x0
2d: 00 00 00         movl  0x0,0x0
30: a1 00 00 00 00   mov   0x0,%eax
35: 03 05 00 00 00 00 add   0x0,%eax
3b: a3 00 00 00 00   mov   %eax,0x0
40: a1 00 00 00 00   mov   0x0,%eax
45: 89 44 24 04     mov   %eax,0x4(%esp)
49: c7 04 21 00 00 00 movl  $0x0,(%esp)
50: e8 fc ff ff ff   call  51 <main+0x51>
55: c9              leave
56: c3              ret
choijm@embedded-desktop:~/syspro/chap2$
```

How to make and run a program in Linux (4/7)

■ Executable code

```
choijm@embedded-desktop: ~/syspro/chap2
choijm@embedded-desktop:~/syspro/chap2$ ls
a.out test.c test.o test.s
choijm@embedded-desktop:~/syspro/chap2$ hexdump a.out
00000000 457f 464c 0101 0001 0000 0000 0000 0000
00000010 0002 0003 0001 0000 8318 0804 0034 0000
00000020 0680 0000 0000 0000 0034 0020 0007 0028
00000030 0019 0016 0006 0000 0034 0000 8034 0804
00000040 8034 0804 00e0 0000 00e0 0000 0005 0000
00000050 0004 0000 0003 0000 0114 0000 8114 0804
00000060 8114 0804 0013 0000 0013 0000 0004 0000
00000070 0001 0000 0001 0000 0000 0000 8000 0804
00000080 8000 0804 0480 0000 0480 0000 0005 0000
00000090 1000 0000 0001 0000 0480 0000 9480 0804
000000a0 9480 0804 00e8 0000 00f4 0000 0006 0000
000000b0 1000 0000 0002 0000 0480 0000 9480 0804
000000c0 9480 0804 00c8 0000 00c8 0000 0006 0000
000000d0 0004 0000 0004 0000 0128 0000 8128 0804
000000e0 8128 0804 0020 0000 0020 0000 0004 0000
000000f0 0004 0000 e551 6474 0000 0000 0000 0000
00000100 0000 0000 0000 0006 0000
00000110 6269 6c2f 2d64 696c 756e
00000120 0000 0004 0000 0010 0000
00000130 0055 0000 0000 0002 0000
00000140 0000 0003 0000 0005 0000
00000150 0000 0003 0000 0000 0000
00000160 0000 0001 0000 0000 0000
00000170 0000 0000 0000 0000 0000
00000180 0000 0000 0000 0012 0000
00000190 0000 0000 0000 0020 0000
000001a0 0804 0004 0000 0011 000e
000001b0 0000 0000 0000 0012 0000
000001c0 6f73 362e 5f00 4f49 735f
000001d0 6573 0064 7270 6e69 6674
000001e0 5f63 7473 7261 5f74 616d
000001f0 6f6d 5f6e 7473 7261 5f74
00000200 005f 4c47 4249 5f43 2e32 0030 0000 0002
00000210 0000 0001 0002 0000 0001 0001 0001 0000
00000220 0010 0000 0000 0000 6910 0d69 0000 0002
00000230 0042 0000 0000 0000 9548 0804 0206 0000
00000240 9558 0804 0107 0000 955c 0804 0207 0000
```

C = A + B;

...
movl 0x8049388, %eax
addl 0x8049384, %eax
movl %eax, 0x804946c
...

...
00a1 8893 0408
0305 8493 0408
00a3 6c94 0408
...

(Language hierarchy)

```
choijm@embedded-desktop: ~/syspro/chap2
choijm@embedded-desktop:~/syspro/chap2$ objdump -f a.out
a.out:      file format elf32-i386
architecture: i386, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x08048318

choijm@embedded-desktop:~/syspro/chap2$ objdump -d a.out > objdump_result.txt
choijm@embedded-desktop:~/syspro/chap2$ vi objdump_result.txt
choijm@embedded-desktop:~/syspro/chap2$ more objdump_result.txt
...
...

Disassembly of section .text:

080482c0 <main>:
080482c0:      55                push   %ebp
080482c1:      89 e5             mov    %esp,%ebp
080482c3:      83 ec 08          sub   $0x8,%esp
080482c6:      83 e4 f0          and   $0xfffffff0,%esp
080482c9:      b8 00 00 00 00    mov   $0x0,%eax
080482ce:      83 c0 0f          add   $0xf,%eax
080482d1:      83 c0 0f          add   $0xf,%eax
080482d4:      c1 e8 04          shr   $0x4,%eax
080482d7:      c1 e0 04          shl   $0x4,%eax
080482da:      29 c4             sub   %eax,%esp
080482dc:      c7 05 70 95 04 08 0a movl  $0xa,0x8049570
080482e3:      00 00 00
080482e6:      c7 05 68 95 04 08 14 movl  $0x14,0x8049568
080482ed:      00 00 00
080482f0:      a1 68 95 04 08    mov   0x8049568,%eax
080482f5:      03 05 70 95 04 08 add   0x8049570,%eax
080482fb:      a3 6c 95 04 08    mov   %eax,0x804956c
08048300:      a1 6c 95 04 08    mov   0x804956c,%eax
08048305:      89 44 24 04       mov   %eax,0x4(%esp)
08048309:      c7 04 24 d0 83 04 08 movl  $0x80483d0,(%esp)
08048310:      e8 7b ff ff ff   call  8048290 <printf@plt>
08048315:      c9                leave
08048316:      c3                ret
08048317:      90                nop
```

How to make and run a program in Linux (5/7)

- What are the execution results of this program?

```
choijm@embedded-desktop: ~/syspro/gdb_exam
choijm@embedded-desktop:~/syspro/gdb_exam$
choijm@embedded-desktop:~/syspro/gdb_exam$ vi gdb_test.c
choijm@embedded-desktop:~/syspro/gdb_exam$
choijm@embedded-desktop:~/syspro/gdb_exam$ cat gdb_test.c
#include <stdio.h>

int a[4] = {5, 6, 7, 8};
int *pa;

main()
{
    printf("%d\n", a[0]);
    printf("%d\n", a[2]);
    printf("%d\n", *a);
    printf("%d\n", *(a+2));
    printf("%d\n", *pa);
    printf("%d\n", *(pa+2));
}
choijm@embedded-desktop:~/syspro/gdb_exam$
choijm@embedded-desktop:~/syspro/gdb_exam$
```



How to make and run a program in Linux (6/7)

■ debugger

```
choijm@embedded-desktop: ~/syspro/gdb_exam
choijm@embedded-desktop:~/syspro/gdb_exam$ vi gdb_test.c
choijm@embedded-desktop:~/syspro/gdb_exam$ cat gdb_test.c
#include <stdio.h>

int a[4] = {5, 6, 7, 8};
int *pa;

main()
{
    printf("%d\n", a[0]);
    printf("%d\n", a[2]);
    printf("%d\n", *a);
    printf("%d\n", *(a+2));
    printf("%d\n", *pa);
    printf("%d\n", *(pa+2));
}
choijm@embedded-desktop:~/syspro/gdb_exam$
choijm@embedded-desktop:~/syspro/gdb_exam$ gcc -o gdb_test.out gdb_test.c
choijm@embedded-desktop:~/syspro/gdb_exam$ ./gdb_test.out
5
7
5
7
Segmentation 오류 (core dumped)
choijm@embedded-desktop:~/syspro/gdb_exam$
choijm@embedded-desktop:~/syspro/gdb_exam$
```

```
choijm@embedded-desktop: ~/syspro/gdb_exam
choijm@embedded-desktop:~/syspro/gdb_exam$ gcc -g -o gdb_test.out gdb_test.c
choijm@embedded-desktop:~/syspro/gdb_exam$ gdb gdb_test.out
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/choijm/syspro/gdb_exam/gdb_test.out...done.
(gdb) run
Starting program: /home/choijm/syspro/gdb_exam/gdb_test.out
warning: no loadable sections found in added symbol-file system-supplied DSO at
0x7ffff7ffa000
5
7
5
7
Program received signal SIGSEGV, Segmentation fault.
0x000000000400567 in main () at gdb_test.c:12
12      printf("%d\n", *pa);
(gdb) list
7      {
8          printf("%d\n", a[0]);
9          printf("%d\n", a[2]);
10         printf("%d\n", *a);
11         printf("%d\n", *(a+2));
12         printf("%d\n", *pa);
13         printf("%d\n", *(pa+2));
14     }
(gdb)
Line number 15 out of range; gdb_test.c has 14 lines.
(gdb) break 10
Breakpoint 1 at 0x40052c: file gdb_test.c, line 10.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/choijm/syspro/gdb_exam/gdb_test.out
warning: no loadable sections found in added symbol-file system-supplied DSO at
0x7ffff7ffa000
5
7
Breakpoint 1, main () at gdb_test.c:10
10      printf("%d\n", *a);
(gdb) n
11      printf("%d\n", *(a+2));
(gdb) n
```

☞ There are various valuable debugger commands such as breakpoint, step, next, info reg, ... → See <http://beej.us/guide/bggdb/> 39

How to make and run a program in Linux (7/7)

■ Make utility

- ✓ Why? Using multiple files → 1) complex gcc command, 2) dependency
- ✓ Makefile format
- ✓ Makefile example

**target : dependency1 dependency2
command1
command2**

```
choijm@embedded: ~/Syspro/make_slide
choijm@embedded:~/Syspro/make_slide$ ls
main.c member1.c member2.c myheader.h
choijm@embedded:~/Syspro/make_slide$ cat member1.c
void func1(void)
{
    printf("This code is written by Member 1\n");
}
choijm@embedded:~/Syspro/make_slide$ cat member2.c
void func2(void)
{
    printf("This code is written by Member 2\n");
}
choijm@embedded:~/Syspro/make_slide$ cat main.c
#include "myheader.h"

int main()
{
    func1();
    func2();
    printf("Here is written by Member 3\n");
}
choijm@embedded:~/Syspro/make_slide$ cat myheader.h
#include <stdio.h>
void func1(void);
void func2(void);
choijm@embedded:~/Syspro/make_slide$ gcc -o project main.c member1.c member2.c
choijm@embedded:~/Syspro/make_slide$ ./project
This code is written by Member 1
This code is written by Member 2
Here is written by Member 3
choijm@embedded:~/Syspro/make_slide$ touch member1.c
choijm@embedded:~/Syspro/make_slide$ gcc -o project main.c member2.c
```

```
choijm@embedded: ~/Syspro/make_slide
choijm@embedded:~/Syspro/make_slide$ ls
main.c Makefile member1.c member2.c myheader.h
choijm@embedded:~/Syspro/make_slide$ cat Makefile
CC = gcc
RM = rm
TARGET = project
OBJECTS = main.o member1.o member2.o

all : $(TARGET)

$(TARGET) : $(OBJECTS)
    $(CC) -o $$@ ^

clean:
    $(RM) -f $(TARGET) $(OBJECTS)
choijm@embedded:~/Syspro/make_slide$ make
gcc -c -o main.o main.c
gcc -c -o member1.o member1.c
gcc -c -o member2.o member2.c
gcc -o project main.o member1.o member2.o
choijm@embedded:~/Syspro/make_slide$ ls
main.c Makefile member1.o member2.o project
main.o member1.c member2.c myheader.h
choijm@embedded:~/Syspro/make_slide$ ./project
This code is written by Member 1
This code is written by Member 2
Here is written by Member 3
choijm@embedded:~/Syspro/make_slide$ touch member1.c
choijm@embedded:~/Syspro/make_slide$ make
gcc -c -o member1.o member1.c
gcc -o project main.o member1.o member2.o
choijm@embedded:~/Syspro/make_slide$ make clean
rm -f project main.o member1.o member2.o
choijm@embedded:~/Syspro/make_slide$ ls
main.c Makefile member1.c member2.c myheader.h
choijm@embedded:~/Syspro/make_slide$
```


Summary

- Discuss the features of Linux
- Understand the commands related to file and process
- Explore the language hierarchy in Linux (UNIX)

☞ Homework 2.

1.1 Make a file using vi editor that contains your favorite poem

1.2 Make a snapshot that

- has at least 10 commands (e.g. ls -l, ps, pipe, redirection, ...) including compilation practice (e.g. gcc, as, gdb, ...)
- shows student's ID and date (using whoami and date)
- Server IP: 220.149.236.2 or 220.149.236.4

1.3 Write a report

- 1) Introduction: What to do, How, ...
- 2) Snapshot for 1.1,
- 3) Snapshot for 1.2,
- 4) Discussion: what you learn, issues, ...

1.4 Deadline: Next week (same time)

1.5 How to submit? Email to mgchoi@dankook.ac.kr



Appendix 1. Snapshot Example

■ Example

```
choijm@embedded: ~/Syspro/chap2/reports
choijm@embedded:~/Syspro/chap2$
choijm@embedded:~/Syspro/chap2$ mkdir reports
choijm@embedded:~/Syspro/chap2$
choijm@embedded:~/Syspro/chap2$ cd reports/
choijm@embedded:~/Syspro/chap2/reports$
choijm@embedded:~/Syspro/chap2/reports$ ls
choijm@embedded:~/Syspro/chap2/reports$
choijm@embedded:~/Syspro/chap2/reports$ vi my_favorite_poem.txt
choijm@embedded:~/Syspro/chap2/reports$
choijm@embedded:~/Syspro/chap2/reports$ ls
my_favorite_poem.txt
choijm@embedded:~/Syspro/chap2/reports$
choijm@embedded:~/Syspro/chap2/reports$ ls -l
total 4
-rw-rw-r-- 1 choijm choijm 339  9월  9 16:37 my_favorite_poem.txt
choijm@embedded:~/Syspro/chap2/reports$
choijm@embedded:~/Syspro/chap2/reports$ cat my_favorite_poem.txt
나 하늘로 날아가리라
새벽빛 와 닿으면 스러지는
이슬 더불어 손에 손을 잡고

나 하늘로 날아가리라
노을빛 함께 만 들이서
기슭에서 날다가 구름 손짓하면은

나 하늘로 날아가리라
아름다운 이 세상 소름 핀네는 날
가서, 아름답게 되라고 말하리라.....

choijm@embedded:~/Syspro/chap2/reports$
choijm@embedded:~/Syspro/chap2/reports$ whoami
choijm
choijm@embedded:~/Syspro/chap2/reports$
```

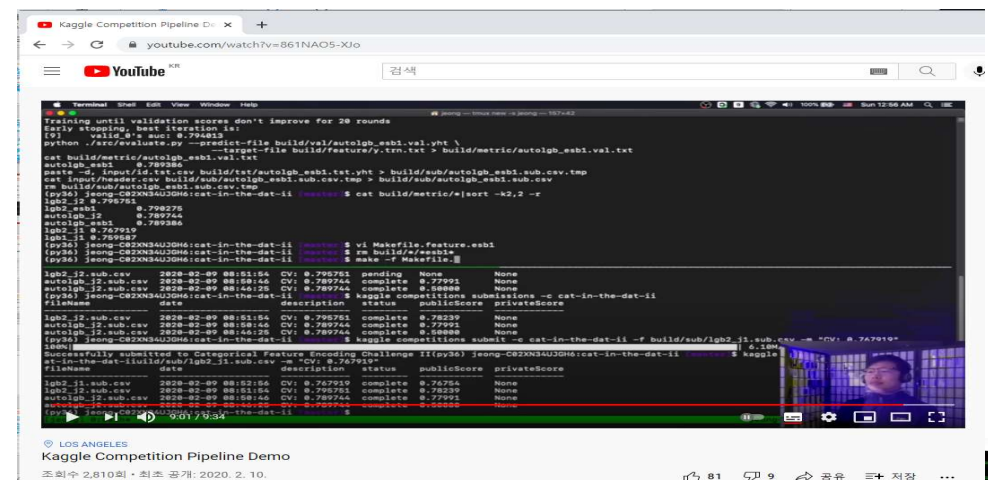
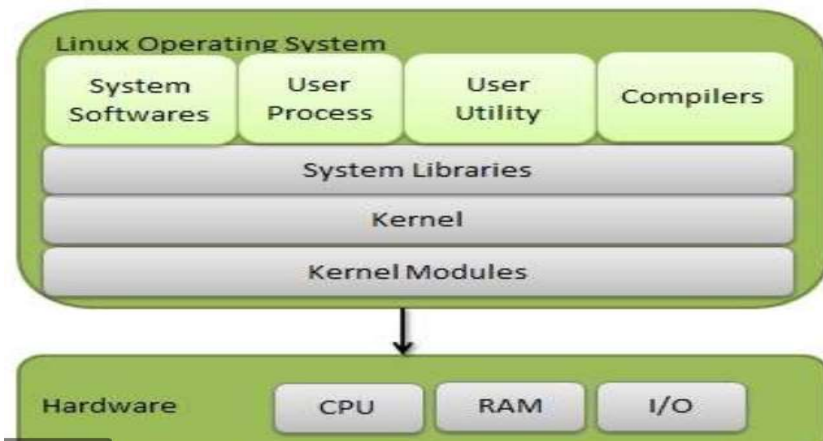
```
choijm@embedded: ~/Syspro/chap2
choijm@embedded:~/Syspro/chap2$ ps
  PID TTY          TIME CMD
 6334 pts/12    00:00:00 bash
 6702 pts/12    00:00:00 ps
choijm@embedded:~/Syspro/chap2$ vi test.c
choijm@embedded:~/Syspro/chap2$ gcc -g -o test.out test.c
choijm@embedded:~/Syspro/chap2$
choijm@embedded:~/Syspro/chap2$ gdb test.out
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from test.out...done.
(gdb) run
Starting program: /home/choijm/Syspro/chap2/test.out
C = 30
[Inferior 1 (process 6711) exited with code 07]
(gdb) list
1      #include <stdio.h>
2
3      int a, b, c;
4
5      int main(int argc, char *argv[])
6      {
7          a = 10;
8          b = 20;
9          c = a + b;
10         printf("C = %d\n", c);
(gdb) quit
choijm@embedded:~/Syspro/chap2$ gcc -S test.c
choijm@embedded:~/Syspro/chap2$
choijm@embedded:~/Syspro/chap2$ whoami
choijm
choijm@embedded:~/Syspro/chap2$ date
2023. 09. 09. ( ㉮ ) 16:48:17 KST
choijm@embedded:~/Syspro/chap2$ man pipe | grep process | wc -l
4
choijm@embedded:~/Syspro/chap2$
```



Quiz for this Lecture

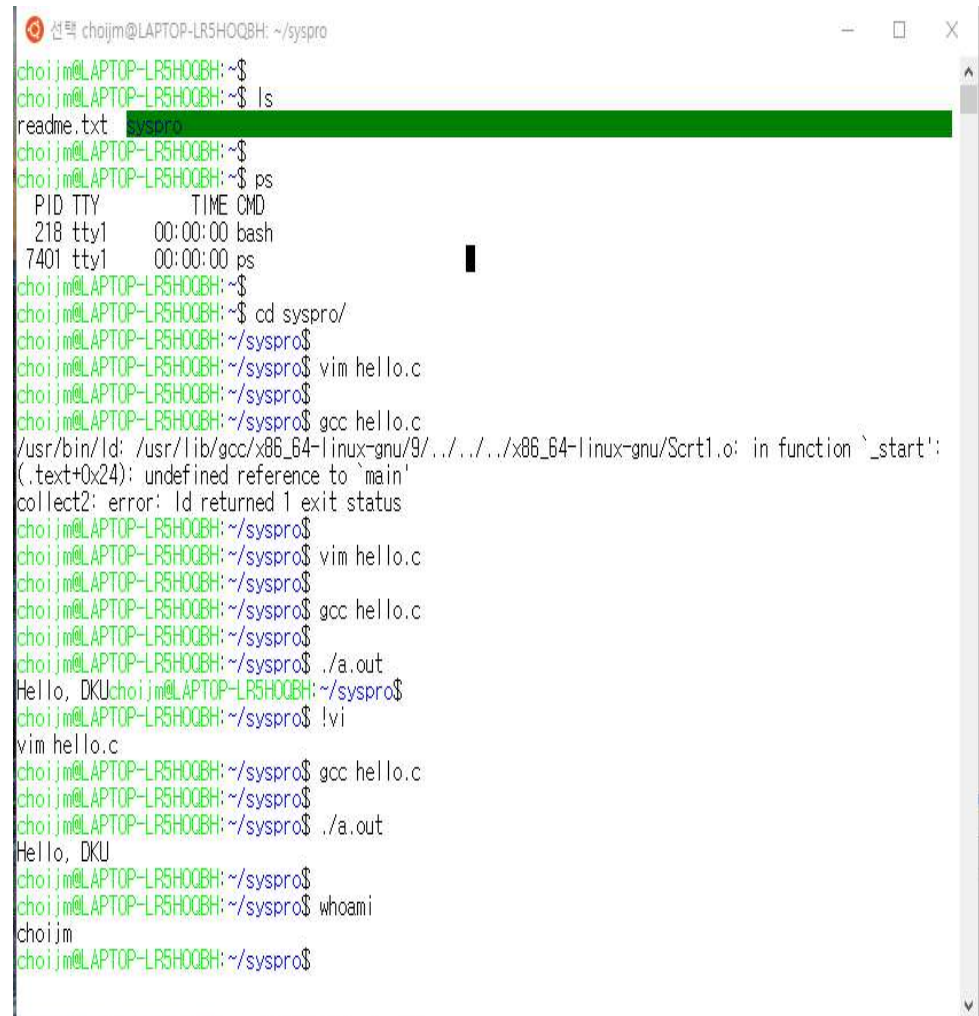
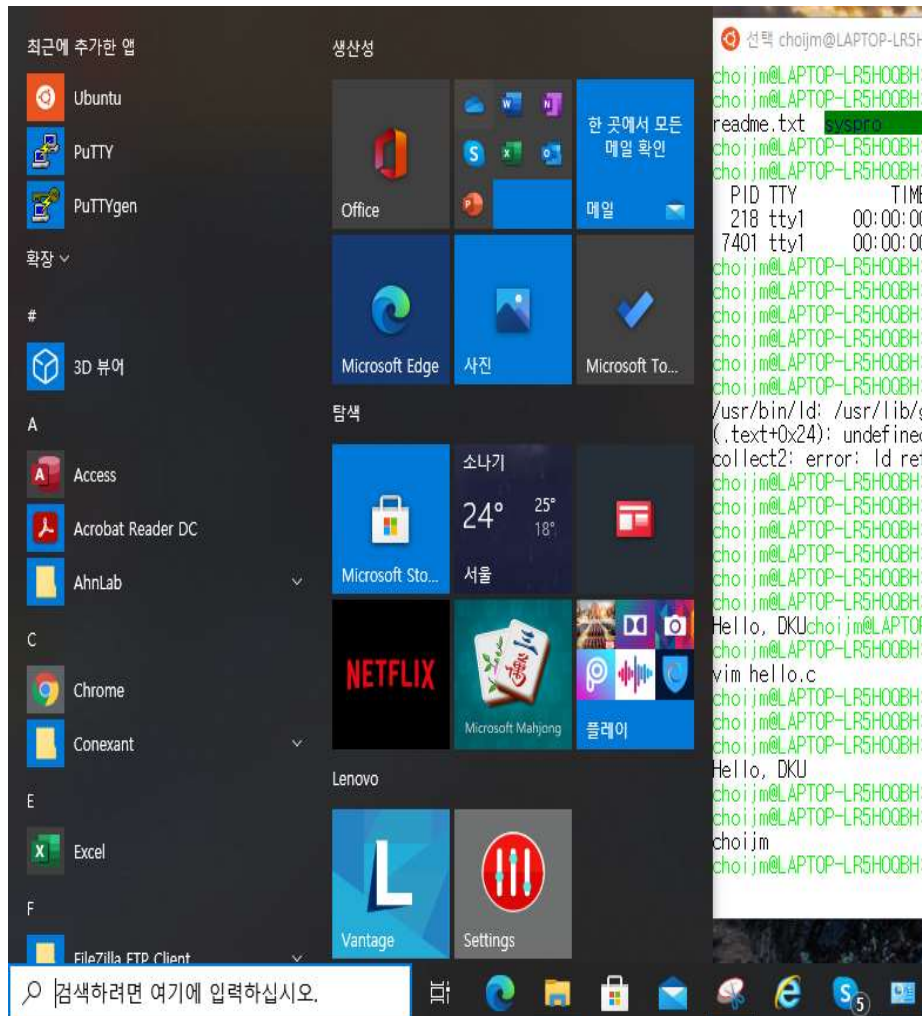
■ Quiz

- ✓ 1. Discuss the difference between OS (Operating System) and Kernel using the below left figure.
- ✓ 2. Explain differences between “\$ls .” and “\$ls ..”. Also, explain differences between “ls” and “ls -l”.
- ✓ 3. What is the background music in “Dr Jeong-Joon Lee’s Kaggle Demo”? What commands can you find in the Kaggle Demo? (at least 5 that you have learned in the LN2.)
- ✓ 4. Discuss three different modes in the vi editor.
- ✓ 5. What are the roles of “break” and “step” command in gdb?



Appendix 2: How to access Linux: Alternative

- WSL (Windows Subsystem for Linux)
 - ✓ A compatibility layer for running Linux binary executables (in ELF format) natively on Windows OS



Appendix 2: How to access Linux: Alternative

- Toast Cloud (or Amazon EC2 or Google)
 - ✓ Supported by NHN (like Amazon EC2 or Google Compute Engine)
 - ✓ Using PaaS in this course
 - IP: 133.186.152.119 (May be different per each VM instance)
 - For general users: same as the Linux server

