# Lab Session 1

## Introduction

Computer Security Lab

Name: Suhwan Song

Email: sshkeb96@snu.ac.kr

# Contents

# Contents

# Install Docker

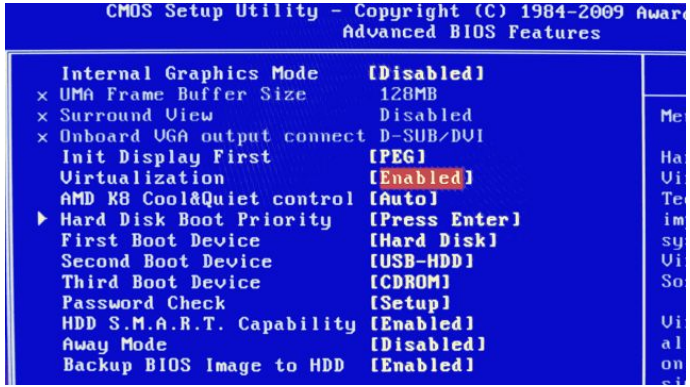
- You can run Docker on most of OSes
- Pick any OS which works best for you (+ architecture)
  - Window
  - Mac OS (x86)
  - Apple silicon
  - ~~Linux~~

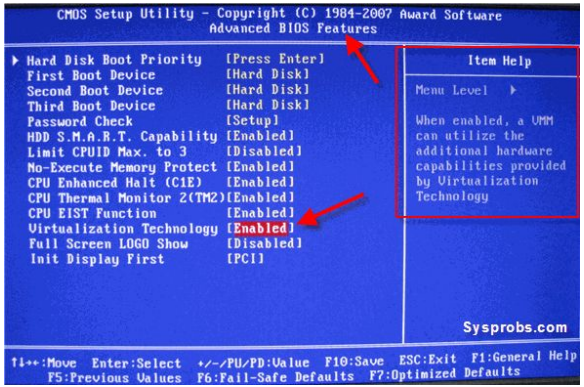
# [Window] Install WSL

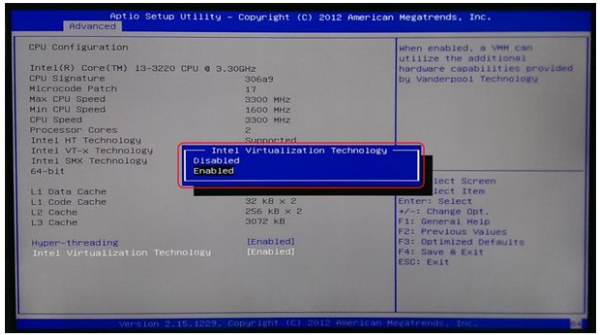
- Docker uses the WSL2 or Hyper-V
  - But, Hyper-V is supported for Windows Pro user. So we use the WSL.
- For installing WSL, Virtualization must be enabled in BIOS.
- To check whether Virtualization is enabled,
  - open “PowerShell” or “Windows Terminal”
  - Type “systeminfo.exe”
    - Check the **“Virtualization Enabled in Firmware”** value is set to true
- If the virtualization is disabled, you need to enable the VT-x(Intel) or SVM(AMD) feature in BIOS.

# [Window] Install WSL - Enable Virtualization

- The method of turning on the virtualization is different for each motherboard manufacturer, so it may be different from the picture below.
  - Search on Google by “How to enable VT-x/SVM in <MOTHERBOARD VENDOR / Laptop Model>”

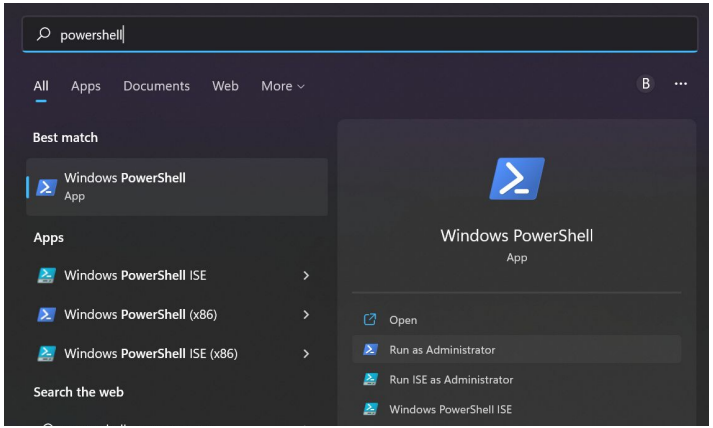


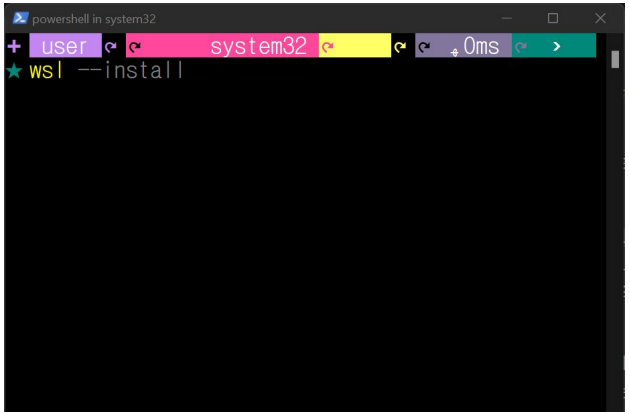



# [Window] Install WSL

- Run PowerShell **as an administrator**
- Enter “wsl --install” in PowerShell

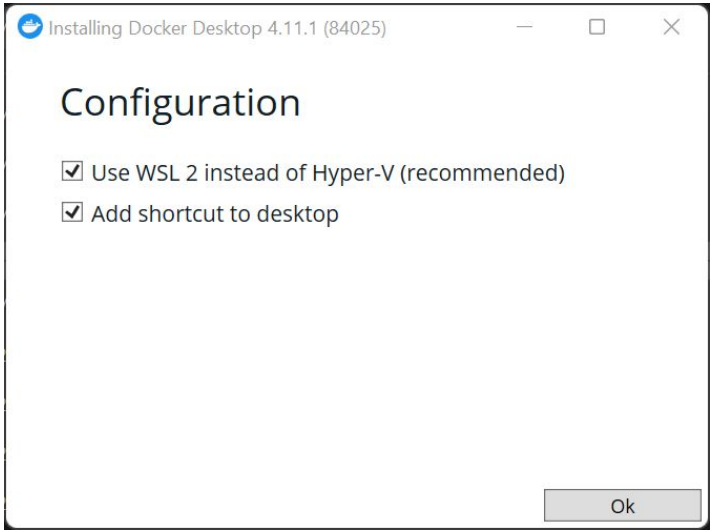



https://docs.microsoft.com/ko-kr/windows/wsl/install

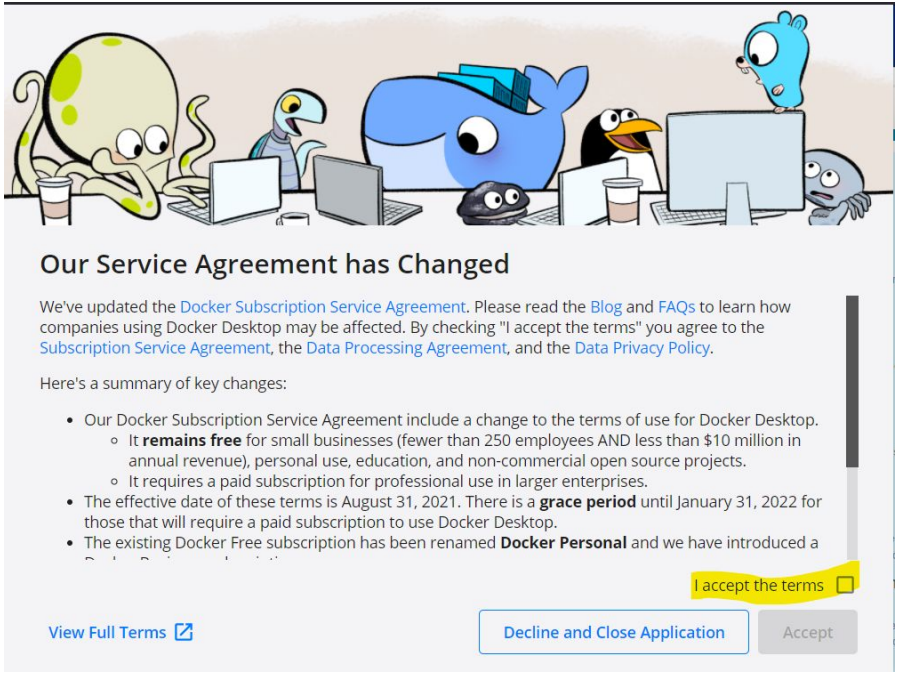
# [Window] Install Docker

- Download and install “Docker Desktop”

# Docker

- Open Docker and accept the agreement



# Install Linux in Docker

- Build a docker image

```
> docker build -t compsec ./
```

- Create a docker container (**DO NOT CHANGE** HOSTNAME(class-sp)!)

```
> docker run -d -h class-sp --name <CONTAINER_NAME> --privileged -it compsec
```

- Start a container named <CONTAINER_NAME> and connect it

```
> docker start <CONTAINER_NAME>
> docker attach <CONTAINER_NAME>
```

docker command-lines

# Install Linux in Docker

```
powershell in Downloads
user  Downloads  0ms  docker build -t compsec ./                                  powershell  16:17:25
[+] Building 3.0s (12/12) FINISHED
 => [internal] load build definition from Dockerfile                                                    0.0s
 => => transferring dockerfile: 32B                                                                     0.0s
 => [internal] load .dockerignore                                                                       0.0s
 => => transferring context: 2B                                                                         0.0s
 => [internal] load metadata for docker.io/library/ubuntu:20.04                                         2.9s
 => [1/8] FROM docker.io/library/ubuntu:20.04@sha256:35ab2bf57814e9ff49e365efd5a5935b6915eede5c7f8581e9e1b85e0eec  0.0s
 => CACHED [2/8] RUN apt-get update  && apt-get install -y sudo                                         0.0s
 => CACHED [3/8] RUN adduser --disabled-password --gecos '' compsec                                     0.0s
 => CACHED [4/8] RUN adduser compsec sudo                                                               0.0s
 => CACHED [5/8] RUN echo '%sudo ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers                                0.0s
 => CACHED [6/8] WORKDIR /home/compsec                                                                  0.0s
 => CACHED [7/8] RUN sudo apt-get update                                                                0.0s
 => CACHED [8/8] RUN sudo apt-get install build-essential gcc g++ vim git -y                            0.0s
 => exporting to image                                                                                  0.0s
 => => exporting layers                                                                                 0.0s
 => => writing image sha256:3d6f82792e330542e001c65d5f5d4792cd3d2aae333887fa4c2d77ce49c5e0d3            0.0s
 => => naming to docker.io/library/compsec                                                              0.0s
```

```
compsec@72be243a9209: ~
user  Downloads  0ms  docker run -d --name comp -p 22:22 --privileged -it compsec:23:17
72be243a92095883ca020c9b12e981aa81b5cb685d11462ca21d3b143260bbb4
user  Downloads  714ms  docker start comp                                  powershell  16:23:23
comp
user  Downloads  129ms  docker attach comp                                 powershell  16:23:26
compsec@72be243a9209:~$
```

# Contents

# Linux Environment

- Bash Shell
  - One of the default user interface.
  - Every Linux User has home directory.

- ~ means current user’s home directory

# Absolute Path / Relative Path

- Linux uses two ways for representing location.

- Absolute Path
  - points to location regardless of the current working directory.
  - it must include the root directory.

- Relative Path
  - path starts from current working directory.
  - **.** means current working directory. (You can check your current working directory using `pwd` command.)
  - **..** means parent directory.

# Some Bash Commands

| Command | Description |
|---|---|
| pwd | print absolute path of current location |
| ls | list current location |
| touch [new file name] | make new empty file |
| mkdir [new directory name] | make new directory |
| cd [location] | move to another location |
| cp [source] [destination] | copy source file to destination |
| cp –r [source] [destination] | copy source folder to destination |
| mv [source] [destination] | move source file to destination |
| mv –r [source] [destination] | move source folder to destination |
| rm [file name] | remove file |
| rm –rf [folder name] | remove folder and files inside of folder |

# Bash Command Example

```
compsec@class-sp:~$ mkdir my_dir
compsec@class-sp:~$ ls
my_dir
compsec@class-sp:~$ cd my_dir
compsec@class-sp:~/my_dir$ pwd
/home/compsec/my_dir
compsec@class-sp:~/my_dir$ touch my_file
compsec@class-sp:~/my_dir$ ls
compsec@class-sp:~/my_dir$ cd ../
```

# Bash Command Example

```
compsec@class-sp:~$ ls ./my_dir
my_file
compsec@class-sp:~$ rm ./my_dir/my_file
compsec@class-sp:~$ ls ./my_dir

compsec@class-sp:~$ ls
my_dir
compsec@class-sp:~$ rm -rf ./my_dir
compsec@class-sp:~$ ls
```

# man command

- If you want to check **“manual”** of commands, use man!
- $ man [COMMAND_NAME]
- Example

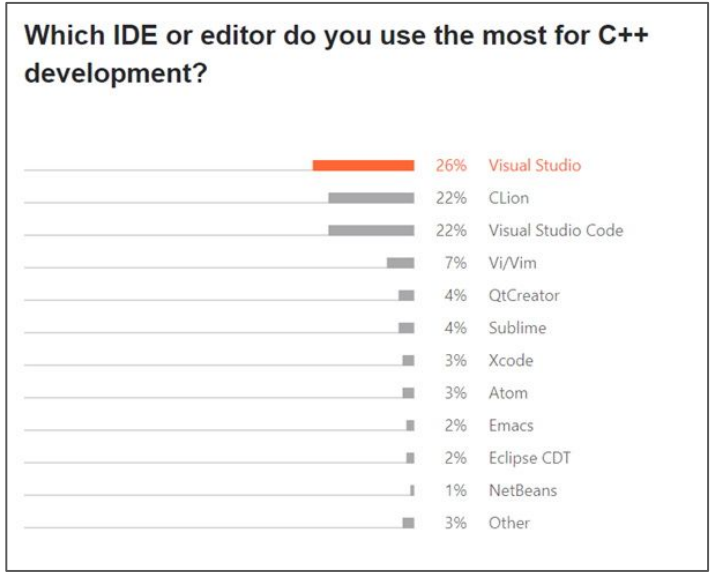
man ls

man cp

...

```
LS(1)                        General Commands Manual                        LS(1)

NAME
     ls – list directory contents

SYNOPSIS
     ls [-@ABCFGHILOPRSTUWabcdefghiklmnopqrstuvwxy1%,] [--color=when]
        [-D format] [file ...]

DESCRIPTION
     For each operand that names a file of a type other than directory, ls
     displays its name as well as any requested, associated information.  For
     each operand that names a file of type directory, ls displays the names of
     files contained within that directory, as well as any requested, associated
     information.

     If no operands are given, the contents of the current directory are
     displayed.  If more than one operand is given, non-directory operands are
     displayed first; directory and non-directory operands are sorted separately
     and in lexicographical order.

     The following options are available:
```

# Text editor in Linux




https://www.incredibuild.com/blog/best-c-ides

- Which editor do you use for coding?

- If you haven't learn any of terminal-based editor, we recommend you learn "VIM" as a start.
  - You can use any other alternatives if you'd like.

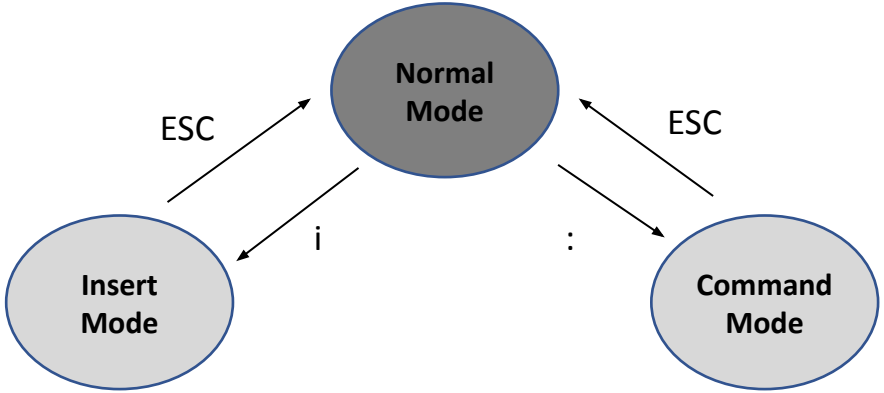
# Vim

- **V**i **IM**proved

- Mode-based mechanism
  - When you execute vim, start at normal mode.
  - If you want to edit, switch to insert mode.
  - If you want to select some area, switch to visual mode.
  - If you command, switch to command line mode.

# Let’s use Vim!

| Command | Description |
|---|---|
| i | Insert text before the cursor. |
| :w | Write the current file |
| :q | Quit Vim. |
| :wq | Write the current file and quit vim. |



# Let’s use Vim!

| Command | Description |
|---|---|
| yy | Yank (copy) a line |
| dd | Delete (cut) a line |
| p | Put (paste) the clipboard after cursor |
| u | Undo |
| Ctrl-r | Redo |

# Contents

# GDB

- The debugger
  - Allows you to see what is going on “inside” another program while it executes.

- It can do four main kinds of things:
  - Start your program, specifying anything that might affect its behavior.
  - Make your program stop on specified conditions.
  - Examine what has happened, when your program has stopped.
  - Change things in your program for experiments.

https://www.sourceware.org/gdb/

# GDB Usage

| Command | Description |
|---|---|
| break [b] | Puts a breakpoint at function, line of source code and etc... |
| delete [d] | Deletes a breakpoint |
| run [r] | Executes the program until a breakpoint or error |
| next [n] | Execute the current instruction and stop execution before the next instruction. |
| continue [c] | Continues running the program until the next breakpoint or error |
| info register | Prints a register |
| list [l] | Prints out some lines from the source code |
| quit [q] | Quit GDB |

# GDB Example

- Example code
  - This code computes the factorial of a number erroneously.
  - You can pinpoint the reason of the error by using GDB.
  - Write the code in main.c and start debugging

- Starting GDB

```
compsec@your-pc:~$ sudo apt-get install gdb
compsec@your-pc:~$ gcc -g -no-pie main.c
compsec@your-pc:~$ gdb ./a.out
```

```c
#include <stdio.h>

int main() {
    int i;
    for (i = 0;i < 10; i++) {
        printf("Hello World\n");
    }
    return 0;
}
```

main.c

# GDB Example

disassemble main

```
(gdb) disas main
Dump of assembler code for function main:
   0x00000000004004e7 <+0>:     push   %rbp
   0x00000000004004e8 <+1>:     mov    %rsp,%rbp
   0x00000000004004eb <+4>:     sub    $0x10,%rsp
   0x00000000004004ef <+8>:     movl   $0x0,-0x4(%rbp)
   0x00000000004004f6 <+15>:    jmp    0x400508 <main+33>
   0x00000000004004f8 <+17>:    lea    0xa5(%rip),%rdi        # 0x4005a4
   0x00000000004004ff <+24>:    callq  0x4003f0 <puts@plt>
   0x0000000000400504 <+29>:    addl   $0x1,-0x4(%rbp)
   0x0000000000400508 <+33>:    cmpl   $0x9,-0x4(%rbp)
   0x000000000040050c <+37>:    jle    0x4004f8 <main+17>
   0x000000000040050e <+39>:    mov    $0x0,%eax
   0x0000000000400513 <+44>:    leaveq
   0x0000000000400514 <+45>:    retq
End of assembler dump.
```

# GDB Example

set breakpoint

run

get rip register

examine memory

execute next instruction

```
(gdb) break *0x00000000004004ff
Breakpoint 1 at 0x4004ff: file example.c, line 6.
(gdb) run
Starting program: /vagrant/a.out

Breakpoint 1, 0x00000000004004ff in main () at example.c:6
6                       printf("Hello World\n");
(gdb) info register rip
rip            0x4004ff 0x4004ff <main+24>
(gdb) info register rdi
rdi            0x4005a4 4195748
(gdb) x/16xb $rdi
0x4005a4:       0x48    0x65    0x6c    0x6c    0x6f    0x20    0x57    0x6f
0x4005ac:       0x72    0x6c    0x64    0x00    0x01    0x1b    0x03    0x3b
(gdb) x/xs $rdi
0x4005a4:       "Hello World"
(gdb) next
Hello World
5               for (i = 0;i < 10; i++) {
(gdb)
```

# GDB Example

continue

quit gdb

```
(gdb) continue
Continuing.

Breakpoint 1, 0x00000000004004ff in main () at example.c:6
6                       printf("Hello World\n");
(gdb) quit
A debugging session is active.

        Inferior 1 [process 7107] will be killed.

Quit anyway? (y or n) y
```

# More documents/tutorials to check!

- Linux commands
  - The Linux command line for beginners: https://ubuntu.com/tutorials/command-line-for-beginners#1-overview

- VIM
  - Interactive VIM tutorial: https://www.openvim.com/
  - VIM Cheat Sheet: https://vim.rtorr.com/

- GDB
  - GDB user manual: https://sourceware.org/gdb/current/onlinedocs/gdb.pdf