System Programming
(ELEC462)

Introduction

Dukyun Nam

HPC Lab @ KNU

Contents

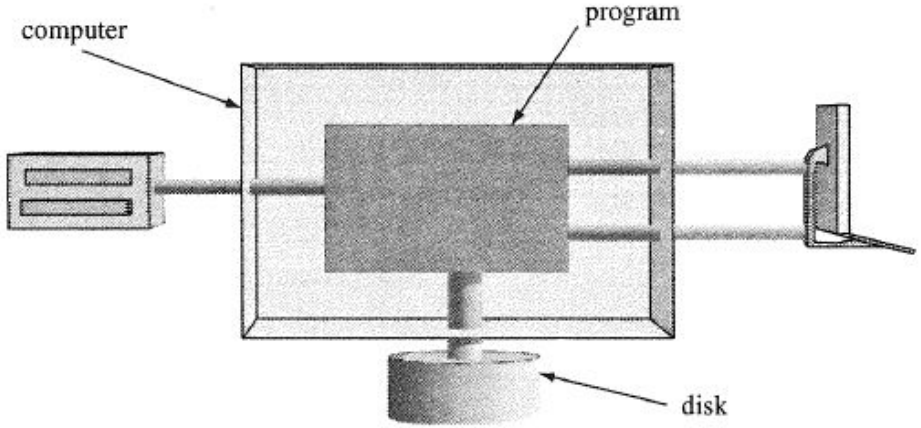
Introduction

- What is ***system programming***, or ***systems programming***?
  - The activity of programming computer system software
    - Computer *programming* is the process of performing a particular computation
    - *System software* is software designed to provide a platform for other software
- Application programming vs system programming
  - **Application programming** aims to produce software which provides services to the *user* directly
    - e.g., Word processor
  - **System programming** aims to produce software and software platform which provides service to *other softwares*

*** https://en.wikipedia.org/wiki/Systems_programming**

Introduction (cont.)

- The simple program model
  - Many programs are based on the model as shown below
    - A program: a piece of code that runs in a computer
      - Data go into the program; the program does some tasks with the data, and the output comes out of the program
      - A human may type at a keyboard and monitor; the program may read or write to a disk; the program may send data to a printer; many possibilities!

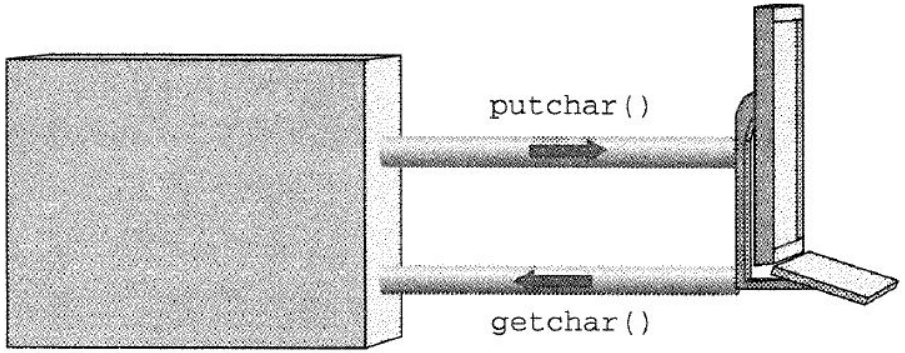


< An application program in a computer >

Introduction (cont.)

- The simple program model
  - The keyboard and screen are both connected to the program
    - Tip) GCC reports “errors” with the source file name and line number

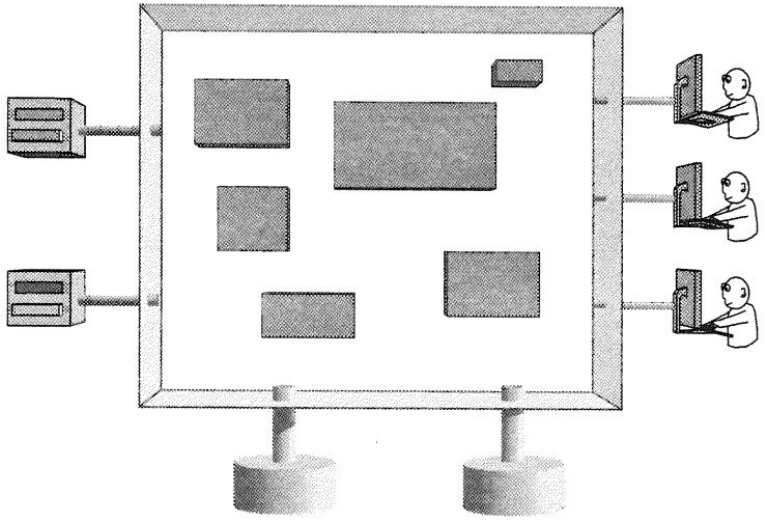
```
/* copy from stdin to stdout */
void main(){
  int c;
  while ( (c = getchar() ) != EOF)
          putchar(c)
}
```



< How application programs see user I/O >

Introduction (cont.)

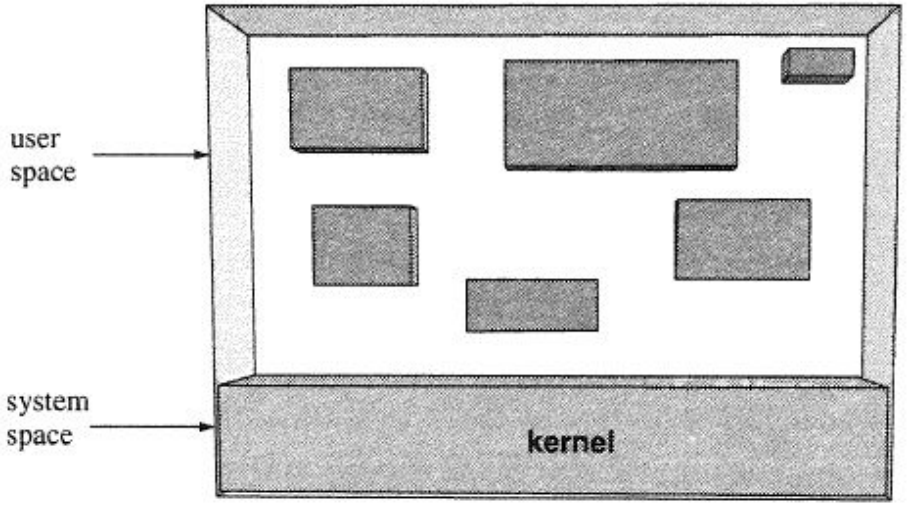
- Face reality
  - There are several keyboards and displays, there may be several disks, one or more printers, and there are certainly several programs running at once
  - Programs that get input from *the keyboard* and send output *the display* or to *the disk* work fine



< Reality: many users, programs, and devices >

Introduction (cont.)

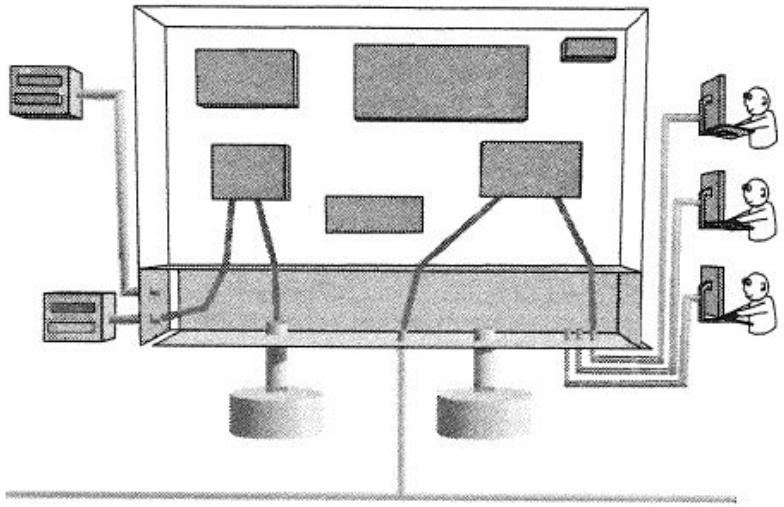
- The role of the operating system
  - To *manage and protect* all the resources and to connect various devices to various programs
  - Operating system: called a **kernel**
  - Memory: space to store programs and data
    - User space
      - Contains users programs
    - System space
      - The operating system is stored



< An operating system is a *program* >

Introduction (cont.)

- Providing services to programs
  - If a program wants to connect to or control any the devices, it needs to ask the kernel
  - Access to the external objects are *services* the kernel provides to user programs



< The kernel manages all connections >

System Resources

- Processors
  - A program: a set of (machine) instructions compiled from source code
  - A processor (CPU): the hardware executing instructions
    - The kernel assigns processors to programs
    - The kernel starts, pauses, resumes, and ends the execution of a program on a processor
- Input/output
  - All data that flow in and out of a program go through the kernel
    - e.g., Data coming from/going out to users at terminals or disks cannot travel without passing through the kernel

System Resources (cont.)

- Process management
  - Process: a program in execution
    - Consisting of i) memory, ii) open files, and iii) other system resources (e.g., socket, device driver, etc.) needed
  - The kernel creates new processes and schedules them to work cooperatively
- Memory
  - Computer's memory is a resource.
  - The kernel:
    - Keeps track of which processes are using which sections of memory, and
    - Protects the memory of one process from being damaged by another process.

System Resources (cont.)

- Devices
  - One can attach all sorts of devices to a computer
    - e.g., Mice, scanners, printers, keyboards, USB drives, iPhones, ..
  - The kernel provides access to devices and takes care of complexities that may derive from communications with the attached.
- Timers
  - Some programs depend on time.
    - e.g., Periodic tasks (by `cron`), sleep, elapsed time, …
  - The kernel makes timers available to processes.

System Resources (cont.)

- Interprocess communication (IPC)
  - In a real world, people need to communicate via phone, email, …
  - In a computer system running simultaneous programs, processes need such communication as well
  - The kernel provides a variety of IPC forms
- Networking
  - A network allows processes on different computers, even on different operating systems for data exchange
  - Network access is a kernel service

Our Goal

- Exploring the following questions:
  - What are the details of each type of service?
  - How does data get from a device to the program and back?
  - …
- We want to understand:
  - What the kernel is doing
  - How the kernel does it
  - How to write programs that use those services

Our Method

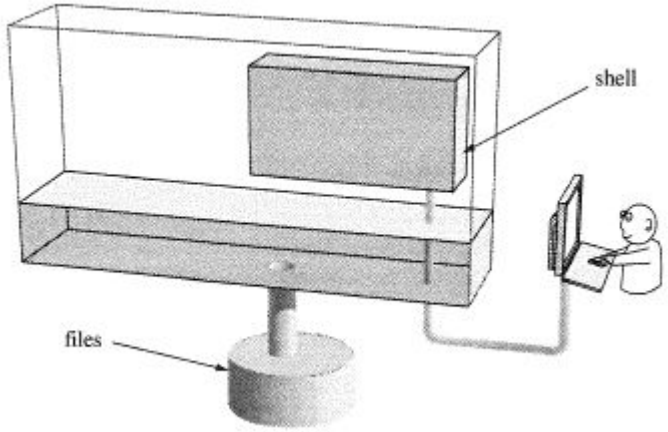
- Three simple steps to learn about system services by
  - 1. Looking at “real” programs
    - We will study standard Unix/Linux programs to see what they do and how they are used in practice
    - We will see what system services these programs use
  - 2. Looking at the system calls
    - We will learn about the system calls we use to invoke these services
  - 3. Writing our own version
    - Once we understand the program, what system services it uses, and how those services are used, we are ready to develop own system programs
    - Our own programs will be extensions of existing programs or ones that use similar principles

Our Method (cont.)

- Three questions that will be asked throughout our course
  - 1. What does that (kernel service) do?
  - 2. How does that work?
  - 3. Can I try to do it?

Unix from the User Perspective

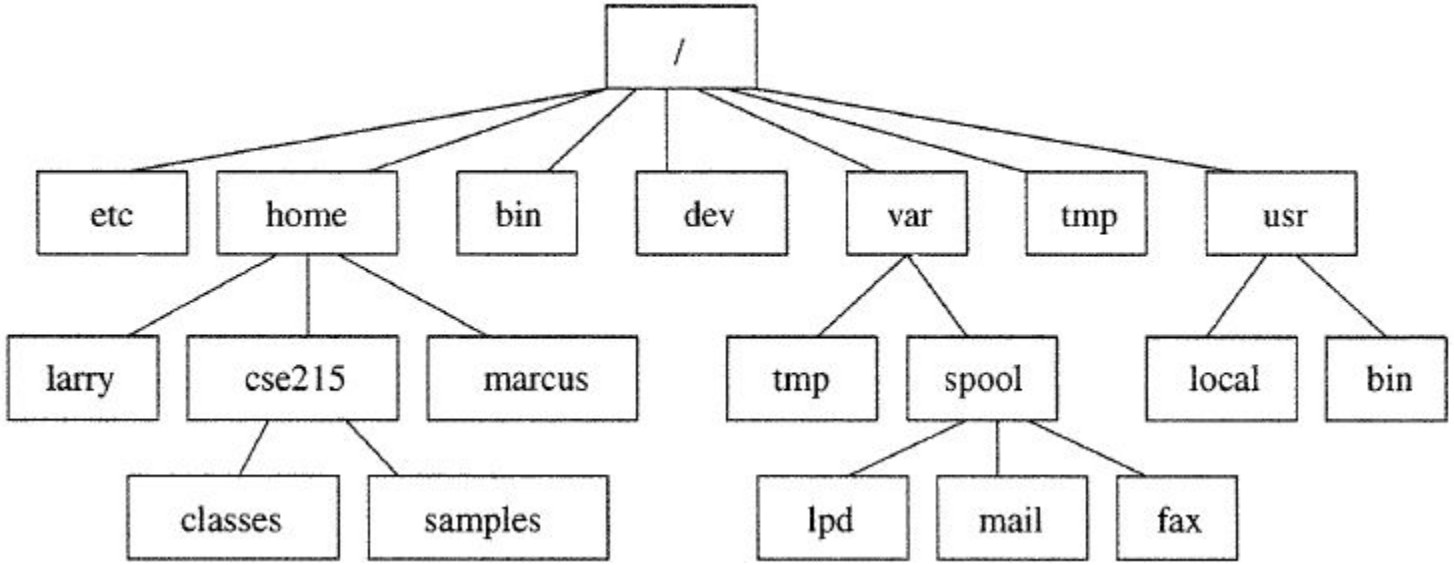
- Log in - Run programs - Log out
  - You login, run some programs, and then log out
  - The computer is the box at the left part of the picture
    - Inside the computer is memory that holds the kernel and user processes
    - The shell prints a prompt to tell the user it is ready to run a program



< The kernel manages all connections >

Unix from the User Perspective (cont.)

- Working with directories
  - A tree of directories



< Part of the directory tree >

Unix and Linux

- Unix is an operating system first developed by AT&T Bell Labs in the late 60s.
  - Primary contributors: Ken Thompson (B lang) and Dennis Ritchie (C Lang)
  - Written in C;
  - Commonly used in internet servers, workstations, etc
  - e.g., OS X, Solaris, BSD (Berkeley Software Distribution)
- Linux:
  - Developed by Linus Torvalds (responsible for kernel), combining GNU (GNU's Not Unix) tools from Richard Stallman
  - A "Unix-like" operating system: user-level interface very similar to Unix
  - Code base is different from original Unix code
  - Several distributions: e.g., Ubuntu, Fedora, SUSE, RedHat, …

*** https://opensource.com/article/18/5/differences-between-linux-and-unix**

Unix and Linux (cont.)

	Unix	Linux
Cost	Different cost structures according to vendors	Freely distributed, but licensed versions
Development & Distribution	Mostly AT&T	Developed by Open Source Community
Manufacturer	Solaris (Oracle), AIX (IBM), HP-UX (HP), OS X (Apple)	Developed by the community, but Linus Torvalds oversees
Usage	Internet Servers, Workstations, and Finance IT Infra with 24x7 availability	Cell phones, tablet PCs, video game consoles, mainframes/supercomputers.
Security	85 -120 viruses	60-100 viruses
	See how amazing these are compared with that of Windows!	
Inception	Developed by AT&T employees at Bell Labs and Dennis Ritchie in 1969. Written in “C” language. Designed to be a portable, multi-tasking and multi-user system in a time-sharing configuration.	Inspired by MINIX (a Unix-like system), later added many features like GUI, and then Linus Torvalds developed it in 1992.

*** https://www.diffen.com/difference/Linux_vs_Unix**

Summary

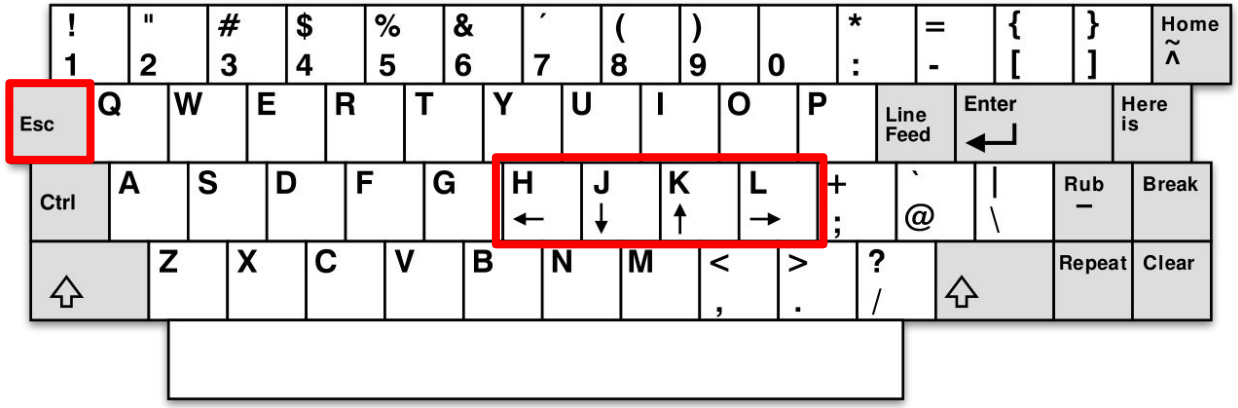
- Chapter 1 introduces the topic and approach of the book
- The purpose of the book is to explain how Unix works and to demonstrate how to write programs that use Unix system services
- The chapter explains that the Unix kernel is a program that manages multiple programs and system resources and connects programs to resources
- The method used to explain these ideas is to study and write common Unix programs

Appendix

`vi` Editor Tutorial

About `vi`

- `vi` (pronounced as distinct letters, /ˌviːˈaɪ/)
  - A screen-oriented text editor
  - The standard Unix editor
  - It is short for “visual”
  - There are lots of clones (`vim`, Elvis, nvi, etc.)



< ADM-3A terminal keyboard layout >

< ADM-3A terminal with keypad >

* https://en.wikipedia.org/wiki/Vi

Entering the `vi` Editor

Command	**Description**
vi *filename*	Edits an existing *filename* or creates a new one if *filename* does not exist.
vi –r *filename*	Edits *filename* using *.filename.swp* as the input . The file *.filename.swp* must be deleted, using: **rm** *.filename.swp*, after use.
vi –R *filename*	Edits *filename* in read-only mode does not allow changes to *filename*.
vi +/*pattern filename*	Edits *filename* at the line containing the *pattern*.

File Interaction in the `vi` Editor

Command	**Description**
:w	Write to the file that your are editing without exiting the editor. Good for intermediate saves.
:w *def*	Write to a new file *def*, but stay in the existing file.
:w! *def*	Write to an existing file *def*, but stay in the existing file.
:r *def*	add (read) file *def* into the current file
:f	Provides information on the file currently editing, including the current line number, if it has been modified etc. <`Ctrl+g`>

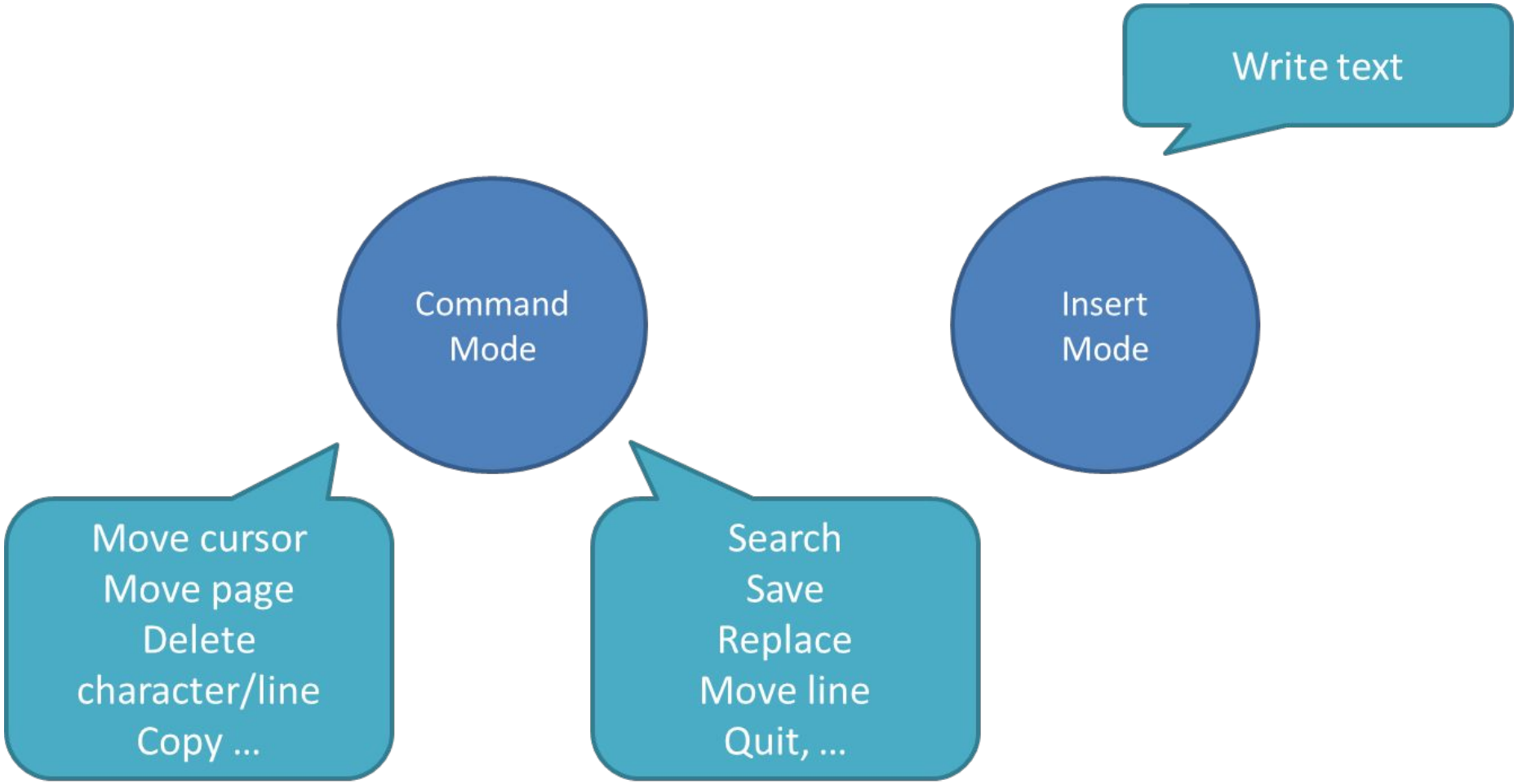
Exiting the `vi` Editor

Command	Description
:q	Exit (quit) without changes. Assumes no changes have been made.
:q!	Exit (quit) without changes. Assumes changes have been made.
ZZ or :x	Save the file only if changes have been made, then exit the editor
:wq	Save the file and exit the editor. This command will save the file even if no changes are made.

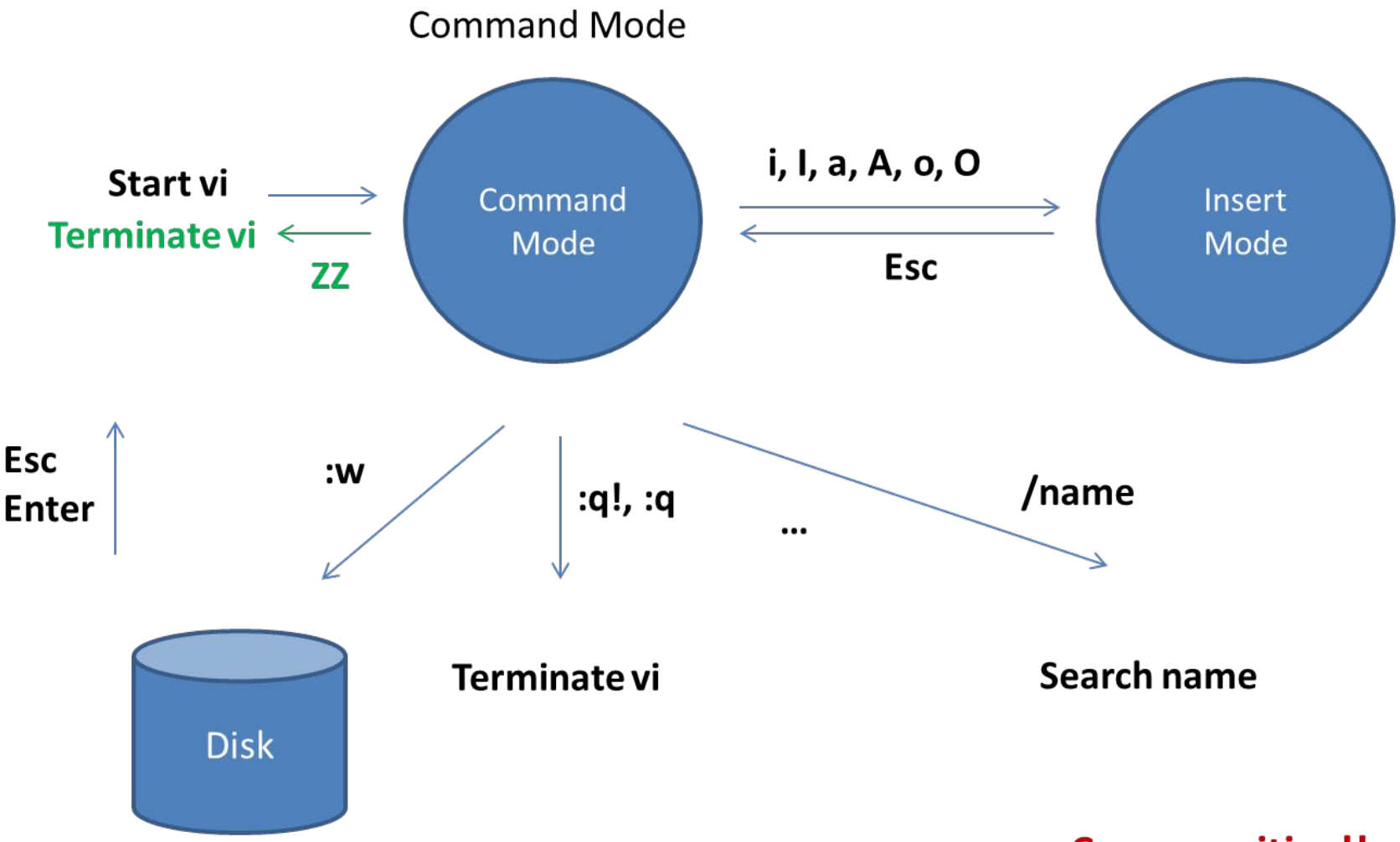
`vi` Modes

- Command mode:
  - The mode you are in when you start (default mode)
  - The mode in which commands are given to move around in the file, to make changes, and to leave the file
- Insert (or Text) mode
  - The mode in which text is created
  - There is more than one way to get into insert mode but only one way to leave: return to command mode by pressing <`Esc`>

`vi` Modes (cont.)



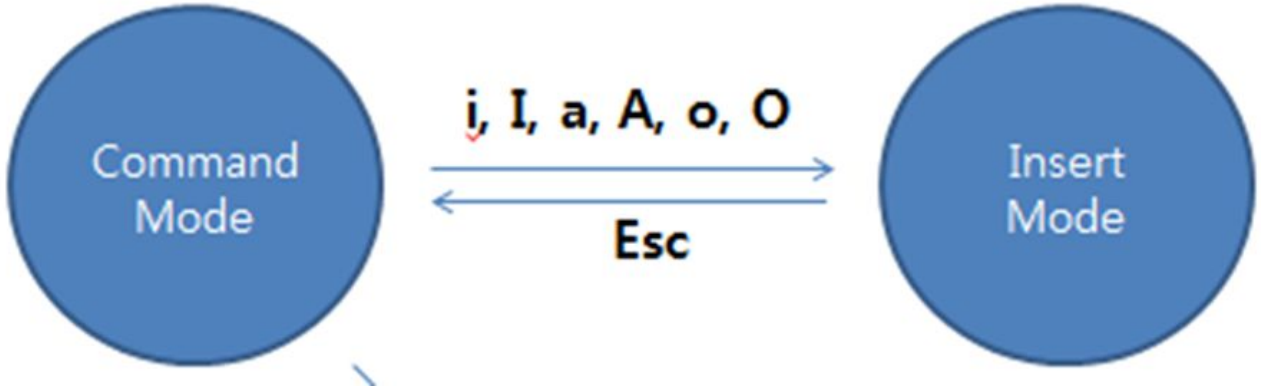
vi Modes (cont.)



Case-sensitive!!

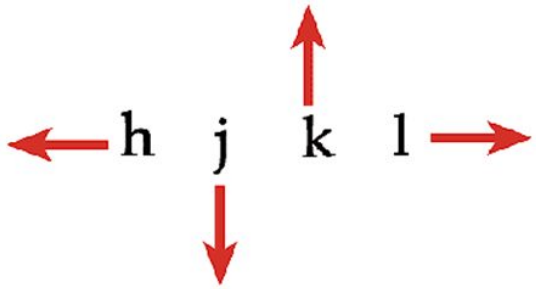
From Command Mode to Insert Mode

Key	Operation
`i`	Insert text before current character
`a`	Append text after current character
`I`	Begin text insertion at the beginning of a line
`A`	Append text at end of a line
`o`	Open a new line below current line
`O`	Open a new line above current line



Command Mode

- Basic cursor movement
  - `h, j, k, l`
  - Arrow key
    - Only support in `vim`



Key	Operation
`k`	Up one line
`j`	Down one line
`l`	Right one character (or use <`Spacebar`>)
`h`	Left one character
`w`	Right one word
`b`	Left one word
`W`	forward one word
`B`	back one word
`gg`	go to the first line
`G`	go to the last line

Command Mode (cont.)

- General command syntax
  - The general format for commands is

    `ncm`

  - Where:
    - **`n`** is an optional multiplier value
    - **`c`** is the command
    - **`m`** is an optional scale modifier
  - Examples:
    - `3dw` – delete 3 special character delimited words
    - `dW` – delete a single, space, delimited word

Command Mode (cont.)

- Delete commands

d	Delete this line
w	Delete special character delimited word
W	Delete whitespace delimited word
}	Delete to next paragraph
^	Delete to beginning of the line

- Delete character commands

n**x**	Delete current [and n-1] character[s]
n**X**	Delete previous n character[s]

Command Mode (cont.)

- Splitting and joining lines
  - If you wish to split a line between words, position yourself on a space between the words to be split, then use the replace (r) followed by the return key. This will replace the space with a carriage return forming a new line.
  - If you wish to join or combine two lines go to the upper of the two lines to be joined, then key in the uppercase **"J"** for join.
- Example:
  - We wish to split the line below between the word "should" and "be".

    "This line should be split."

    1. Position the cursor here
    2. Key in "r" followed by "enter"

  - To join the lines:

    "This line should
    be split."

    1. Position the cursor anywhere on this line
    2. Key in "J"

Command Mode (cont.)

- Search commands

/text	Search for text forward (wraps to beginning)
?*text*	Search for text backward
n	Search for same text again
N	Search for text reverse direction of initial search

Command Mode (cont.)

- Copy/Paste commands

[*n*]**Y** or	Copy (yank) n lines
[*n*]**y***m*	Copy (yank) a portion of the file determined by the measurement (e.g. 2yW will yank 2 space separated words.
P,p	Paste yanked or deleted data before (P), after (p) cursor position

  - Note: to cut and paste, use the **d**m commands to delete and the p or P to paste

- Visual block command

v	Start Visual mode per character
V	**Start Visual mode linewise**

Command Mode (cont.)

- Some “nice to know” features
  - To turn line numbers on and off use:
    - :set nu and :set nu!
  - To execute a single Unix command from inside the editor use:
    - :!cmd
  - To go to *linenum*
    - :*linenum*
  - To temporarily return to the shell use:
    - :sh (type “exit” to return to editor)
  - To repeat a colon command or to go back to earlier colon commands use:
    - : ↑ or↓(up arrow or down arrow)
  - To show the name of the current file use:
    - :f
- Resource: https://www.fprintf.net/vimCheatSheet.html