

Chapter 1.4: System Programing



GV: Nguyễn Thị Thanh Vân

Operating System Concepts – 10th Edition

Silberschatz, Galvin and Gagne ©2018



Outline

- Introduction
- System Programs
- System Programming



Operating System Concepts – 10th Edition

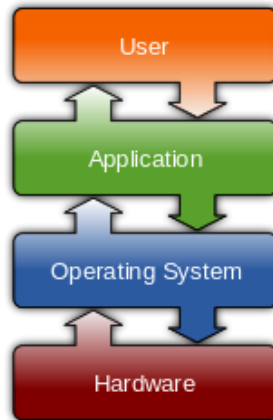
2a.2

Silberschatz, Galvin and Gagne ©2018



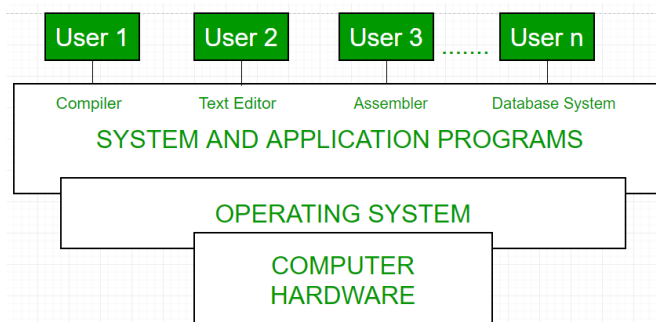
Introduction

- Operation System



Introduction

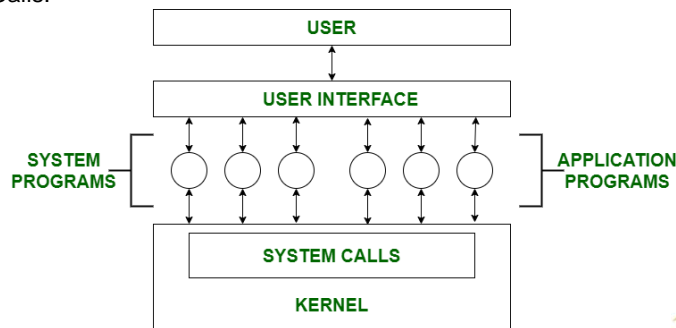
- System Programming** can be defined as the act of building Systems Software using System Programming Languages.
- According to Computer Hierarchy:
 - one which comes at last is Hardware
 - then it is Operating System, System Programs,
 - and finally Application Programs.





System Programs

- System Programs
 - Help Program Development and Execution can be done conveniently
 - Some of the System Programs are simply user interfaces, others are complex.
 - It traditionally lies between the user interface and system calls.
 - the user can only view up-to-the System Programs he can't see System Calls.



System Program types

- **File Management** – defined as the process of manipulating files in the computer system, its management includes the process of creating, modifying and deleting files
- **Status Information** – Information like date, time amount of available memory, or disk space is asked by some users.
- **File Modification** – For Files stored on disks or other storage devices, we used different types of editors. For searching contents of files or perform transformations of files we use special commands.
- **Programming-Language support** – Compilers, Assemblers, Debuggers, and interpreters are already provided to users. It provides all support to users.
- **Program Loading and Execution** – after Assembling and compilation, program must be loaded into memory for execution. Loaders, relocatable loaders, linkage editors, and Overlay loaders are provided by the system.
- **Communications** – Virtual connections among processes, users, and computer systems are provided by programs. Users can send messages to another user on their screen, User can use email, web, remote login, the transformation of files from one user to another.





System Programming

- Unix
- Windows
- C++



Linux: Command and program

1. Concept
2. Basic commands working with folder
3. System Programming Languages: Assembly, C, Python



9



Command

- ❖ Command: - a executable binary or
- a text file (written in the syntax of the shell.)
- ❖ Two command types:
 - Outside command: is a executable file that can be found out its location in the system. Shell created a child process to handle it.
 - Inside command (shell built-in): does not exist as a single file. It is available in the shell and ready to execute (as a keyword), no need to create a child process to handle it.
- ❖ **type:** type xxx
 - Outside command: return result: xxx is /bin/mkdir
 - Inside command: return result xxx is a shell builtin
- Syntax: **command [option] argument**



10



Path

- echo \$PATH: the paths that are set before
- ```
echo $PATH
/usr/kerberos/sbin:/usr/kerberos/bin...
```
- Absolute path: independence with the current directory of the user and start with /
  - The relative path: depends on the current directory of the user and does not start with /



11



## Time to execute the command

**time:** allow us know time to execute 1 command or 1 program in second

```
$ time find / -name tệp -print > result
```

```
55.6 real 1.5 user 18.3 sys
```

- ▶ real: total real time from press <ENTER> to Shell prompt come back
- ▶ user: major time to execute the command
- ▶ Sys: time that UNIX kernel use administrate that command

$\text{real} \geq \text{user} + \text{sys}$



12



## Basic commands working with folder

1. cd (change directory)
2. pwd (print working directory)
3. ls (list)
4. mkdir (make directory)
5. rmdir (remove directory)
6. basename và dirname



13



## Basic commands working with normal files

---

1. Create file: touch, cat
2. Copy, rename: cp
3. Move, rename: mv
4. list: ls
5. Remove: rm
6. Find file: find
7. Link: ln
8. Compare: cmp (compare), comm, diff, diff3
9. Edit: cat, head, tail, pg, more
10. Display with text/binary: od [tùy chọn]
11. Count: wc
12. size: sum
13. Compress and decompress: pack và gzip
14. Divide: Split
15. cut
16. sort
17. awk



14



## Data Flow Management

---

1. UNIX standard I/O
2. Redirecting data streams
3. Connecting pipes
4. Filter



15

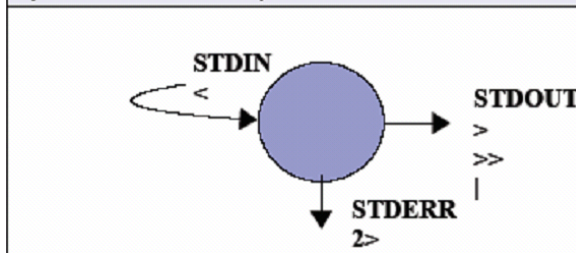


## UNIX standard I/O

3 standard I/O at terminal (/dev/tty):

| Chanel          | Num | Name          | Devices           |
|-----------------|-----|---------------|-------------------|
| Standard Input  | 0   | <b>stdin</b>  | Keyboard /dev/tty |
| Standard Output | 1   | <b>stdout</b> | Monitor /dev/tty  |
| Standard Error  | 2   | <b>stderr</b> | Monitor /dev/tty  |

*A process and it's 3 descriptors*



16



## Redirecting data flows

### ❖ Redirecting to new file

- Use ">"

- Ex : ls > file1

### ❖ Redirecting to old file

- Use ">>"

- Ex: cat file\_2 >> file\_1

### ❖ Redirecting from a file

- Use "<"

- Ex: tee < file1





17



## pipe

- A shell technique used to concatenate the data streams of several processes
- We can communicate with each other using pipes, for example :

**command\_1 | command\_2 | command\_3**

Where:

command\_1 need has output,

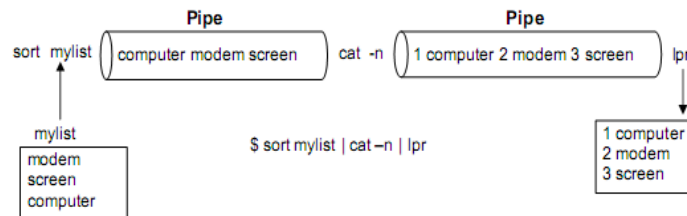
command\_2 is filter

command\_3 need has input.

PIPE: each pipe connects two data streams of two adjacent instructions through an intermediate pipe, represented by the BUFFER.

Operation: **command\_i ==> BUFFER ==> command\_j**

Ex:



18



## Filtering

1. tr (replace)

tr string\_1 string\_2

2. fgrep

3. grep

4. egrep

**grep [option] PATTERN FILE**

"string" PATTERN is regular expression





## Regular expression

- **^c**: rows begin with "c" (char, string)  
grep '^Begin' .
- **c\$**: rows end with "c"  
grep 'End\$'
- **\<c**: rows contain words begin with "c" (char, string)  
grep '\<Be'
- **c\>**: rows contain words end with "c" (char, string)



## Wildcards

- **.**: any ASCII character, except <RETURN>:  
grep '.\*' file => all rows in "file", including blank lines.
- **[ ]**: an ASCII character in square brackets, but needs to be enclosed between quotes like: [^xyzt]
- **-**: two ASCII characters inside square brackets, for example [b-y], represents a character in the range, but also needs to be enclosed in quotes like '[b-y]'.
- **\**: remove the special meaning of the character following it and return the original meaning.
- **^**: exception

Ex: an expression like [^xyzt] represents a character other than x, y, z, t





## Shell programming

- Variable: \$0, \$1, \$2....; \$#, \$\*, \$@
- Function

```
try_func() {
 echo <gtri> #return stdout
 return <gtri>
}

Call function
x=$(try_func)
```

- Structures
  - If
  - For
  - While, until
  - Case
- File System programming
  - Directory
  - File



## Structures

```
if condition
then
 statements
else
 statements
fi
```

```
if condition1
then
 statements
elif condition2; then
 statements
else
 statements
fi
```

```
for variable in values
do
 statements
done
```

```
while codition
do
 statements
done
```

```
until codition
do
 statements
done
```

```
case mau in
 mau1)
 statements;;
 mau2)
 statements;;
 *)
 statements ;;
esac
```





## Ex, run script

```
$ for file in *
> do
> if grep -l 'Hello' $file
> then
> more $file
> fi
> done
```

```
Cat > vidul.sh
#!/bin/sh
for file in *
do
 if grep -l 'Hello' $file
 then
 more $file
 fi
done
exit 0
```

Run script:  
chmod +x <filename>  
./<filename>



## Ex

Define Max\_2num(), then find Max of N intergers

```
max_2num()
{
if [$1 -gt $2]
then
 m=$1
else
 m=$2
fi
echo $m
return $m
}
```

```
#Main
max=$1
for i in $*
do
 max=$((max_2num $max $i))
done
echo "Max $# so: $max"
exit 0
```





## Lab No.4. Simple shell programs

- No.4.
  - number is even or odd
  - year is leap year or not
  - find the factorial of a number
  - swap the two integers
- Others:
  - Print the prime numbers in any sequence of numbers passed in from the command line (using a function check 1 number is prime or not)
  - Write a function to find the greatest common divisor of two numbers, then use the function you just wrote to find the UCLN of an array.
  - Input an array and sort the array ascending.
  - Check increment, decrement, symmetric arrays.
  - Input 1 array. Remove the odd elements in the array, Then print the remaining array



## Linux file system calls

- **Basic file functions:** Linux has many system calls to handle file, the table below shows some of common system calls.

| Functions    | Description                            | Returns                                                                                                                                    |
|--------------|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <b>open</b>  | Open a file                            | If success, return a file descriptor, otherwise, return -1                                                                                 |
| <b>close</b> | Close a file                           | Return 0 on success, otherwise, return -1                                                                                                  |
| <b>read</b>  | read data from a file                  | On success, return the number of bytes that been read, otherwise return -1                                                                 |
| <b>write</b> | write data to a file                   | On success, the number of bytes written is returned, otherwise, return -1                                                                  |
| <b>lseek</b> | seek to a specified position in a file | Upon successful completion, return the resulting offset location as measured in bytes from the beginning of the file, otherwise, return -1 |

- Ex, simulation of the Linux cp command
  - buffer is used to transfer data from the source file to destination file.





## Linux file system calls

- Directory
  - Note that **DIR** structure is an internal structure used by **readdir**, **closedir** to maintain information about the directory being read.
  - The **dirent** structure contains the inode number and the name. This information is collected into a file called dirent.h.

| Function                           | Description        | Returns                                                        |
|------------------------------------|--------------------|----------------------------------------------------------------|
| mkdir(const char * pathname, mode) | Create a directory | 0 if OK, -1 on error                                           |
| rmdir(const char * pathname)       | Delete a directory | 0 if OK, -1 on error                                           |
| opendir(const char * pathname)     | Open a directory   | Pointer of DIR if OK, NULL on error                            |
| readdir(DIR * dp)                  | Read a directory   | Pointer of dirent if OK, NULL at the end of directory or error |
| closedir(DIR * dp)                 | Close a directory  | 0 if OK, -1 on error                                           |

- Ex, simulates command ls to list all the file in the current directory.



## C - Functions in File Operations

- C File I/O: There are types and functions in the library **stdio.h** that are used for file I/O. Reading from or writing to a file in C requires 3 basic steps:
  - Open the file.
  - Do all the reading or writing.
  - Close the file
- For files you want to read or write, you need a file pointer: **FILE \*fp**;
  - FILE is some kind of structure that contains all the information necessary to control the file
- Open File -- **fopen(const char filename, const char mode)**
  - will initialize an object of the type FILE
- Close File -- **fclose(FILE \*fp)**
  - When done with a file, it must be closed





## C - Functions in File Operations

### Text I/O Functions

| Mode           | Description                          |
|----------------|--------------------------------------|
| <b>fEOF</b>    | detects end-of-file marker in a file |
| <b>fscanf</b>  | reads formatted input from a file    |
| <b>fprintf</b> | prints formatted output to a file    |
| <b>fgets</b>   | reads a string from a file           |
| <b>fputs</b>   | prints a string to a file            |
| <b>fgetc</b>   | reads a character from a file        |
| <b>fputc</b>   | prints a character to a file         |

- Ex: `fputc` `cp.c` use `fputc(int char, FILE *stream)` writes a character (an unsigned char) specified by the argument `char` to the specified stream and advances the position indicator for the stream.
  - Similarly, `fgetc(FILE *stream)` reads a character from the specified stream and advances the position indicator for the stream



## C - Functions in File Operations

### Binary I/O Functions

- When the files are binary, the previous functions will not work.
- For reading from and writing to a file on the disk respectively in case of binary files, use:

**fread(void \*buffer, size, number, FILE \*stream);**

**fwrite(void \*buffer, size, number, FILE \*stream);**

- ▶ "buffer": a pointer to buffer used for reading/writing the data,
- ▶ "void": a pointer that can be used for any type variable.
- ▶ "size": size of the objects to be read/written (ex, `sizeof(char)`)
- ▶ "number": number of objects to be read/written,
- ▶ "stream": the file pointer or stream which the data is to be read from/written to.
- If success, `fread/fwrite` return the number of items read or written. This number equals the number of bytes transferred only when size is 1. If fail, a lesser number of bytes is returned.





## Practice

---

- Ex No. 2,3 – file OS-LAB.pdf



## End of Chapter 1.4

---

