LECTURE-1

Notes

Operating Systems

- A System what works as an interface between the use and the computer hardware and controlls the execution.
- Functions
    - Resourse managment → Manages system resources.
    - Process Managment → Provides a platform for the application to run.
    - Memory Managment → Allocates, Deallocates and protects the memory.
    - File system Mangment → Organizes and secures the data.
    - Securty and Acces Control → controls permission and protects System integrity.

Microsoft Windows

- GUI easy to use
- Wide compatibility
- Rick ecosystem for personal and office productivity

UNIX and UNIX like OS

- More Stable, Secure and Scalable
- Usage
    - Mobile →IOS(Build on Darwin), Android(Build on Linux Kernel)
    - Desktop →MacOS(Build on Darwin) , Ubuntu, Fedora
    - Server → Linux, BSD, Solaris

Virtualization

- OS gives each process a virtual view of system resources.
    - Each process gets it own “virtual” CPU, memory, I/O.
- Goals
    - Efficiency → Optimal resources utilisation.
    - Fairness → Balanced allocation across.
    - Security → Isolation between process
- CPU Virtualization
    - System architecture
        - It connects to memory and graphics
        - I/O chip manages peripherals (keyboard, mouse, disks, network)
        - OS hides hardware complexity from applications.
        - Example: Opening a file involves disk access, buffering, CPU scheduling are all handled by the OS.

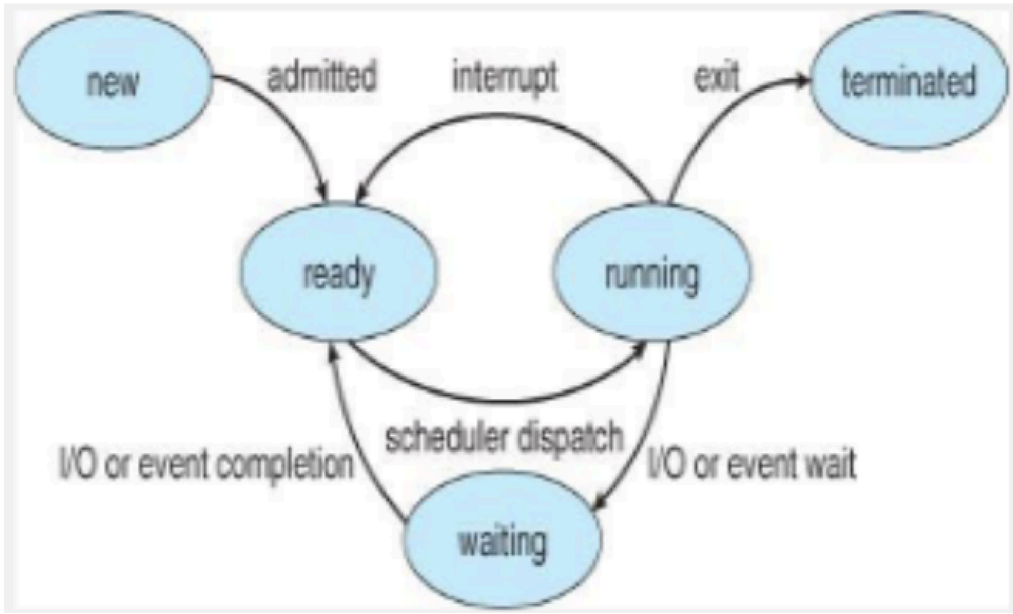
Process

- It is a active program in execution.
    - OS manages the processes: it creates,runs,suspends,and terminates them.
    - A single program can run multiple times, creating multiple processes(they need access to both hardware and software).
- Process Control Block (PCB)
    - Every Process has a profile called a PCB.
    - The OS creates a PCB when a new process.
    - It stores all the key information the OS need to manage that process such as its ID, current state, priority, and resources it’s using.
- Process Components

- Text Segment → Executes code(read-only)
- Stores global and static variables.
- Heap → Used for dynamic meomory allocation.
- Stack → Manages function call, Local variable and return addresses.

- Process Creation:
  - Stored on Disk → Program exists as an executable file.
  - Loaded into Memory → When executed the OS loads essential parts into RAM.
  - Memory Allocation → OS reserves space (heap+stack) for varible and execution state.
  - CPU Execution → OS assigns the CPU to start running instructions.
- Beyond Process Creation:
  - Concurrency
    - Many processes run at once.
    - OS suspends/resumes them
  - Security:
    - Stops crashes spreading.
    - Prevents data leaks between processes
- Processes Lifecycle:
  - Running:
    - The process is currently using the CPU to execute instructions.
    - Only one process runs at a time on a single-core system.
  - Ready:
    - The process is waiting in a queue for CPU time.
    - OS decides-when to move it to running state.
  - Waiting(Blocked):

- The process is waiting for some I/O operation(such as reading from disk, or user input).



- OS SWITCHES BETWEEN THESE STATES TO MANAGE MULTI-TASKING.

Speeds and Latency

- Latency → Time between request and response.
    - SSD Latency → 100 microseconds
    - Network Ping → 10 milliseconds
    - Human Reaction time → 250 milliseconds

Blocking and Scheduling

- Blocked:
    - Waiting for I/O (In CPU terms I/O takes eternity)
- Unblocked:
    - I/O completes, return to Ready States.
- De-Scheduled:
    - CPU taken by higher-priority process or OS itself.

Tutorial

Kernal

- Processes → scheduling and execution
- Memory → allocation and protection
- Files → access and organization
- Security → permissions and access control

Shell

- It is a Program that lets users talk to the OS.
- It converts human-related commands into instructions the kernal understands.
- Starts automatically when you log in or open a terminal.
- Types → Command line (e.g. Bash, Zsh) and Graphical Shell (Windows Explorer)

GUI vs CLI

- Graphical User Inderface(GUI)
  - Easy to learn — ideal for beginners
  - Point-and-click interaction
  - First experience for most users
- Command Line Interface (CLI)
  - Offers more control and flexibility
  - Faster for repetitive tasks
  - Ideal for automation and advanced work (e.g.,scripting, servers)

Terminal

- Program that lets you type Commands and see text based outputs/results,
- Used to write Scripts and automate tasks

Essential Shell Commands

- Navigation:
  - ls → list files
  - cd dirname → change directory
  - pwd → show current directory
- File Management:
  - mkdir foldername → create folder
  - cp source target → copy
  - mv oldname newname → move/rename
  - rm file.txt / rm -r folder → delete
- Content and Search:
  - cat file.txt → view file
  - ls > file.txt → redirect output
  - grep "pattern" filename → search
  - wc filename → count words/lines

Shell Script

- A text file containing a series of commands for the shell to execute.
  - The shell itself is a commandline interpreter (CLI), while a shellscript is a saved list of instructions (usually with .shextension) Includes:
    - Built in features (variables, loops)
    - System commands (ls, cp, cat, mv, grep, find)
    - Any installed programs or tools

```
#!/bin/bash
echo "Hello, welcome to shell scripting Tutorial!"
```

- Why Shell Script?

- Automation→ Saves time by automating repetitive tasks (e.g. file cleanup, backups, etc)
- System Administration → Manage users, file and system resources.
- Batch Processing → Process Multiple Files or jobs in one go.
- Rapid Prototyping → Quickly test ideas and workflows. (useful for scripting small tools or experiments)

Writing Shell Script

- Create a Script File → Add Script Content →Save and Exit →Make the Script Executable → Run the script

Concept	Description
Shebang (#!)	Specifies the shell to interpret the script (e.g., #!/bin/bash)
Comments (#)	Used for documentation and clarification within the script
Commands	Executed line-by-line (e.g., echo, ls, pwd)
Exit Codes	Indicate success or failure: 0 = success, 1 = error (Best practice)

- Variables

```
Basics:

name=Alice #correct
name= Alice #wrong (no space allowed)

Accesing Values:

echo $name #Alice
echo ${name} #Alice

Arithmetic:
```

```
x=5
y=3
echo $((x+y)) #8

Enviroment Variables:
   echo $SHELL #Path to your shell
   echo $USER #Current user
   echo $HOME #Home directory
   echo $PATH #Where shell looks for commands

Predefined Shell Variables
   -Command Arguments:
      $0 – script name
      $1, $2... – arguments
      $@ – all arguments as a list
      $# – number of arguments
   -Status:
      $? – exit status of last command
   -Example: ./myscript.sh hello world
      $0 – myscript.sh
      $1 – hello
      $2 – world
      $# – 2
      $@ – hello world
```

Simple Practice Scripts

```
# Check if ODD or EVEN
echo "Enter a number: "
read number
if [$ ((num % 2))=0];
then
   echo "$number is even"
```

```
else
   echo "$number is odd"
fi

#Loop Through Numbers
for i in {1..5}
do
   echo "Number $i"
done

# Display Current Date and Time:
echo "Current date and time"
date
```

Lab

Task 1: Files (and Shell Usage)

```
# Step 1: Go to your home directory
cd ~

# Step 2: Create CO2101 directory
mkdir CO2101

# Step 3: Create lab1 subdirectory inside CO2101
mkdir CO2101/lab1

# Step 4: Create a recursive list of all files and directories in your home direct
ory
ls -R ~ > CO2101/lab1/myhome1.txt

# Step 5: Make a complete copy of CO2101 directory called CO2101-copy
cp -r CO2101 CO2101-copy

# Step 6: Examine the contents of myhome1.txt using various commands
```

```
cat CO2101/lab1/myhome1.txt       # prints all contents
head CO2101/lab1/myhome1.txt      # shows first 10 lines
tail CO2101/lab1/myhome1.txt      # shows last 10 lines
less CO2101/lab1/myhome1.txt      # scrollable view (press 'q' to quit)
view CO2101/lab1/myhome1.txt      # open in vi-style viewer (press ':q' to quit)
```

Task 2: Count Your Files

```
# Count total files + directories (including hidden ones)
find ~ -xdev | wc -l

# Count only plain files
find ~ -type f -xdev | wc -l

# Count only directories
find ~ -type d -xdev | wc -l
```

Explanation:

`find ~ -xdev` prevents crossing into other mounted file systems.

`wc -l` counts the lines = number of items found.

- `type f` → files only
- `type d` → directories only

Task 3: Memory and Resource Usage by a Process

```
# Step 1: Go to your lab1 directory to store timing results
cd ~/CO2101/lab1

# Step 2: Run test 1 — search for 'hello' in documentation
/usr/bin/time -o timing1.txt -v rgrep hello /usr/share/doc/a*

# Step 3: Run test 2 — generate RSA keys (change NBITS if needed)
/usr/bin/time -o timing2.txt -v ssh-keygen -t rsa -b 4096
```

```
# Step 4: Run test 3 — any custom long-running command (example: find all fi
les)
/usr/bin/time -o timing3.txt -v find /usr -name "*.conf" 2>/dev/null
```

To Examine Timing Results

```
# View each timing result
less timing1.txt
less timing2.txt
less timing3.txt
```