

System Administration (2)

IT401

Prepared by
Dr. Ahmed El-Douh

Lecture 5



System Administration 2

Lecture 5

IT401

File Permission Control and Process Management

Prepared by
Dr. Ahmed El-Douh

Objectives

Upon completion of this lecture, you will be able to:

- List file system permissions on files and directories, and interpret the effect of those permissions on access by users and groups.
- Change the permissions and ownership of files with command-line tools.
- Control the default permissions of user-created files, explain the effect of special permissions, and use special and default permissions to set the group owner of files that are created in a directory.

Outlines

- Control Access to Files with Linux system permissions
- Monitoring and Managing Linux Processes



Control Access to Files with Linux system permissions



Linux file system permissions

- Access to files is controlled by **file permissions**.
- The system is designed to be **simple, flexible, and easy to apply**.
- **Permissions are applied across three distinct categories** of users:
 1. The owning **User** (The creator/owner).
 2. The owning **Group** (A single primary group).
 3. **Other** Users (Everyone else on the system).
- **Permissions Hierarchy** Rule: Specific permissions override general ones. The User permission overrides the Group permission, which overrides the Other permission.

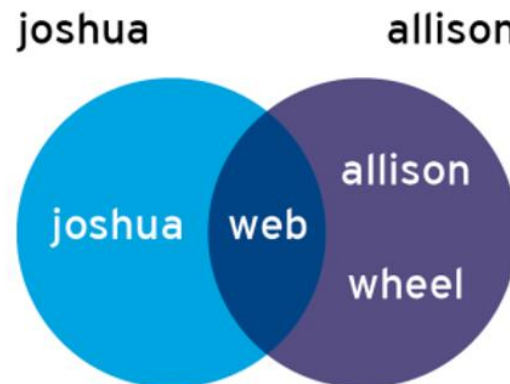
GROUP PERMISSIONS: Enabling Collaboration

Scenario: Users **joshua** and **allison** need to collaborate on files.

Shared Group: The common group between them is **web** (The intersection point).

Implementation: Files are **associated with the group web**.

Goal: By setting appropriate **Group Permissions**, both users gain the necessary access to the files.



Effects of Permissions on Files and Directories

Permission	Effect on files	Effect on directories
r (read)	File contents can be read.	Contents of the directory (the file names) can be listed.
w (write)	File contents can be changed.	Any file in the directory can be created or deleted.
x (execute)	Files can be executed as commands.	The directory can become the current working directory. You can run the <code>cd</code> command to it, but it also requires read permission to list files there.

View File and Directory Permissions and Ownership

Default File Permission Assignment

File Owner and Group:

❑ **Owner (User):** The user who created the file

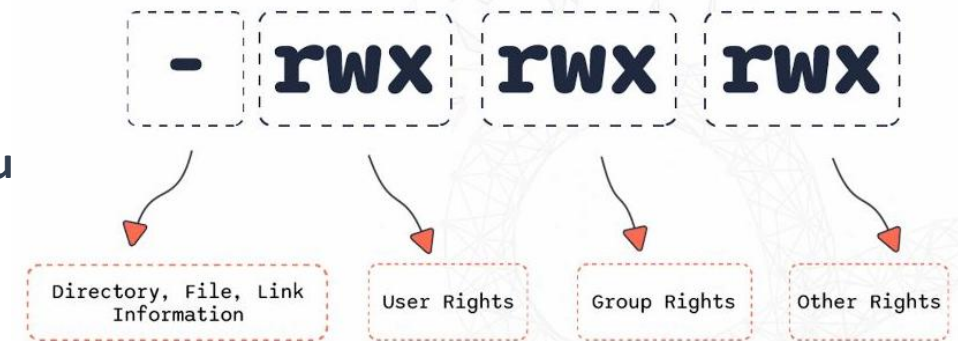
rw-: Owner (student) permissions - read and write, no execu

❑ **Group:** The user's primary group (from /etc/passwd)

rw-: Group (student) permissions - read and write, no execute

❑ **Others:** Everyone else on the system

r--: Others permissions - read only



```
[student@desktop ~]$ touch test1
[student@desktop ~]$ ls -l test1
-rw-rw-r--. 1 student_student 0 Oct  1 08:51 test1
```

View File and Directory Permissions and Ownership

- To prevent the descent into the directory and see the expanded listing of the directory itself, add the **-d** option to ls:

```
[user@host ~]$ ls -ld /home  
drwxr-xr-x. 5 root root 4096 Feb 31 22:00 /home
```

The first character of the long listing is the file type, and is interpreted as follows:

- **-** is a regular file.
- **d** is a directory.
- **l** is a symbolic link.
- **c** is a character device file.
- **b** is a block device file.
- **p** is a named pipe file.
- **s** is a local socket file.

View File and Directory Permissions and Ownership

EXAMPLES: LINUX USER, GROUP, OTHER CONCEPTS

Users and their groups:

lucy	lucy,ricardo
ricky	ricky,ricardo
ethel	ethel,mertz
fred	fred,mertz

File attributes (permissions, user & group ownership, name):

drwxrwxr-x	ricky	ricardo	dir	(which contains the following files)
-rw-rw-r--	lucy	lucy	lfile1	
-rw-r--rw-	lucy	ricardo	lfile2	
-rw-rw-r--	ricky	ricardo	rfile1	
-rw-r-----	ricky	ricardo	rfile2	

File Permission Control

❑ **The main purpose of the Purpose** : Secure files/directories by modifying access rights for users, groups, and others

❑ **Permission Basics:**

▪ Three Entities:

- **u** → Owner
- **g** → Group
- **o** → Others

a → all (user + group + others)

▪ Three Permissions:

- **r** = Read (4)
- **w** = Write (2)
- **x** = Execute (1)

[**rw**x]



الصلاحية	الرمز	القيمة الرقمية (Octal Value)
Read (قراءة)	r	4
Write (كتابة)	w	2
Execute (تنفيذ)	x	1

Change File and Directory Permissions

❑ What is **chmod**?

- **chmod** stands for **Change Mode** — it's the command used in Linux to set file and directory permissions.

❑ Symbolic Mode

- + Adds the given permission(s) without changing existing ones.
- Removes the given permission(s) but leaves others unchanged.
- = Replaces all permissions for the category with exactly what you specify.
- **chmod u+x file.txt** # Add execute for user
- **chmod g-w file.txt** # Remove write for group
- **chmod o=r file.txt** # Set read only for others
- **chmod a+rw file.txt** # Add read/write for everyone

Change File and Directory Permissions

chmod

☐ Numeric Mode

Uses numbers:

r = 4, **w = 2**, **x = 1**, **- = 0** Sum them to get permission values.

chmod 754 lab.txt

- 7 → user: 4+2+1 = **rwX**
- 5 → group: 4+0+1 = **r-X**
- 4 → others: 4+0+0 = **r--**

```
[student@desktop ~]$ touch lab.txt
[student@desktop ~]$ chmod 754 lab.txt
```

```
[student@desktop ~]$ ls -ld lab.txt
-rwxr-xr-- 1 student student 0 Oct  1 14:17 lab.txt
```

File Permission Control

chmod

Examples

- Remove read and write permission for group and other on **file1**:

```
[student@desktopX ~]$  
chmod go-rw file1
```

- Add execute permission for everyone on **file2**:

```
[student@desktopX ~]$  
chmod a+x file2
```

- Set read, write, and execute permission for user, read, and execute for group, and no permission for other on **sampledir**:

```
[student@desktopX ~]$  
chmod 750 sampledir
```

FILE/DIRECTORY OWNERSHIP MANAGEMENT (**chown**)

- Change the file's **User** or **Group** owner to manage access for collaboration.
- Files are initially owned by the **creator user** and their **primary group**.
- **Key Command:** Use **chown** (change owner).

○ **Practical Examples:**

- **Change User Owner:** `chown student foofile`
- **Change Recursively:** `chown -R student foodir` (Applies to all sub-contents).
- **Change Group Only:** `chown :admins foodir` (Use a colon : before the group name).

```
[root@desktopX ~]#  
chown student foofile
```

```
[root@desktopX ~]#  
chown -R student foodir
```

```
[root@desktopX ~]#  
chown :admins foodir
```

FILE/DIRECTORY OWNERSHIP MANAGEMENT (**chown**)

- **Change User & Group Together:**

- Use the syntax **owner:group**.
- *Example:* `chown visitor:guests foodir`

```
[root@desktopX ~]#  
chown visitor:guests foodir
```

- **Privileges for Ownership Change:**

- **File Owner:** Only the **root** user can change the file's primary owner.
- **Group Owner:** Can be changed by **root** OR the file's current owner.

- **Group Granting Rules:**

- **Root** can set group ownership to *any* group on the system.
- **Non-root** users can only set group ownership to groups they are currently a member of.

Manage Default Permissions and File: Access Special Permissions

▪ **setuid / setgid** on Executable Files:

- Command runs with the permissions of the file's owner/group, not the user executing it (e.g., the **passwd** command).
- Indicated by **s** (or **S**) in the user/group execute position.

```
[student@desktop ~]$ ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 32648 Aug 10 2021 /usr/bin/passwd
```

▪ **sticky bit** on Directories:

- Restricts file deletion: Only the file's owner (or **root**) can delete files inside the directory (e.g., / **tmp**).
- Indicated by **t** (or **T**) in the **other** execute position.

```
[student@desktop ~]$ ls -ld /tmp
drwxrwxrwt. 20 root root 4096 Oct 2 14:42 /tmp
```

▪ **setgid** on Directories:

- New files created inside the directory **inherit the directory's group ownership**, which is crucial for **group collaboration**.
- Indicated by **s** (or **S**) in the **group** execute position.

Effects of **Special Permissions** on Files and Directories

Permission	Effect on files	Effect on directories
u+s (suid)	File executes as the user that owns the file, not as the user that ran the file.	No effect.
g+s (sgid)	File executes as the group that owns the file.	Files that are created in the directory have a group owner to match the group owner of the directory.
o+t (sticky)	No effect.	Users with write access to the directory can remove only files that they own; they cannot remove or force saves to files that other users own.

Effects of **Special Permissions** on Files and Directories

- Symbolically: setuid = **u+s**; setgid = **g+s**; sticky = **o+t**
- Numerically (fourth preceding digit): setuid = 4; setgid = 2; sticky = 1

Examples

- Add the setgid bit on **directory**:

```
[root@desktopX ~]#  
chmod g+s directory
```

- Set the setgid bit, and read/write/execute for user and group on **directory**:

```
[root@desktopX ~]#  
chmod 2770 directory
```



Monitoring and Managing Linux Processes



WHAT IS A **PROCESS**?

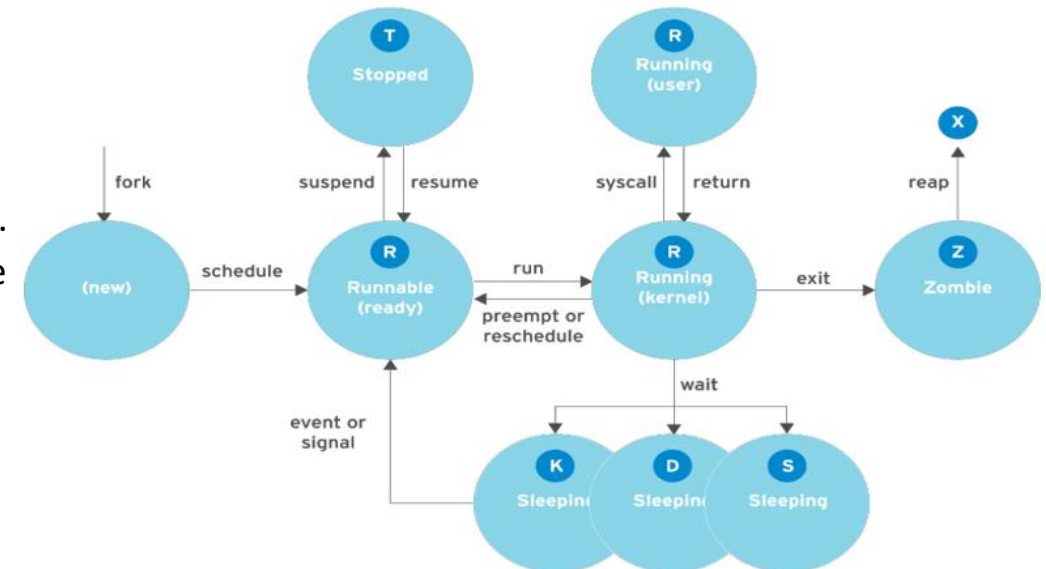
- **Definition:** A **process** is a running instance of an executable program.
- **Key Components (A Process Includes):**
 - An **address space** (allocated memory).
 - **Security properties** (ownership and privileges).
 - **Execution threads** of program code.
 - The **process state** (Running, Sleeping, Stopped, etc.).
- **Process Environment:** Includes **variables**, a **scheduling context**, and allocated **system resources** (like file descriptors and network ports).
- **Process Creation (The Hierarchy):**
 - A **parent** process uses **fork** to create a **child** process structure.
 - Each process is assigned a unique **PID** and has a **PPID** (Parent's PID).
 - All **processes** descend from **systemd (PID:1)** (the first system process).

PROCESS STATES

In a multitasking OS, a process is assigned a **state** that reflects its immediate resource needs (CPU time, I/O, etc.).

Key States and Transitions:

- 1.**(new)**: Initial state after **fork**.
- 2.**Runnable (ready)**: Process is ready to run and waiting for the **CPU scheduler**.
- 3.**Running (user/kernel)**: The process is currently executing instructions on the CPU.
- 4.**Sleeping (S, D, K)**: The process is temporarily inactive, waiting for a resource or an **event**.
- 5.**Stopped (T)**: The process is temporarily suspended (**suspend**) but can be **resumed**.
- 6.**Zombie (Z)**: The process has finished execution (**exit**) but its entry remains in the process table until the **parent process** cleans it up (**reap**).



Linux Process States

NAME	FLAG	KERNEL-DEFINED STATE NAME AND DESCRIPTION
Running	R	TASK_RUNNING: The process is either executing on a CPU or waiting to run. Process can be executing user routines or kernel routines (system calls), or be queued and ready when in the <i>Running</i> (or <i>Runnable</i>) state.
Sleeping	S	TASK_INTERRUPTIBLE: The process is waiting for some condition: a hardware request, system resource access, or signal. When an event or signal satisfies the condition, the process returns to <i>Running</i> .
	D	TASK_UNINTERRUPTIBLE: This process is also <i>Sleeping</i> , but unlike S state, will not respond to delivered signals. Used only under specific conditions in which process interruption may cause an unpredictable device state.
	K	TASK_KILLABLE: Identical to the uninterruptible D state, but modified to allow the waiting task to respond to a signal to be killed (exited completely). Utilities frequently display <i>Killable</i> processes as D state.
Stopped	T	TASK_STOPPED: The process has been <i>Stopped</i> (suspended), usually by being signaled by a user or another process. The process can be continued (resumed) by another signal to return to <i>Running</i> .
	T	TASK_TRACED: A process that is being debugged is also temporarily <i>Stopped</i> and shares the same T state flag.
Zombie	Z	EXIT_ZOMBIE: A child process signals its parent as it exits. All resources except for the process identity (PID) are released.
	X	EXIT_DEAD: When the parent cleans up (<i>reaps</i>) the remaining child process structure, the process is now released completely. This state will never be observed in process-listing utilities.

LISTING PROCESSES

LISTING PROCESSES: The **ps** Command

- The **ps** command is the primary tool for listing current processes on the system.
- **ps** provides detailed information, including:
 - **UID** (User Identification) to determine privileges.
 - **PID** (Process Identification) for unique tracking.
 - **CPU and real time** already expended.
 - **Memory allocation**.
 - The **controlling terminal** (location of **STDOUT**).
 - The current **process state**.

LISTING PROCESSES

LISTING PROCESSES: **ps** Options

▪ **ps aux** (Common Display):

- Displays **all processes** on the system.
- Includes processes **without a controlling terminal**.

▪ **ps lax** (Long, Detailed Display):

- Provides **more technical detail**.
- May display **faster** because it avoids looking up the username.

• **UNIX Equivalent:** The options **-ef** provide similar functionality for displaying all processes.

```
[student@serverX ~]$ ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.1	0.1	51648	7504	?	Ss	17:45	0:03	/usr/lib/systemd/syst

```
[student@serverX ~]$ ps lax
```

F	UID	PID	PPID	PRI	NI	VSZ	RSS	WCHAN	STAT	TTY	TIME	COMMAND
4	0	1	0	20	0	51648	7504	ep_pol	Ss	?	0:03	/usr/lib/systemd/
1	0	2	0	20	0	0	0	kthrea	S	?	0:00	[kthreadd]
1	0	3	2	20	0	0	0	smpboo	S	?	0:00	[ksoftirqd/0]
1	0	5	2	0	-20	0	0	worker	S<	?	0:00	
[kworker/0:0H]												
1	0	7	2	-100	-	0	0	smpboo	S	?	0:00	[migration/0]
-- output truncated --												

Controlling Jobs



Control Access to Files: **JOBS AND SESSIONS**

- **Job Control:** A feature of the shell that enables a single shell instance to manage and run multiple commands.
- A **job** is created for every command or pipeline entered at the shell prompt.
- **Process Group:** All processes associated with a single job/pipeline are members of the same process group.

Control Access to Files: **RUNNING JOBS IN THE BACKGROUND**

- Start any command or pipeline in the **background** by adding an **ampersand (&)** at the end of the command line.
- **Shell Output:** The **bash shell** immediately provides two identifiers:
 - The **job number** (unique to the current session).
 - The **PID** (**Process ID**) of the new child process.
- **Result:** The shell prompt immediately returns, allowing the user to execute new commands while the job runs non-interactively in the **background**.

```
[student@serverX ~]$ sleep 10000 &  
[1] 5947
```

Control Access to Files: RUNNING JOBS IN THE BACKGROUND

The **bash** shell tracks jobs, per session, in a table displayed with the **jobs** command.

```
[student@serverX ~]$ jobs
[1]+  Running                  sleep 10000 &
[student@serverX ~]$
```

A background job can be brought to the foreground by using the **fg** command with its job ID (*%job number*).

```
[student@serverX ~]$ fg %1
sleep 10000
—
```

In the preceding example, the **sleep** command is now running in the foreground on the controlling terminal. The shell itself is again asleep, waiting for this child process to exit.

To send a foreground process to the background, first press the keyboard-generated *suspend* request (**Ctrl+z**) on the terminal.

```
sleep 10000
^Z
[1]+  Stopped                  sleep 10000
[student@serverX ~]$
```

```
[student@desktop ~]$ fg %1
sleep 10000
^Z
[1]+  Stopped                  sleep 10000
[student@desktop ~]$ bg
[1]+ sleep 10000 &
```

Control Access to Files: **RUNNING JOBS IN THE BACKGROUND**

The **ps j** command is used to display detailed information about running jobs and processes.

- **Key Identifiers:**

- **PGID:** The Process Group ID, which is the PID of the **process group leader** (the first process in the job's pipeline).
- **SID:** The Session ID, which is the PID of the **session leader** (the controlling interactive shell).

- **Process State (STAT):** The **STAT** field indicates the process state. For the suspended example, the state is **T** (for Suspended/Stopped).

```
[student@serverX ~]$ ps j
PPID  PID  PGID  SID TTY      TPGID STAT   UID    TIME COMMAND
2764  2768  2768  2768 pts/0    6377 Ss     1000    0:00 /bin/bash
2768  5947  5947  2768 pts/0    6377 T      1000    0:00 sleep 10000
2768  6377  6377  2768 pts/0    6377 R+    1000    0:00 ps j
[student@serverX ~]$
```

KILLING PROCESSES

- **Signal Definition:** A **signal** is a software interrupt delivered to an executing **process**.
- **Signal Purpose:** Signals report events or notifications to the program.
- **Sources of Signals:**
 - An **error** within the program.
 - An **external event** (e.g., I/O request, timer expiration).
 - An **explicit request** (e.g., a signal-sending command or keyboard sequence).
- System administrators use **fundamental signals** (like **HUP** / **SIGHUP**) for routine **process management**.

SIGNAL NUMBER	SHORT NAME	DEFINITION	PURPOSE
1	HUP	Hangup	Used to report termination of the controlling process of a terminal. Also used to request process reinitialization (configuration reload) without termination.
2	INT	Keyboard interrupt	Causes program termination. Can be blocked or handled. Sent by pressing INTR key combination (Ctrl+c).
3	QUIT	Keyboard quit	Similar to SIGINT , but also produces a process dump at termination. Sent by pressing QUIT key combination (Ctrl+\).
9	KILL	Kill, unblockable	Causes abrupt program termination. Cannot be blocked, ignored, or handled; always fatal.
15 <i>default</i>	TERM	Terminate	Causes program termination. Unlike SIGKILL , can be blocked, ignored, or handled. The polite way to ask a program to terminate; allows self-cleanup.
18	CONT	Continue	Sent to a process to resume if stopped. Cannot be blocked. Even if handled, always resumes the process.
19	STOP	Stop, unblockable	Suspends the process. Cannot be blocked or handled.

SENDING SIGNALS

- **Foreground Control (Keyboard):** Use key combinations for immediate action on the foreground process:
 - **Ctrl+z** to suspend.
 - **Ctrl+c** to kill (terminate).
 - **Ctrl+** to core dump.
- **Background Control (Command):** Signalling background processes requires a **signal-sending command**.
- **The kill Command:** Sends a specified signal (by **name** or **number**) to a process using its **PID**. It is used for *all* signals, not just killing.
- **Permissions:** You can kill your own processes, but **root privilege** is required to kill processes owned by other users.

```
[student@serverX ~]$ kill PID[student@serverX ~]$ kill -signal PID[student@serverX ~]$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM    15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT    19) SIGSTOP    20) SIGTSTP
-- output truncated --
```

Controlling Processes

Controlling Processes

kill -9 1234	# Force kill by PID
pkill -f "pattern"	# Kill by process name/pattern
kill -STOP 1234	# Pause process
kill -CONT 1234	# Resume process

LOGGING USERS OUT ADMINISTRATIVELY

- **Command:** The **w** command shows users currently logged into the system and their **cumulative activities**.
- **Location Identification:** Use the **TTY** and **FROM** columns to determine the user's location.
- **Terminal Types (TTY):**
 - **pts/N** (pseudo-terminal) for graphical environment windows.
 - **tty** for system console or direct device.
- **Remote Users:** Remote connections show the connecting system name in the **FROM** column when the **-f** option is used.

```
[student@serverX ~]$ w -f
12:43:06 up 27 min,  5 users,  load average: 0.03, 0.17, 0.66
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU   WHAT
student   :0        :0              12:20    ?xdm?  1:10   0.16s  gdm-session-wor
student   pts/0     :0              12:20    2.00s  0.08s  0.01s  w -f
root      tty2      :0              12:26    14:58  0.04s  0.04s  -bash
bob       tty3      :0              12:28    14:42  0.02s  0.02s  -bash
student   pts/1     desktop2.example.12:41  1:07    0.03s  0.03s  -bash
```

Monitor Process Activity

LOAD AVERAGE

- The **load average** metric is calculated by the Linux kernel as an **exponential moving average** of the **load number**.
- **Load Number**: This is a **cumulative CPU count** of **active system resource requests** (running threads and threads waiting for I/O).
 - The **load number** calculation runs every five seconds by default, accumulating and averaging active requests for all CPUs.
- **Exponential Moving Average (EMA)**: A mathematical formula used to smooth out data highs and lows, increase the significance of current activity, and decrease the quality of aging data.
- **Displayed Values**: The **load average** collectively refers to the three displayed values, which represent the system activity data averaged for the last **1, 5, and 15 minutes**.

Understanding the Linux load average calculation

- **Load Representation:** The **load average** represents the perceived system load, based on expected service wait times for **CPU, disk, and network I/O**.
- **Active Tasks:** Linux counts not only processes but also **threads individually** as separate tasks.
 - Threads counted include those **running (R)** and those **waiting for I/O resources (D)**, which corresponds to the **Uninterruptable Sleeping** state.
- **Load Number Calculation:** The **load number** is a **global calculation** summed for all CPUs, ensuring an accurate cumulative count even if tasks reschedule to different CPUs after sleep.
- **CPU Counting:** Linux counts each **physical CPU core** and **hyperthread** as a separate **individual CPU** with independent request queues.
 - The number of logical CPUs can be viewed via `/proc/cpuinfo`.

```
[student@serverX ~]$ grep "model name" /proc/cpuinfo
model name : Intel(R) Core(TM) i5 CPU           M 520  @ 2.40GHz
model name : Intel(R) Core(TM) i5 CPU           M 520  @ 2.40GHz
model name : Intel(R) Core(TM) i5 CPU           M 520  @ 2.40GHz
model name : Intel(R) Core(TM) i5 CPU           M 520  @ 2.40GHz
[student@serverX ~]$ grep "model name" /proc/cpuinfo | wc -l
4
```

Interpret Load Average Values

- **Values Meaning:** The three displayed values represent the **weighted values** over the last **1, 5, and 15 minutes**.
- **Monitoring Tools:** Commands like **top**, **uptime**, **w**, and the **gnome-system-monitor** display the load average values.
- **Per-CPU Calculation:** To determine if the system is experiencing significant waiting, the displayed load average should be **divided by the number of logical CPUs** in the system.
- **Interpretation Threshold:**
 - A value **below 1** indicates **satisfactory resource utilization** and minimal wait times.
 - A value **above 1** indicates **resource saturation** and some amount of service waiting times.

```
[student@serverX ~]$ uptime  
15:29:03 up 14 min,  2 users,  load average: 2.92, 4.48, 5.20
```

REAL-TIME PROCESS MONITORING

- The `top` **program** provides a **dynamic view** of the system's processes.
- **Output:** It displays a summary header followed by a process or thread list, similar to the information provided by `ps`.
- **Dynamic Nature:** Unlike the static output of `ps`, `top` continuously refreshes at a configurable interval.
- **Features:** `top` allows for column reordering, sorting, and highlighting, and user configurations can be saved.
- **Default Columns:** Recognizable columns include the **Process ID (PID)** and the **User name (USER)**, which is the process owner.

Linux Processes: Process Control

- ✓ A **process** is a running instance of an executable program. Processes are assigned a state, which can be running, sleeping, stopped, or zombie. The **ps** command lists processes.
- ❖ Each terminal has its own session and can have a foreground process and independent background processes. The jobs command displays processes within a terminal session.
- ✓ A **signal** is a software interrupt that reports events to an executing program. The **kill, pkill, and killall** commands use signals to control processes.
- ❖ **Load average** is an estimate of how busy the system is. To display load average values, you can use the **top, uptime, or w** command.



Thank You