# Linux Programming: Assignment-10

Samriddhi Guha | ENG24CY0157 | 3-C (25)

## Detailed Analysis of Working and History of Linux CLI Tools: mpstat and ps -ef

### Command 1: mpstat (Multiprocessor Statistics):-

#### 1. History, Invention, and Authors:

The *mpstat* command, which is part of the *sysstat* package, was developed in 1999 by French programmer Sebastien Godard, partially to fill a major monitoring gap with multi-core and Symmetric Multiprocessing (SMP) support in Linux servers.

Older tools like top could show global averages but would easily hide performance issues where one core was maxed out while others were idle. Godard was inspired by the System Activity Reporter (sar), included with commercial Unix operating systems like Solaris and AIX, and arranged to parse the kernel counters from /proc/stat to create the mpstat command. It is still used today for diagnosing per-processor imbalances and interrupt storms.

#### 2. Utility for System Administrators:

For a System Administrator, mpstat is indispensable for diagnosing specific types of performance degradation that other tools miss:

- **Detecting Single-Core Saturation:** A server might report only 25% total CPU usage (on a 4-core system) but actually have one core at 100% and three at 0%. mpstat reveals this imbalance immediately.
- **Identifying I/O Bottlenecks:** By monitoring the %iowait column, administrators can distinguish between a slow application (high %usr) and a slow hard drive (high %iowait).
- **Virtualization Analysis:** In cloud environments (AWS/Azure), the %steal metric informs the admin if the physical host is overloaded and stealing cycles from their Virtual Machine.

#### 3. How it Works:

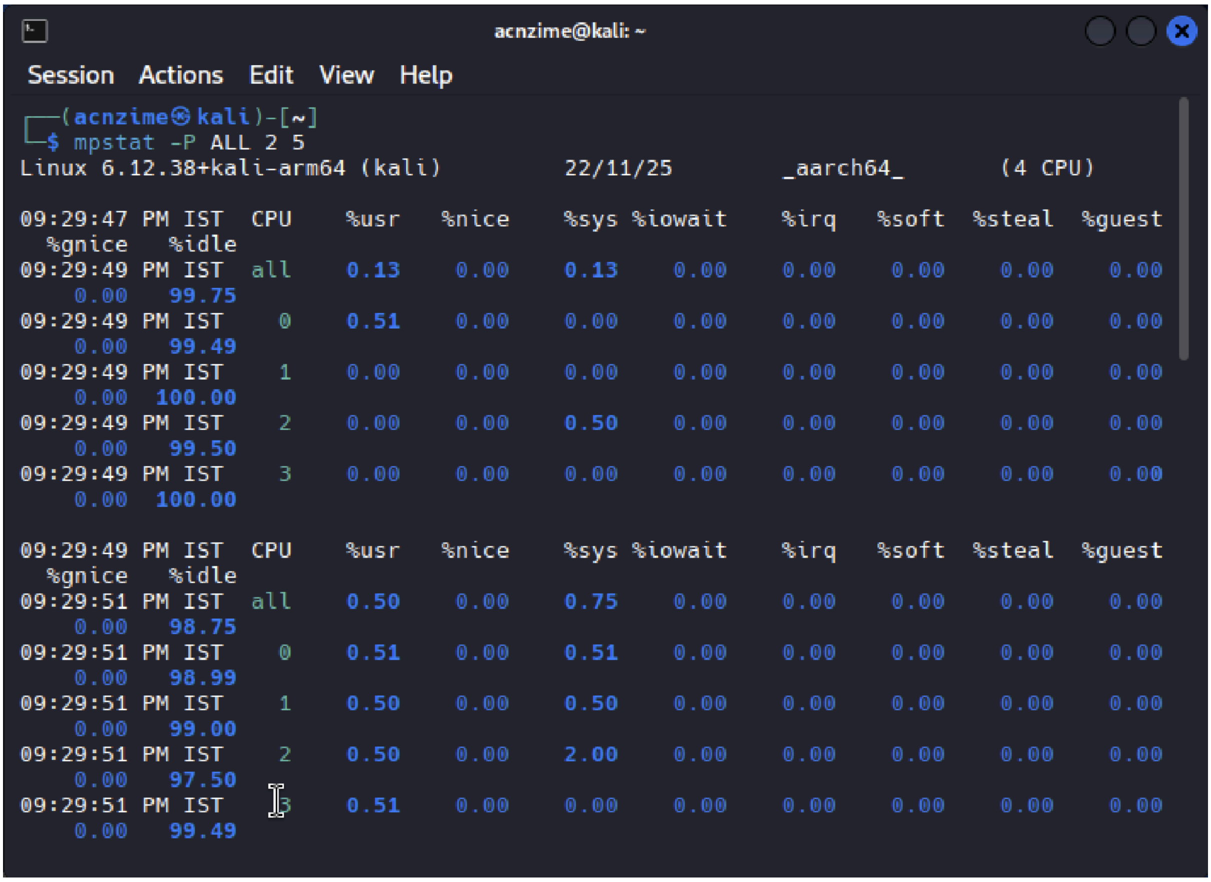
The mpstat command works by reading the kernel's process accounting file located at /proc/stat. This file contains raw counters of *"jiffies" (units of time)* the CPU has spent in various modes. mpstat takes two snapshots of this file over a specified interval and calculates the difference to determine usage percentages.

***Syntax: mpstat [interval] [count]***

Example: To monitor all processors with updates every 2 seconds, for a total of 5 reports,

```
mpstat -P ALL 2 5
```

## 4. Execution In CLI:



```
┌──(acnzime㉿kali)-[~]
└─$ mpstat -P ALL 2 5
Linux 6.12.38+kali-arm64 (kali)         22/11/25        _aarch64_       (4 CPU)

09:29:47 PM IST  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest
  %gnice   %idle
09:29:49 PM IST  all    0.13    0.00    0.13    0.00    0.00    0.00    0.00    0.00
    0.00   99.75
09:29:49 PM IST    0    0.51    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    0.00   99.49
09:29:49 PM IST    1    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    0.00  100.00
09:29:49 PM IST    2    0.00    0.00    0.50    0.00    0.00    0.00    0.00    0.00
    0.00   99.50
09:29:49 PM IST    3    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    0.00  100.00

09:29:49 PM IST  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest
  %gnice   %idle
09:29:51 PM IST  all    0.50    0.00    0.75    0.00    0.00    0.00    0.00    0.00
    0.00   98.75
09:29:51 PM IST    0    0.51    0.00    0.51    0.00    0.00    0.00    0.00    0.00
    0.00   98.99
09:29:51 PM IST    1    0.50    0.00    0.50    0.00    0.00    0.00    0.00    0.00
    0.00   99.00
09:29:51 PM IST    2    0.50    0.00    2.00    0.00    0.00    0.00    0.00    0.00
    0.00   97.50
09:29:51 PM IST    3    0.51    0.00    0.00    0.00    0.00    0.00    0.00    0.00
    0.00   99.49
```

## 5. Options and Flags:

- **-P {cpu_list | ALL}**: Tells mpstat to report statistics for specific processors (e.g., -P 0,2) or all the processors individually.
- **-A**: Equivalent to "All". This switches on almost all other flags, showing interrupts, soft interrupts, and CPU stats in one massive report.
- **-I {SUM | CPU | ALL}**: Reports interrupt statistics. SUM shows the total number of interrupts per processor, which is useful for debugging hardware failures.
- **-u**: Reports CPU utilization. This is the default behavior if no other flag is used.
- **-V**: Prints version number. Useful for checking compatibility with newer kernel features.
- **interval (parameter)**: The time in seconds between reports (e.g., mpstat 5 waits 5 seconds between data points).
- **count (parameter)**: The number of reports to generate before automatically exiting.

## Command 2: ps -ef (Process Status):-

### 1. History, Invention, and Authors:

The *ps (process status)* command is a foundational utility in computing, originally debuting in AT&T Unix Version 4 in the early 1970s. Its evolution is defined by the "Unix Wars" of the 1980s, a period of fragmentation that split the ecosystem into two competing lineages: the Berkeley Software Distribution (BSD) and System V (SysV) developed by AT&T.

The specific ps -ef syntax represents the AT&T System V tradition. Its hallmark is the strict use of dashed flags (e.g., -e to select "every" process). In contrast, BSD systems popularized "dash-less" arguments, leading to the alternative ps aux syntax. When POSIX standards were established in the 1990s to ensure software compatibility, they officially adopted the System V model. Consequently, while modern Linux distributions can interpret both styles via the procps-ng package, ps -ef remains the preferred syntax for enterprise administration and portable scripting across diverse platforms like Solaris, AIX, and HP-UX.

### 2. Utility for System Administrators:

ps -ef is the "eyes" of the administrator regarding running software. It is used to:

- **Kill unresponsive programs:** Admins use it to find the **PID** (Process ID) required to terminate a frozen application.
- **Verify Service Accounts:** It confirms if a secure service (like a web server) is running as a non-privileged user (e.g., www-data) rather than root, which is a security best practice.
- **Trace Parent/Child Relationships:** The **PPID** (Parent Process ID) column allows an admin to see exactly which process spawned another (e.g., knowing which terminal spawned a suspicious script).

### 3. How it Works:

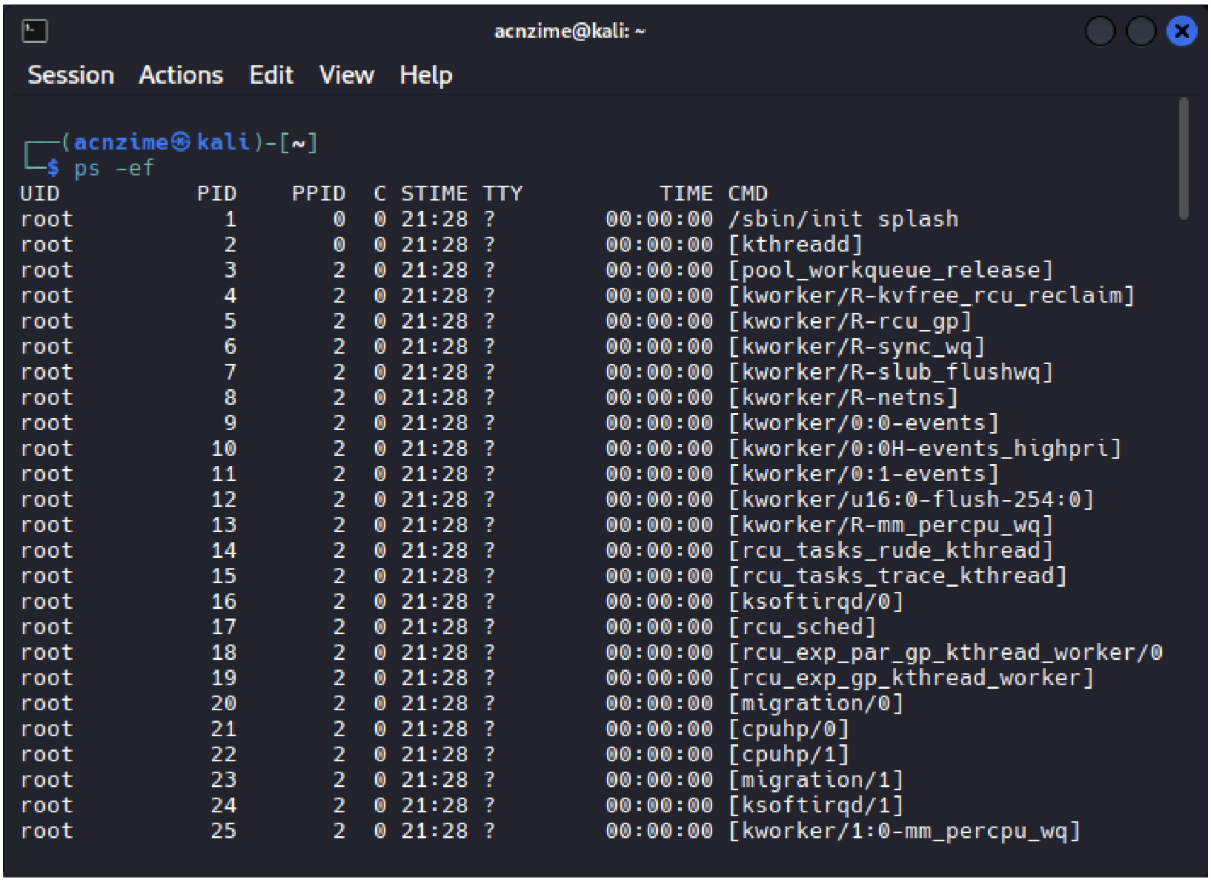
On Linux, ps does not query the kernel directly for process data. Instead, ***it parses the virtual /proc filesystem.*** The kernel exposes every running process as a directory (e.g., /proc/1234/). The ps command crawls these directories, reading files like /proc/[PID]/stat and /proc/[PID]/cmdline, and formats that data into a human-readable table.

***Basic Syntax: ps -ef***

Example: To view every process on the system in full format,

```
ps -ef
```

## 4. Execution In CLI:



```
┌──(acnzime㉿kali)-[~]
└─$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 21:28 ?        00:00:00 /sbin/init splash
root           2       0  0 21:28 ?        00:00:00 [kthreadd]
root           3       2  0 21:28 ?        00:00:00 [pool_workqueue_release]
root           4       2  0 21:28 ?        00:00:00 [kworker/R-kvfree_rcu_reclaim]
root           5       2  0 21:28 ?        00:00:00 [kworker/R-rcu_gp]
root           6       2  0 21:28 ?        00:00:00 [kworker/R-sync_wq]
root           7       2  0 21:28 ?        00:00:00 [kworker/R-slub_flushwq]
root           8       2  0 21:28 ?        00:00:00 [kworker/R-netns]
root           9       2  0 21:28 ?        00:00:00 [kworker/0:0-events]
root          10       2  0 21:28 ?        00:00:00 [kworker/0:0H-events_highpri]
root          11       2  0 21:28 ?        00:00:00 [kworker/0:1-events]
root          12       2  0 21:28 ?        00:00:00 [kworker/u16:0-flush-254:0]
root          13       2  0 21:28 ?        00:00:00 [kworker/R-mm_percpu_wq]
root          14       2  0 21:28 ?        00:00:00 [rcu_tasks_rude_kthread]
root          15       2  0 21:28 ?        00:00:00 [rcu_tasks_trace_kthread]
root          16       2  0 21:28 ?        00:00:00 [ksoftirqd/0]
root          17       2  0 21:28 ?        00:00:00 [rcu_sched]
root          18       2  0 21:28 ?        00:00:00 [rcu_exp_par_gp_kthread_worker/0
root          19       2  0 21:28 ?        00:00:00 [rcu_exp_gp_kthread_worker]
root          20       2  0 21:28 ?        00:00:00 [migration/0]
root          21       2  0 21:28 ?        00:00:00 [cpuhp/0]
root          22       2  0 21:28 ?        00:00:00 [cpuhp/1]
root          23       2  0 21:28 ?        00:00:00 [migration/1]
root          24       2  0 21:28 ?        00:00:00 [ksoftirqd/1]
root          25       2  0 21:28 ?        00:00:00 [kworker/1:0-mm_percpu_wq]
```

## 5. Options and Flags:

The flags for ps in this style are strictly POSIX/Unix System V compliant.

- -e: Select **Every** process. Without this, ps only shows processes associated with your current terminal session. (Synonymous with -A).
- -f: Full-format listing. This expands the output to include the UID, PPID, C, and STIME columns, which are omitted in the standard view.
- **-H**: Show hierarchy. This indents the child processes under their parents, creating a visual tree structure to easily understand process dependencies.
- **-u {userlist}**: Select by user. Example: ps -u alice will only show processes owned by the user 'alice'.
- **-p {pidlist}**: Select by PID. Example: ps -p 1234 is used to check the status of one specific known process ID.
- **-C {cmdlist}**: Select by command name. Example: ps -C chrome will show all processes running the executable named 'chrome'.