Processes & Init Systems

Session 8

Processes & Init Systems

Session 8

Objectives:

- Understand the role of the init system and what systemd does
- List and monitor processes using tools like ps, top, and htop
- Start, stop, enable, disable, and check services with systemctl
- Understand process IDs, foreground/background jobs, and signals
- Use commands like kill, nice, jobs, &, and killall to manage processes

What is systemd?

systemd: is the first process (PID 1) launched by the Linux kernel after boot

- Manages services, logging, dependencies, and system states
- It replaces older SysV init and offers faster boot, better logging, and cleaner service control
- Use unit file to define and manage a resource or service on a Linux system.

Each unit file tells systemd:

- What to start
- How to start it
- When to start it
- How to manage restarts
- What dependencies exist

Ex:

Type	Description	Example
.service	For running services or daemons	nginx.service
.timer	For scheduling tasks (like cron)	backup.timer
.mount	For managing mounted filesystems	home.moun

Controlling Services with systemctl

systemctl: is the Swiss army knife for systemd. You'll use it to start, stop, restart, enable, and inspect services.

Command	**What it does**
systemctl status	Show system-wide status summary
systemctl status <service>	Show detailed status of a service
systemctl start <service>	Start the service **now**
systemctl stop <service>	Stop the service **now**
systemctl restart <service>	Restart the service
systemctl enable <service>	Enable it to start on **boot**
systemctl disable <service>	Disable autostart at boot
systemctl list-units --type=service	Show all currently active services

Controlling Services with systemctl

Example: Let play with the ssh service which is **network service** that allows users to securely access a computer remotely over a network (more details in future lecture).

1. Check if ssh is running:
    - systemctl status ssh

    Output:

```
ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2025-05-02 10:43:21 CEST; 2h 16min ago
 Main PID: 791 (sshd)
    Tasks: 1 (limit: 4575)
   Memory: 2.1M
      CPU: 35ms
   CGroup: /system.slice/ssh.service
           └─791 /usr/sbin/sshd -D
```

Controlling Services with systemctl

Example: Let play with the ssh service which is **network service** that allows users to securely access a computer remotely over a network (more details in future lecture).

1. Verify if is it enable at boot:
   - systemctl is-enabled ssh

   Output:
   Enabled

2. Disable it (just for testing)
   - sudo systemctl disable ssh

3. Verify :
   - systemctl is-enabled ssh

   Output:
   Disabled

4. Enable it again :
   - sudo systemctl enable ssh

Controlling Services with systemctl

Example: Let play with the ssh service which is **network service** that allows users to securely access a computer remotely over a network (more details in future lecture).

1. Stop ssh:
   - sudo systemctl stop ssh

2. Verify
   - systemctl status ssh

   Output:

   ```
   ssh.service - OpenBSD Secure Shell server
       Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
       Active: inactive (dead) since Fri 2025-05-02 13:00:12 CEST; 2s ago
   ```

3. Start it again :
   - sudo systemctl start ssh

Viewing & Managing Processes

What is a process: A process is a running instance of a program. Every command you run (even ls, sleep, or bash) becomes a process.

Each process has:

- A PID (Process ID)
- A PPID (Parent Process ID)
- A status (running, sleeping, zombie, etc.)
- A user, CPU and memory usage

Viewing & Managing Processes

Commands:

Command	**Description**
ps aux	Show all running processes
ps -f	Full-format listing with parent info
top	Real-time monitoring
htop	Graphical process viewer
kill PID	Send signal to terminate a process
kill -9 PID	Forcefully stop a process
killall name	Kill all processes with a given name

Viewing & Managing Processes

EX: ps aux | head -5

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  22536  1208 ?        Ss   10:00   0:01 /sbin/init
user      1547  0.1  1.0  52864 10532 ?        Sl   10:01   0:12 /usr/bin/code
user      2000  0.0  0.2  12000  1500 ?        S    10:10   0:00 bash
```

ps	The process status command — used to list running processes
aux	Options to show all processes in a detailed, non-truncated format
head -5	Displays only the first 5 lines of the output

Viewing & Managing Processes

EX: ps aux | head -5

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  22536  1208 ?        Ss   10:00   0:01 /sbin/init
user      1547  0.1  1.0  52864 10532 ?        Sl   10:01   0:12 /usr/bin/code
user      2000  0.0  0.2  12000  1500 ?        S    10:10   0:00 bash
```

Column	**Meaning**
USER	The user who owns the process
PID	Process ID
%CPU	CPU usage
%MEM	Memory usage
VSZ	Virtual memory size
RSS	Resident memory size
STAT	Process state
START	Time the process started
TIME	CPU time consumed
COMMAND	Command used to start the process

Viewing & Managing Processes

EX: Code explanation for the program status

Code	**Meaning**
R	Running — currently using CPU
S	Sleeping — waiting for event (e.g., input, time)
D	Uninterruptible sleep — usually waiting for I/O (disk, network)
Z	Zombie — completed but not cleaned up by parent
T	Stopped — paused (e.g., Ctrl+Z)
X	Dead — defunct, should not appear
I	Idle — (kernel thread) only in some systems like htop

Flag	Meaning
s	Session leader (usually a shell)
l	Multi-threaded process (uses clone())
+	Process is in the foreground process group

Viewing & Managing Processes

EX: Kill a program

1. Start a dummy background program
   sleep300 &

2. ps aux

3. How to find directly your program?

Viewing & Managing Processes

EX: Kill a program

1. Start a dummy background program
   sleep300 &

2. ps aux

3. How to find directly your program?
   - ps aux | grep sleep

4. kill <PID>
5. Rerun ps aux and check that there is nothing

top vs ps

Feature	**ps aux**	**top**
📷 Snapshot	One-time snapshot of processes	Real-time, continuously updating view
🧾 Output	Printed list (can be piped or saved)	Interactive full-screen interface
🔧 Usage	Useful for scripting, filtering	Great for live monitoring
🔁 Refresh	Does **not** update unless re-run	Auto-refreshes every 1–3 seconds
🧠 Sorting	Manual (sort) or via ps options	Built-in: press keys to sort by CPU, memory, etc.

In top:

Press k to kill a process
Press P to sort by CPU
Press M to sort by memory
Press q to quitround process group

Foreground & Background Jobs

Key Concepts:

- A foreground job runs in the terminal and blocks it until it finishes.
- A background job runs "behind the scenes" while you still use the terminal.
- You can pause, resume, or bring back background jobs.
- Jobs are tracked with job numbers like %1, %2, etc
- What is a Linux Daemon?
    - A system daemon in Linux is typically a background system process that awaits a specific set of conditions before jumping into action

Foreground & Background Jobs

Key Commands:

Command	**Description**
command &	Run command in the background
jobs	List current background and stopped jobs
fg %n	Bring job number n to the foreground
bg %n	Resume a stopped job in the background
CTRL+Z	Suspend (pause) the current foreground process
CTRL+C	Terminate the foreground process
kill %n	Send a signal to a job using its job number

Foreground & Background Jobs

EX

1. Start a dummy background program

   ```
   sleep300 &
   Output: [1] 3456
   ```

   Job [1] has process ID 3456.

2. Check Background Jobs

   ```
   jobs
   Output: [1]+  Running              sleep 300 &.
   ```

3. Move it to the foreground

   ```
   fg %1
   ```

4. Suspend a Foreground Job

   ```
   CTRL+Z
   Output [1]+  Stopped              sleep 300
   ```

Signals & Process Priorities

- Every process can receive signals: special instructions from the OS or user
- Some signals can stop, pause, or restart a process
- Linux processes also have a priority (called nice value) that affects CPU scheduling

Signals

🚦 Signals: Common Examples

Signal	**Code**	**Meaning**
SIGTERM	15	Terminate the process gracefully
SIGKILL	9	Force kill the process immediately
SIGINT	2	Sent by CTRL+C in the terminal
SIGSTOP	19	Pause a process (like CTRL+Z)
SIGCONT	18	Resume a paused process

Signals & Process Priorities

🚦 Signals: How to send a signal

Command	**Description**
kill -15 PID (default)	Graceful termination
kill -9 PID	Force kill using SIGKILL
kill -l	List all available signals
killall name	Kill all processes with that name
trap	Catch or react to signals in a script

Signals

Exemples:

```
sleep 300 &
ps aux | grep sleep
```

- Graceful Kill
  - kill -15 <PID>

- Force Kill
  - kill -9 <PID>

Process reaction

Can handle it: clean up, save data, exit

vs

Cannot handle it: terminated instantly

Process Priorities

Priority affects who gets CPU time first. Use nice to be polite with long-running background jobs.

Concept	**Description**
nice	Set process priority when launching it
renice	Change priority of a running process
Priority	From -20 (highest) to 19 (lowest priority)
Default	Most user processes start at priority 0

Process Priorities

Exemples: Runs sleep with lower priority

```
nice -n 10 sleep 1000 &
ps aux | grep sleep
```

Check:

```
ps -o pid,ni,cmd -p <PID>
```

Rechange priority:

```
renice 5 <PID>
```

Creating a Custom systemd Service

- A service is a background program managed by systemd
- You can create your own service to automatically run your script at boot or on demand
- A unit file defines how the service should behave

Why?

You’ve written a script that logs temperature data every 30 seconds to a CSV file:

```
#!/bin/bash
while true; do
  echo "$(date), $(read_temp_sensor)" >> /var/log/temps.csv
  sleep 30
done
```

❓ Problem:

If you run this script manually:

- It stops when the terminal closes
- It doesn't restart after reboot
- If it crashes, it doesn’t come back

Why?

✅ Solution: Turn it into a systemd service

- It starts automatically at boot
- Runs in the background continuously
- Restarts itself if it crashes
- Logs errors with journalctl

```
[Unit]
Description=Temperature Logger

[Service]
ExecStart=/usr/local/bin/log_temp.sh
Restart=always

[Install]
WantedBy=multi-user.target #“Enable this service when the system reaches multi-user mode
```

Exemple

- **Step 1 – Create a script**

Create /usr/local/bin/hello_loop.sh:

```
#!/bin/bash
while true; do
  echo "Hello from systemd at $(date)" >> /tmp/hello.log
  sleep 30
done
chmod +x /usr/local/bin/hello_loop.sh
```

Exemple

- **Step 2 – Create the service unit file**

Create /etc/systemd/system/hello.service

```
[Unit]
Description=My Hello Logging Service

[Service]
ExecStart=/usr/local/bin/hello_loop.sh
Restart=always

[Install]
WantedBy=multi-user.target
```

Exemple

- **Step 3 – Reload systemd and start the service**

```
sudo systemctl daemon-reload
sudo systemctl start hello.service
sudo systemctl enable hello.service
```

- **Check status and logs**

```
systemctl status hello.service
tail -f /tmp/hello.log
```