1- Run a background task to simulate sensor polling.

```
ghannam@Ghannam:~$ ls &
[1] 4265
ghannam@Ghannam:~$ client-server-mqtt  Desktop  Documents  Downloads  host_share  iot_logger  mqttfx  MQTT-FX  Music  Pictures  Public  ros2_ws  scripts,  snap  Templates  venvs
s
fg
bash: fg: job has terminated
[1]+  Done                    ls --color=auto
ghannam@Ghannam:~$
```

2- List processes and filter for the background task.

```
ghannam@Ghannam:~$ ps -e
    PID TTY          TIME CMD
      1 ?        00:00:01 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 pool_workqueue_release
      4 ?        00:00:00 kworker/R-rcu_gp
      5 ?        00:00:00 kworker/R-sync_wq
      6 ?        00:00:00 kworker/R-kvfree_rcu_reclaim
      7 ?        00:00:00 kworker/R-slub_flushwq
      8 ?        00:00:00 kworker/R-netns
      9 ?        00:00:00 kworker/0:0-events
     10 ?        00:00:00 kworker/0:0H-events_highpri
     12 ?        00:00:00 kworker/u32:0-ipv6_addrconf
     13 ?        00:00:00 kworker/R-mm_percpu_wq
     14 ?        00:00:00 rcu_tasks_kthread
     15 ?        00:00:00 rcu_tasks_rude_kthread
     16 ?        00:00:00 rcu_tasks_trace_kthread
     17 ?        00:00:00 ksoftirqd/0
     18 ?        00:00:00 rcu_preempt
     19 ?        00:00:00 rcu_exp_par_gp_kthread_worker/0
     20 ?        00:00:00 rcu_exp_gp_kthread_worker
     21 ?        00:00:00 migration/0
     22 ?        00:00:00 idle_inject/0
     23 ?        00:00:00 cpuhp/0
     24 ?        00:00:00 cpuhp/1
     25 ?        00:00:00 idle_inject/1
     26 ?        00:00:00 migration/1
     27 ?        00:00:00 ksoftirqd/1
     29 ?        00:00:00 kworker/1:0H-events_highpri
     30 ?        00:00:00 cpuhp/2
     31 ?        00:00:00 idle_inject/2
     32 ?        00:00:00 migration/2
     33 ?        00:00:00 ksoftirqd/2
     36 ?        00:00:00 cpuhp/3
```

```
fg
bash: fg: job has terminated
[1]+  Done                    ls --color=auto
ghannam@Ghannam:~$
```

3- Check network states (established connections).

```
ghannam@Ghannam:~$ netstat -tuna | grep ESTAB
udp        0      0 192.168.122.198:68      192.168.122.1:67        ESTABLISHED
ghannam@Ghannam:~$ ss -tuna | grep ESTAB
udp   ESTAB  0      0      192.168.122.198%enp1s0:68    192.168.122.1:67
ghannam@Ghannam:~$
```

4- Try foreground and background switching.

```
ghannam@Ghannam:~$ sleep 200
^Z
[1]+  Stopped                 sleep 200
ghannam@Ghannam:~$ bg
[1]+ sleep 200 &
ghannam@Ghannam:~$ jobs
[1]+  Running                 sleep 200 &
ghannam@Ghannam:~$ fg %1
sleep 200
^C
ghannam@Ghannam:~$
```

5- kill a process if needed

```
ghannam@Ghannam:~$ sleep 200 &
[1] 4842
ghannam@Ghannam:~$
ghannam@Ghannam:~$ jobs
[1]+  Running                 sleep 200 &
```

```
ghannam@Ghannam:~$ kill -9 %1
ghannam@Ghannam:~$ jobs
[1]+  Killed                  sleep 200
ghannam@Ghannam:~$
```

1. Keystrokes → bash
Your terminal emulator sends the line through a pty to bash.
When you press Enter, bash reads the line (via Readline: history, keybindings, etc.).
2. Parse & expand
Bash tokenizes and applies expansions in this rough order: alias → brace → tilde → parameter $VAR → command substitution $(...) → arithmetic → word splitting → pathname globbing * ? [] → quote removal.
It decides the command form (simple, pipeline, redirections, background &, etc.).
3. Command lookup
Bash checks: functions → builtins → hashed paths → $PATH.
ls is usually /usr/bin/ls (GNU coreutils). Try: type -a ls.
4. Set up I/O and jobs
If you used redirections/pipes, bash opens files/creates pipes and assigns FDs (stdin=0, stdout= stderr=2).
If backgrounded, it sets job control (process group, notifications).
5. Create the process
Bash fork()s (or posix_spawn).
In the child: apply redirections (dup2), set process group, restore default signals, inherit cwd, env, umask, rlimits.
6. Execute the program
Child calls execve("/usr/bin/ls", argv, envp).
Kernel checks execute permission & file type:
ELF binary → load program segments, start the dynamic linker.
Script with shebang → exec the interpreter from #!.
7. Dynamic linking (if needed)
The loader (ld-linux...) maps shared libs (e.g., glibc), resolves symbols, then jumps to program entry → main().
8- Program runs (ls)
ls stats/reads the directory (e.g., getdents64), may query terminal size (ioctl(TIOCGWINSZ)) and detect TTY to choose columns/colors.
It formats the listing.
9. Writing output
ls writes to stdout (write() syscalls).
If to a terminal: goes to the pty; your terminal renders it.
If piped to another command: data flows through the pipe buffer.
10. Exit & reap
ls calls exit(status). Kernel sends SIGCHLD to bash.
Bash waitpid() reaps the child, sets $? to the exit status, and prints the next prompt ($PS1).

1. Daemon process
Definition:
A background process detached from any terminal, running silently to provide services (e.g., sshd, systemd).
Characteristics:
Parent is usually init (PID 1) or systemd.
Keeps running until stopped/restarted.
Has no controlling terminal.
Detection:
```
ps -ef | grep sshd
```

2. Zombie process
Definition:
A process that has finished execution but still has an entry in the process table because the parent hasn't read its exit status (via wait()).
Characteristics:
State = Z in process status.
Takes no CPU, but still consumes a PID.
Detection:
```
ps aux | grep 'Z'
```

3. Orphan process
Definition:
A process whose parent has exited, leaving it running. The kernel reassigns it to init (PID 1).
Characteristics:
Parent = init or systemd.
Still active, unlike zombies.
Detection:
```
ps -ef | grep <process_name>
```

Why do we need IPC?
In Linux, many programs (processes) run at the same time.
Sometimes they need to talk to each other or share data.
That's what Inter-Process Communication (IPC) is for.
Example:
A web browser asks a video player process to play YouTube audio.
A client program sends data to a server program.
Without IPC, each process would be isolated and couldn't cooperate.