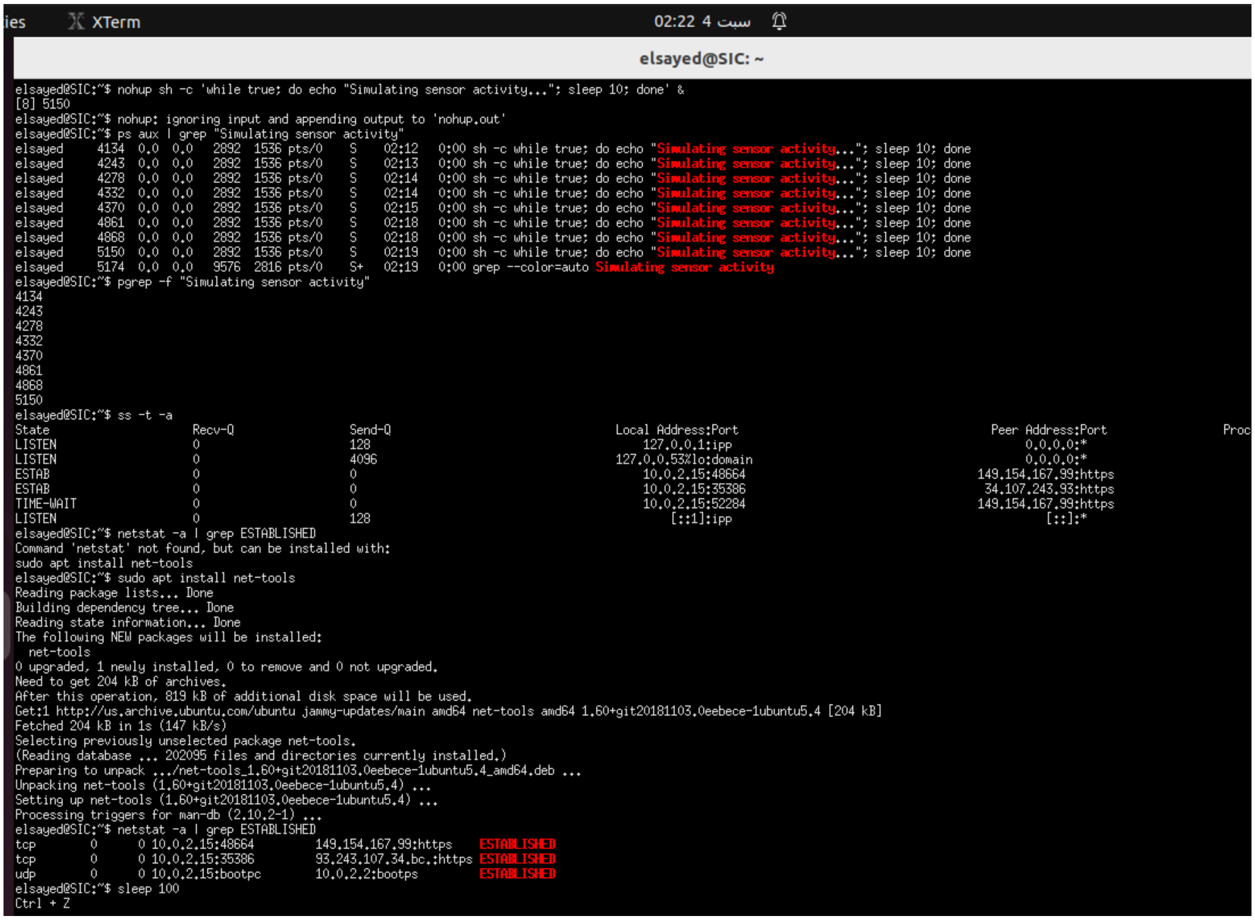
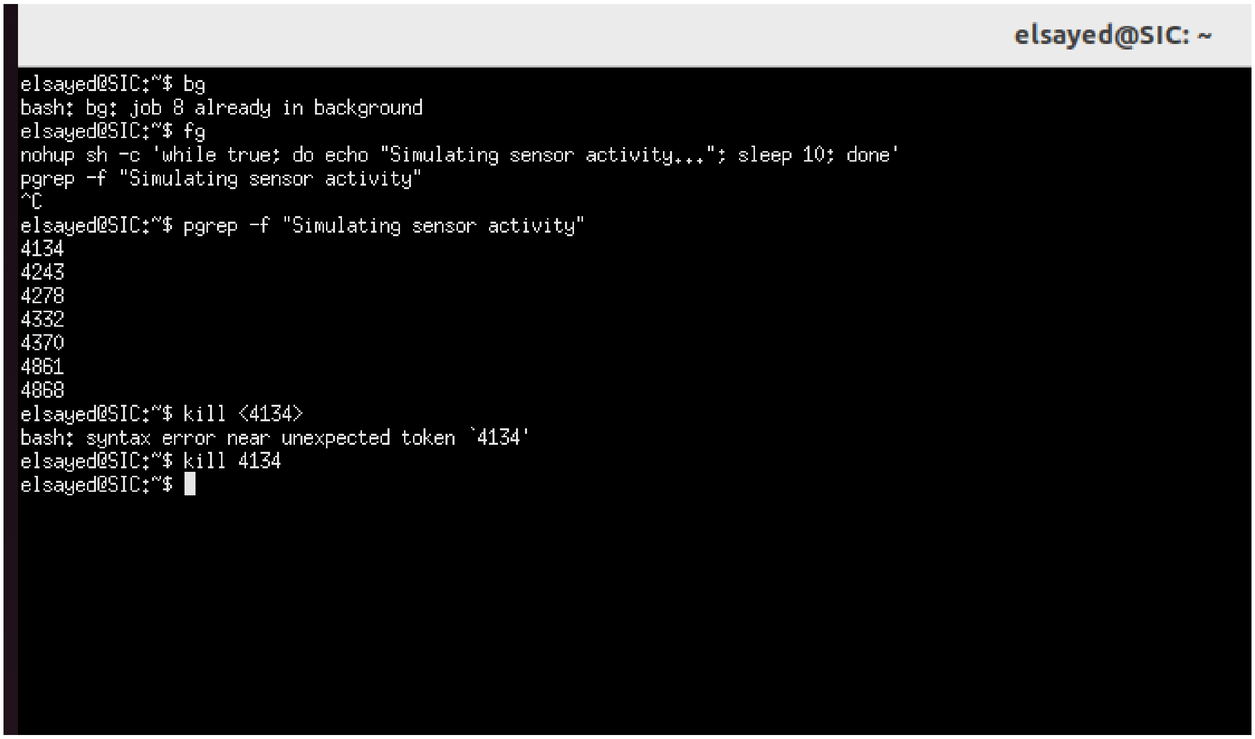


```
elsayed@SIC:~$ nohup sh -c 'while true; do echo "Simulating sensor activity..."; sleep 10; done' &
[8] 5150
elsayed@SIC:~$ nohup: ignoring input and appending output to 'nohup.out'
elsayed@SIC:~$ ps aux | grep "Simulating sensor activity"
elsayed     4134  0.0  0.0   2892  1536 pts/0    S    02:12   0:00 sh -c while true; do echo "Simulating sensor activity..."; sleep 10; done
elsayed     4243  0.0  0.0   2892  1536 pts/0    S    02:13   0:00 sh -c while true; do echo "Simulating sensor activity..."; sleep 10; done
elsayed     4278  0.0  0.0   2892  1536 pts/0    S    02:14   0:00 sh -c while true; do echo "Simulating sensor activity..."; sleep 10; done
elsayed     4332  0.0  0.0   2892  1536 pts/0    S    02:14   0:00 sh -c while true; do echo "Simulating sensor activity..."; sleep 10; done
elsayed     4370  0.0  0.0   2892  1536 pts/0    S    02:15   0:00 sh -c while true; do echo "Simulating sensor activity..."; sleep 10; done
elsayed     4861  0.0  0.0   2892  1536 pts/0    S    02:18   0:00 sh -c while true; do echo "Simulating sensor activity..."; sleep 10; done
elsayed     4868  0.0  0.0   2892  1536 pts/0    S    02:18   0:00 sh -c while true; do echo "Simulating sensor activity..."; sleep 10; done
elsayed     5150  0.0  0.0   2892  1536 pts/0    S    02:19   0:00 sh -c while true; do echo "Simulating sensor activity..."; sleep 10; done
elsayed     5174  0.0  0.0   9576  2816 pts/0    S+   02:19   0:00 grep --color=auto Simulating sensor activity
elsayed@SIC:~$ pgrep -f "Simulating sensor activity"
4134
4243
4278
4332
4370
4861
4868
5150
elsayed@SIC:~$ ss -t -a
State                Recv-Q               Send-Q                         Local Address:Port                          Peer Address:Port              Proc
LISTEN               0                    128                              127.0.0.1:ipp                               0.0.0.0:*
LISTEN               0                    4096                         127.0.0.53%lo:domain                            0.0.0.0:*
ESTAB                0                    0                                10.0.2.15:48664                       149.154.167.99:https
ESTAB                0                    0                                10.0.2.15:35386                        34.107.243.93:https
TIME-WAIT            0                    0                                10.0.2.15:52284                       149.154.167.99:https
LISTEN               0                    128                                  [::1]:ipp                                  [::]:*
elsayed@SIC:~$ netstat -a | grep ESTABLISHED
Command 'netstat' not found, but can be installed with:
sudo apt install net-tools
elsayed@SIC:~$ sudo apt install net-tools
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  net-tools
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 204 kB of archives.
After this operation, 819 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu jammy-updates/main amd64 net-tools amd64 1.60+git20181103.0eebece-1ubuntu5.4 [204 kB]
Fetched 204 kB in 1s (147 kB/s)
Selecting previously unselected package net-tools.
(Reading database ... 202095 files and directories currently installed.)
Preparing to unpack .../net-tools_1.60+git20181103.0eebece-1ubuntu5.4_amd64.deb ...
Unpacking net-tools (1.60+git20181103.0eebece-1ubuntu5.4) ...
Setting up net-tools (1.60+git20181103.0eebece-1ubuntu5.4) ...
Processing triggers for man-db (2.10.2-1) ...
elsayed@SIC:~$ netstat -a | grep ESTABLISHED
tcp        0      0 10.0.2.15:48664         149.154.167.99:https    ESTABLISHED
tcp        0      0 10.0.2.15:35386         93.243.107.34.bc.:https ESTABLISHED
udp        0      0 10.0.2.15:bootpc        10.0.2.2:bootps         ESTABLISHED
elsayed@SIC:~$ sleep 100
Ctrl + Z
```



```
elsayed@SIC:~$ bg
bash: bg: job 8 already in background
elsayed@SIC:~$ fg
nohup sh -c 'while true; do echo "Simulating sensor activity..."; sleep 10; done'
pgrep -f "Simulating sensor activity"
^C
elsayed@SIC:~$ pgrep -f "Simulating sensor activity"
4134
4243
4278
4332
4370
4861
4868
elsayed@SIC:~$ kill <4134>
bash: syntax error near unexpected token `4134'
elsayed@SIC:~$ kill 4134
elsayed@SIC:~$ 
```

1) What happens step-by-step when you type `ls` in bash?

1. **Readline** reads your keystrokes; bash gets the command line.
2. **Parsing & expansions**: history (`!`), aliases, variables (`$VAR`), command substitution (`$(...)`), globbing (`*`), quoting, redirections.
3. **Command lookup**: bash checks builtins; if external, searches `$PATH` (often cached in a hash).
4. **Fork/exec**: bash **forks** a child; the child sets redirections, signals, process group.
5. **execve()**: kernel replaces the child with `/bin/ls`.
6. **Program load**: ELF loader + dynamic linker map `ls` and shared libs (e.g., libc).
7. **Run & I/O**: `ls` writes to **stdout (fd 1)**, which is your terminal device (`/dev/pts/N`).
8. **Exit & status**: `ls` exits; bash **waits** and stores the status in `$?`.

---

2) Types of processes: daemon, zombie, orphan — and how to detect them

- **Daemon**: background service detached from a TTY (often started by `systemd`).
  Detect: no controlling TTY; see with `ps -eo pid,ppid,tty,stat,cmd | grep -v TTY`.
- **Zombie**: process has exited but parents haven't reaped it yet. Shows **`Z`** in `STAT`.
  Detect: `ps -eo pid,ppid,stat,cmd | awk '$3 ~ /Z/ {print}'`
- **Orphan**: parent dies while child keeps running; it's adopted by PID 1 (systemd).
  Detect: `ps -eo pid,ppid,stat,cmd | awk '$2==1 {print}'` (look for ones you expect to have a different parent).

---

3) Why do we need IPC? Common mechanisms + real-life examples

Processes are isolated for safety; IPC lets them **share data and coordinate**.

Mechanisms

- **Pipes** (`|`) & **FIFOs** (`mkfifo`) → shell pipelines (`dmesg | less`).
- **Signals** (`kill -TERM PID`) → control/notify processes (graceful shutdown).
- **Sockets** (Unix/TCP/UDP) → client/server apps (Nginx ↔ app; SSH).
- **Message queues** (POSIX/System V) → structured messages between procs.
- **Shared memory** + **semaphores/mutexes** → high-speed data (video frames).
- **mmap files** → map files into memory (databases, compilers).
- **D-Bus** → desktop/system services talk (NetworkManager, systemd).

Examples

- Web server ↔ database over TCP socket.

- `journalctl` reading logs via the journald Unix socket.
- Sensor collector writing to shared memory; analytics reader consuming it.
- Microservices communicating over message queues (e.g., Redis, RabbitMQ).