

## Day 4 – Phase 4: Process and Network Monitoring

- 1) I have the “x” letter not working on the laptop, that's why I did not use the nano command; however, I used the echo one.

```
aya@aya-VirtualBox: ~/Desktop

aya@aya-VirtualBox:~/Desktop$ echo '#!/bin/bash' > sensor_poll.sh
echo 'while true' >> sensor_poll.sh
echo 'do' >> sensor_poll.sh
echo '    echo "Assuming it is a sensor data at $(date)" >> sensor.log' >> sensor_poll.sh
echo '    sleep 5' >> sensor_poll.sh
echo 'done' >> sensor_poll.sh
aya@aya-VirtualBox:~/Desktop$ cat sensor_poll.sh
#!/bin/bash
while true
do
    echo "Assuming it is a sensor data at $(date)" >> sensor.log
    sleep 5
done
aya@aya-VirtualBox:~/Desktop$ chmod +x sensor_poll.sh
chmod: cannot access 'sensor_poll.sh': No such file or directory
aya@aya-VirtualBox:~/Desktop$ chmod +x sensor_poll.sh
aya@aya-VirtualBox:~/Desktop$ ls -l
total 16
drwxrwxr-x 3 aya aya 4096 22:55 30 文件 gitdemo
-rwxrwxr-x 1 aya aya 108 22:35 31 文件 sensor_poll.sh
-rw----- 1 aya aya 2602 23:07 30 文件 ssh
-rw-r--r-- 1 aya aya 572 23:07 30 文件 ssh.pub
aya@aya-VirtualBox:~/Desktop$ ./sensor_poll.sh &
[1] 11449
```

- 2)

```
aya@aya-VirtualBox: ~/Desktop

aya@aya-VirtualBox:~/Desktop$ ps -f | grep sensor_poll.sh
aya      11449   11023   0 22:38 pts/0    00:00:00 /bin/bash ./sensor_poll.sh
aya      11697   11023   0 22:41 pts/0    00:00:00 grep --color=auto sensor_poll.sh
```

- 3)

```
aya@aya-VirtualBox: ~/Desktop

aya@aya-VirtualBox:~/Desktop$ netstat
Command 'netstat' not found, but can be installed with:
sudo apt install net-tools
aya@aya-VirtualBox:~/Desktop$ sudo apt install net-tools
[sudo] password for aya:
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 1 not upgraded.
Need to get 204 kB of archives.
After this operation, 819 kB of additional disk space will be used.
Get:1 http://eg.archive.ubuntu.com/ubuntu jammy-updates/main amd64 net-tools amd64 1.60+git20181103.0eebece-1ubuntu5.4 [204 kB]
Fetched 204 kB in 3s (80.9 kB/s)
Selecting previously unselected package net-tools.
(Reading database ... 203016 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) ...
aya@aya-VirtualBox:~/Desktop$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0 aya-VirtualBox:39562    ubuntu-mirror-1.ps:htp TIME_WAIT
tcp        0      0 0 aya-VirtualBox:45256    mrs08s19-in-f3.1e:htps ESTABLISHED
tcp        0      0 0 aya-VirtualBox:45248    mrs08s19-in-f3.1e:htps ESTABLISHED
tcp        0      0 0 aya-VirtualBox:57872    217.138.110.34.bc:htps ESTABLISHED
tcp        0      0 0 aya-VirtualBox:59886    mrs09s14-in-f10.1:htps ESTABLISHED
tcp        0      0 0 aya-VirtualBox:53372    93.243.107.34.bc:htps ESTABLISHED
udp        0      0 0 aya-VirtualBox:bootpc   _gateway:bootps        ESTABLISHED
udp6       0      0 0 aya-VirtualBox:51581    fd17:625c:f037:2:domain ESTABLISHED
udp6       0      0 0 aya-VirtualBox:37267    fd17:625c:f037:2:domain ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State      I-Node   Path
unix    3      [ ]         STREAM     CONNECTED  11809
unix    3      [ ]         STREAM     CONNECTED  7039
unix    3      [ ]         STREAM     CONNECTED  50582    /run/dbus/system_bus_socket
unix    3      [ ]         STREAM     CONNECTED  7562     /run/systemd/journal/stdout
unix    2      [ ]         DGRAM      CONNECTED  11906
```

4)

```
aya@aya-VirtualBox: ~/Desktop
aya@aya-VirtualBox:~/Desktop$ jobs
[1]+  Running                  ./sensor_poll.sh &
aya@aya-VirtualBox:~/Desktop$ fg %1
./sensor_poll.sh
^C
```

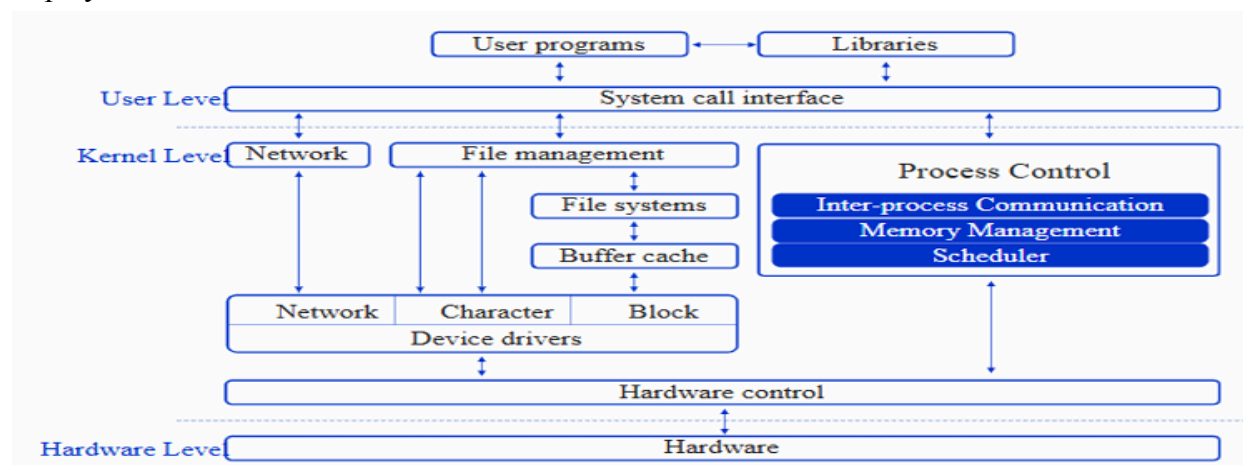
5)

```
aya@aya-VirtualBox: ~/Desktop
aya@aya-VirtualBox:~/Desktop$ ./sensor_poll.sh &
[2] 14123
aya@aya-VirtualBox:~/Desktop$ kill 14123
aya@aya-VirtualBox:~/Desktop$ ps -a
  PID TTY          TIME CMD
  1226 tty2      00:00:00 gnome-session-b
  14086 pts/0      00:00:00 sensor_poll.sh
  14151 pts/0      00:00:00 sleep
  14168 pts/0      00:00:00 ps
[2]+  Terminated              ./sensor_poll.sh
aya@aya-VirtualBox:~/Desktop$ kill 14086 14151
bash: kill: (14151) - No such process
aya@aya-VirtualBox:~/Desktop$ ps-a
ps-a: command not found
[1]+  Terminated              ./sensor_poll.sh
aya@aya-VirtualBox:~/Desktop$ ps-a
ps-a: command not found
aya@aya-VirtualBox:~/Desktop$ ps -a
  PID TTY          TIME CMD
  1226 tty2      00:00:00 gnome-session-b
  14373 pts/0      00:00:00 ps
```

## Open-Ended Questions

- 1) If we take the linux architecture as a reference, firstly for the user level (Command Entry), I write 'ls' in the bash terminal which receives the , and input parses the command to identify its name and any arguments. Then, the shell processing where the bash checks if 'ls' is a built-in command (it's not) and it searches for the 'ls' executable in directories listed in the '\$PATH' environment variable. For the System Call Interface Transition, the bash makes system calls to create a new process: 'fork()' - creates a child process, -'execve()' - replaces the child process with the 'ls' program in which those transitions (from User Level to Kernel Level) is done through the system calls (c functions). Regarding the Kernel Level (Process Manager), it creates a new process control block (PCB), allocates process ID (PID), and sets up memory space for the new process "process control subsystem". Then, the scheduler decides when the 'ls' process gets CPU time, and the memory management allocates virtual memory pages for the process. Afterwards, we will be in the "File System Operations" layer in which the 'ls' program needs to read directory contents, makes system calls like 'opendir()', 'readdir()', 'stat()' so that the file management subsystem processes these calls and the file systems layer accesses the actual directory data. The Hardware Interaction, the communication between the kernel layer and the hardware is done through the usage of the application binary interface (ABI). the Output Generation, the 'ls' formats the directory listing in user space, the program writes output using 'write()' system calls to stdout with the Character Device Drivers handle terminal output and finally, the Text flows through the terminal emulator back to the screen. Eventually, there is Process Cleanup in which when the 'ls' process completes and calls 'exit()', the Process Control cleans up process resources, Memory Management frees allocated memory, Control returns to the parent bash process, and the Bash displays the command prompt again.
  - a) The entire process involves data flowing up and down through all the architecture layers, from the user input at the user level, through system calls to kernel space, potentially down to hardware for disk access, and back up through the stack to display results on the screen.

b)



- 2) The demon: is a long-running background process that runs without the need for a terminal to show its output on such as systemd (system daemon “init”). It can be detected by “ps -ef | grep sshd”.
  - a) The zombie: it is when the parent does not care about the removal of the child resources after it dies (finishes its work) as if there is a child process running and finished so the resources do exist as the parent does not delete them until the kernel or the systemd (big parent) delete those resources. It can be detected by “ps aux | grep Z”.
  - b) The orphan: is a process running where the parent dies while the child process is still running. It can be detected by “ps -ef” to check the processes with PID=1.
- 3) In Linux, processes usually run independently; however, in many situations, they need to share data, coordinate tasks, or signal events and that’s why we need IPC. Without IPC, each process would be isolated, making it impossible to build modern systems where multiple processes must work together.

Main reasons we need IPC:

- a) Data sharing: processes need to exchange information.
- b) Synchronization: processes must not conflict (e.g., two processes writing to the same file).
- c) Event notification: one process may need to notify another (e.g., server notifies client).
- d) Resource sharing: coordinate access to shared resources (e.g., database, hardware).
- e) Modularity: break a big program into multiple processes that communicate efficiently.
- f) IPC Mechanisms in Linux: pipes which is a one-way communication channel between processes (example: `ls | grep txt` (the output of `ls` goes through a pipe to `grep`)).