System and Network Administration - Lab 7 - Processes and signals


```
Jaffar Totanji - j.totanji@innopolis.university
```

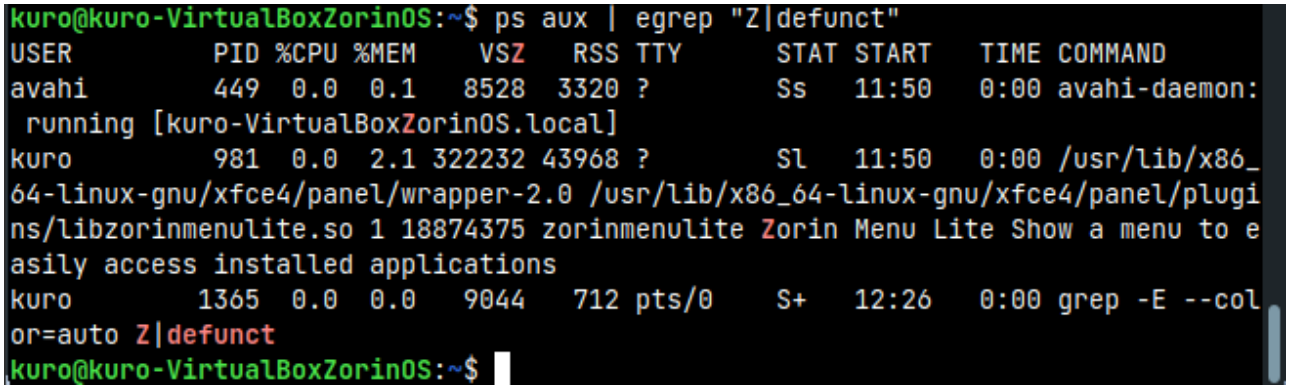

Questions to answer:

1. A zombie process is a process whose execution is completed but still has an entry in the process table. Zombie processes usually occur for child processes, as the parent process still needs to read its child's exit status. Once this is done using the wait system call, the zombie process is eliminated from the process table. This is known as reaping the zombie process.

   To kill such processes, we first need to find them. We can do that using:

   ```
   ps aux | egrep "Z|defunct"
   ```

   `ps aux` shows all our processes in a human-readable format with useful columns. `egrep` is then used to match an extended regular expression `"Z|defunct"` which matches zombie processes and anything with a `Z` in it. Then, we can select a zombie process that we want to kill and copy its `PID`.

   I don't have any zombie processes on my VM nor on my main machine, but let's pretend that I did have one with pid `5555`. I could do the following to kill it:

   ```
   pstree -p -s 5555
   ```

   That would display a tree-like structure containing the ancestors of the process. From there, we can identify the immediate parent of that process (let's say its 7777), and send it a `SIGCHLD` signal, which tells the parent process to execute the wait() system call and clean up its zombie children.

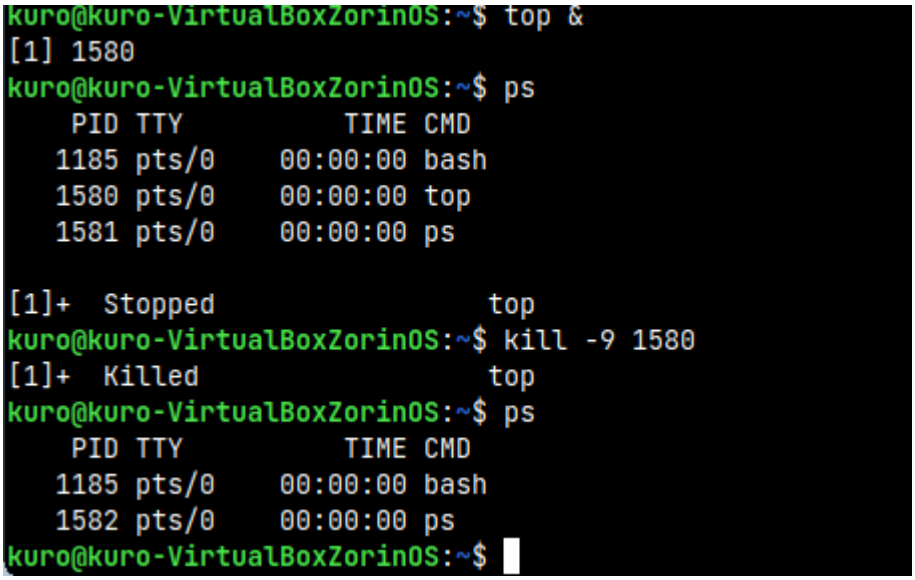
   ```
   kill -s SIGCHLD 7777
   ```

   If that doesn't get rid of the process, then we can try killing the parent process:

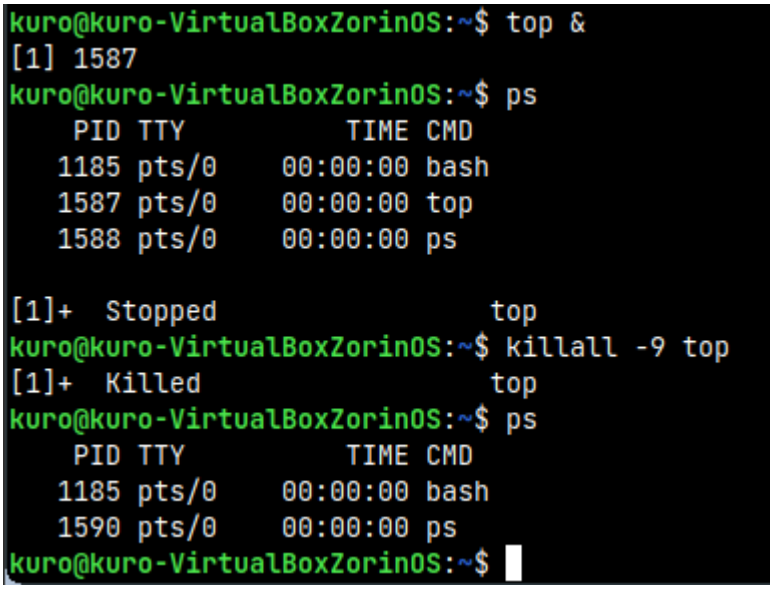
```
kill -9 7777
```

If that doesn't kill it, then a system reboot might be the last resort.

2.
   - `kill` will simply send a signal to a process based on its `PID`.



   - `killall` does the same thing, but by specifying the exact name of the process, instead of its PID. If more than one process runs with that name, all of them will be killed.



   - `pkill` does essentially the same thing with a few differences. One difference is that `killall` takes the exact name of process as the argument whereas `pkill` can take partial or complete name.

```
kuro@kuro-VirtualBoxZorinOS:~$ top &
[1] 1613
kuro@kuro-VirtualBoxZorinOS:~$ ps
    PID TTY          TIME CMD
   1185 pts/0    00:00:00 bash
   1613 pts/0    00:00:00 top
   1614 pts/0    00:00:00 ps

[1]+  Stopped                 top
kuro@kuro-VirtualBoxZorinOS:~$ pkill -9 top
[1]+  Killed                  top
kuro@kuro-VirtualBoxZorinOS:~$ ps
    PID TTY          TIME CMD
   1185 pts/0    00:00:00 bash
   1628 pts/0    00:00:00 ps
kuro@kuro-VirtualBoxZorinOS:~$
```

3. The `Tasks` and `%Cpu(s)` rows contain the following fields:

```
top - 14:18:09 up  2:28,  1 user,  load average: 0,18, 0,07, 0,02
Tasks: 155 total,   1 running, 154 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,0 us,  0,3 sy,  0,0 ni, 99,7 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
```

   - `Tasks`:
     - `total`: Total number of processes.
     - `running`: Processes that are either executing on the CPU or ready and waiting to be executed.
     - `sleeping`: Processes that are waiting for an even or an I/O operation to complete.
     - `stopped`: Processes that have been stopped by a job control signal or because they are being traced.
     - `zombie`: Processes whose execution is completed but still have an entry in the process table.
   - `%Cpu(s)`:
     - `us`: The time the CPU spends executing processes in userspace.
     - `sy`: The time the CPU spends executing processes in kernelspace.
     - `ni`: The time spent on executing processes with a manually set “nice”.
     - `id`: The time the CPU remains idle.
     - `wa`: The time the CPU spends waiting for I/O to complete.
     - `hi`: The time spent on handling hardware interrupts.
     - `si`: The time spent on handling software interrupts.
     - `st`: The amount of time lost due to the processor being busy on some VM.

4. Here's the script and a sample of its work:

```
Open   script.sh ~/Desktop   Save
1 #!/bin/bash
2 ps | awk '{if ($4 == "sleep") {system("kill -15 " $1); print "Got one and killed it!"} else {print "Not what
  we are looking for :("} }'
```

```
kuro@kuro-VirtualBoxZorinOS:~/Desktop$ ps
    PID TTY          TIME CMD
   1154 pts/0    00:00:00 bash
   1430 pts/0    00:00:00 ps
kuro@kuro-VirtualBoxZorinOS:~/Desktop$ bash -c "exec -a fun${RANDOM}process sleep infinity" &
[1] 1431
kuro@kuro-VirtualBoxZorinOS:~/Desktop$ bash -c "exec -a fun${RANDOM}process sleep infinity" &
[2] 1432
kuro@kuro-VirtualBoxZorinOS:~/Desktop$ bash -c "exec -a fun${RANDOM}process sleep infinity" &
[3] 1433
kuro@kuro-VirtualBoxZorinOS:~/Desktop$ ./script.sh
Not what we are looking for :(
Not what we are looking for :(
Got one and killed it!
Got one and killed it!
Got one and killed it!
Not what we are looking for :(
Not what we are looking for :(
Not what we are looking for :(
[1]   Terminated              bash -c "exec -a fun${RANDOM}process sleep infinity"
[2]-  Terminated              bash -c "exec -a fun${RANDOM}process sleep infinity"
[3]+  Terminated              bash -c "exec -a fun${RANDOM}process sleep infinity"
kuro@kuro-VirtualBoxZorinOS:~/Desktop$ ps
    PID TTY          TIME CMD
   1154 pts/0    00:00:00 bash
   1441 pts/0    00:00:00 ps
kuro@kuro-VirtualBoxZorinOS:~/Desktop$
```

5. Here's the script:

```
5.sh
#!/bin/bash
trap "echo Interrupt received" SIGUSR1

while :
do
    echo "Hello world!"
    sleep 10
done
```

I will demonstrate the usage in 2 terminals for clarity (Alternatively, I could have run the script in the background using `&`):

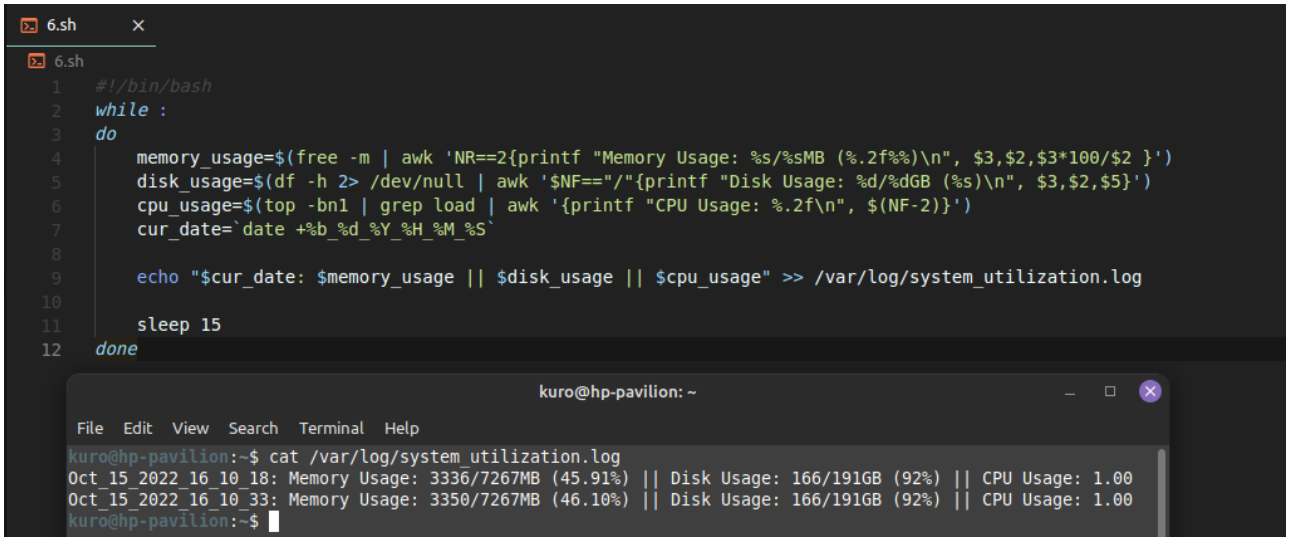
```
kuro@hp-pavilion: ~/zaker/sna/lab_7
File  Edit  View  Search  Terminal  Help
kuro@hp-pavilion:~/zaker/sna/lab_7$ ./5.sh
Hello world!
Hello world!
Hello world!
Hello world!
Interrupt received
Hello world!
Hello world!
Interrupt received
Hello world!
Killed
kuro@hp-pavilion:~/zaker/sna/lab_7$

kuro@hp-pavilion: ~
File  Edit  View  Search  Terminal  Help
kuro@hp-pavilion:~$ ps -a
    PID TTY          TIME CMD
  34123 pts/0    00:00:00 5.sh
  34366 pts/0    00:00:00 sleep
  34368 pts/1    00:00:00 ps
kuro@hp-pavilion:~$ kill -10 34123
kuro@hp-pavilion:~$ kill -10 34123
kuro@hp-pavilion:~$ kill -9 34123
kuro@hp-pavilion:~$
```

In the first terminal, we simply run the script, and it will start printing `Hello world!` every 10 seconds.

In the second terminal, we use `ps -a` to get the `PID` of the script process (or just use the name with `killall`), and then send `10` signal to it (which stands for `SIGUSR1`), we can see the output `Interrupt received` in the second terminal which was caught by `trap`. We then send `9` signal to stop the script.

6. Here's the script and a sample of the produced log:



End of Exercises