System Programming II- Lab Exercise

Process Management

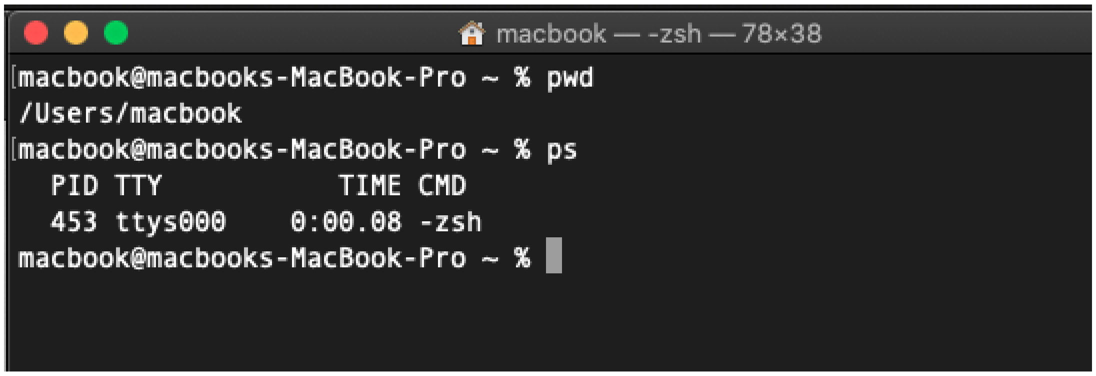
Name: Akech Dau Atem

Reg: SCT211-0535/2022

Activity 1: Monitoring Linux Processes with ps command

Perform the following steps:

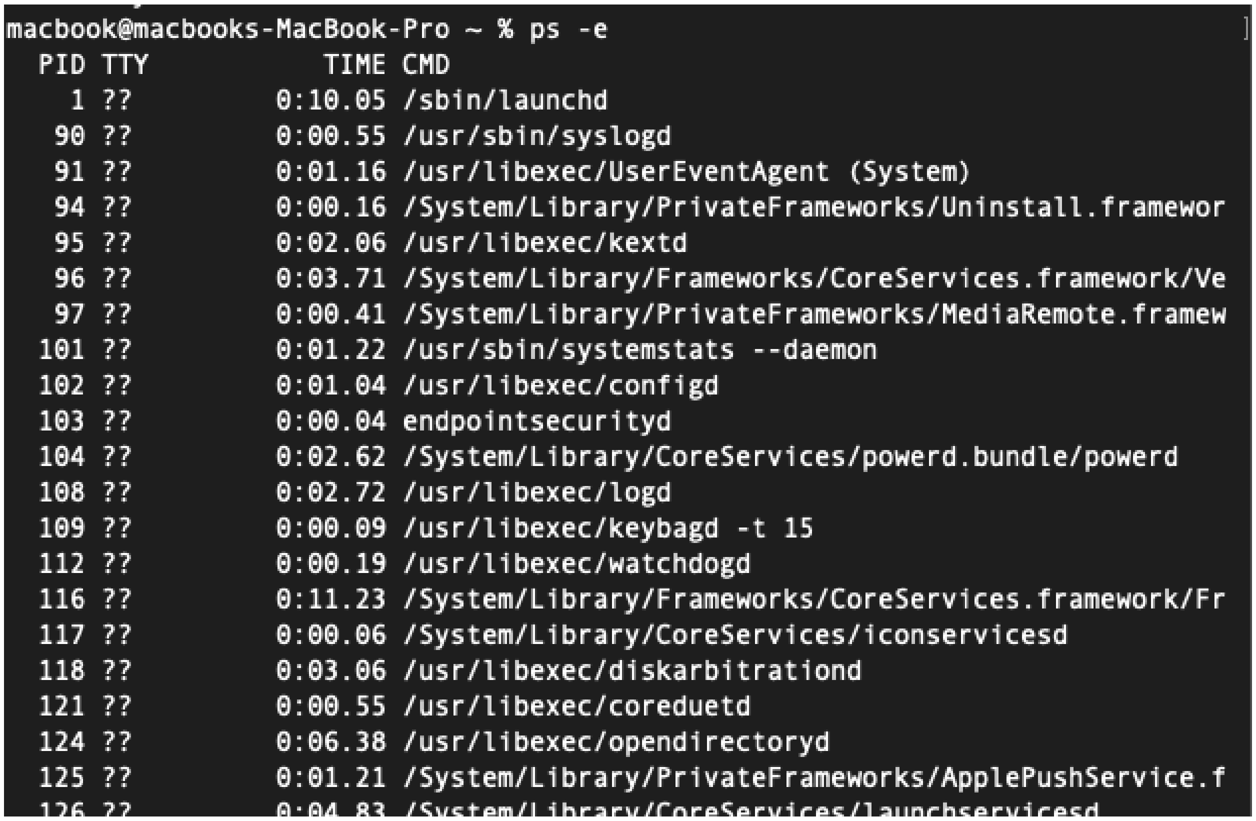
1. Login to your Machine with Linux installation
2. Issue a Linux command to confirm that you are located in your home directory.
3. The ps_ command provides a list of processes that are running, or at least that were running at the time the command was called. Run the command ps in your terminal
4. Take a screen shot of what you see and paste here



```
macbook@macbooks-MacBook-Pro ~ % pwd
/Users/macbook
macbook@macbooks-MacBook-Pro ~ % ps
  PID TTY           TIME CMD
  453 ttys000    0:00.08 -zsh
macbook@macbooks-MacBook-Pro ~ % 
```

5. How many processes are currently running? What information is displayed for each process? Paste the screen here
   - There are X processes running
   - The number of processes is small because the ps command by default shows only those associated with the current terminal session. As shown in the above screenshot.

6. Use the ps command with the '-e' option to display information about all processes in the system. Run the command ps -e



```
macbook@macbooks-MacBook-Pro ~ % ps -e
  PID TTY           TIME CMD
    1 ??         0:10.05 /sbin/launchd
   90 ??         0:00.55 /usr/sbin/syslogd
   91 ??         0:01.16 /usr/libexec/UserEventAgent (System)
   94 ??         0:00.16 /System/Library/PrivateFrameworks/Uninstall.framewor
   95 ??         0:02.06 /usr/libexec/kextd
   96 ??         0:03.71 /System/Library/Frameworks/CoreServices.framework/Ve
   97 ??         0:00.41 /System/Library/PrivateFrameworks/MediaRemote.framew
  101 ??         0:01.22 /usr/sbin/systemstats --daemon
  102 ??         0:01.04 /usr/libexec/configd
  103 ??         0:00.04 endpointsecurityd
  104 ??         0:02.62 /System/Library/CoreServices/powerd.bundle/powerd
  108 ??         0:02.72 /usr/libexec/logd
  109 ??         0:00.09 /usr/libexec/keybagd -t 15
  112 ??         0:00.19 /usr/libexec/watchdogd
  116 ??         0:11.23 /System/Library/Frameworks/CoreServices.framework/Fr
  117 ??         0:00.06 /System/Library/CoreServices/iconservicesd
  118 ??         0:03.06 /usr/libexec/diskarbitrationd
  121 ??         0:00.55 /usr/libexec/coreduetd
  124 ??         0:06.38 /usr/libexec/opendirectoryd
  125 ??         0:01.21 /System/Library/PrivateFrameworks/ApplePushService.f
  126 ??         0:04.83 /System/Library/CoreServices/launchservicesd
```

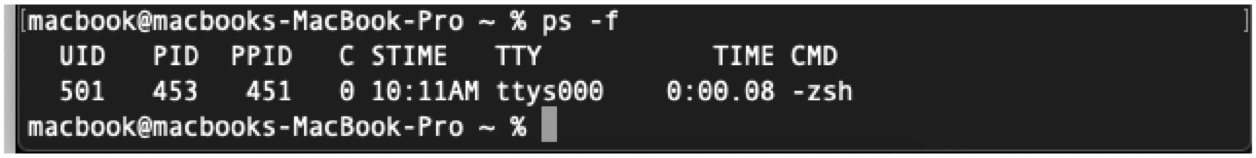
7. Analyze the output and identify the running processes on your system. Note the PID, TTY, and CMD columns. What do these columns mean?

   This command gives a broader view of the system by listing all processes running across the system. I noticed several system processes like systemd, dbus-daemon, etc., which are necessary for the OS to function.

Columns:

- PID: Identifies each process.
- TTY: Displays ? for processes not associated with a terminal (e.g., background services).
- CMD: Command used to start the process.

8. Use the 'ps' command with the '-f' option to display a full-format listing of the processes. Run the command ps - f



```
[macbook@macbooks-MacBook-Pro ~ % ps -f
  UID   PID  PPID   C STIME   TTY           TIME CMD
  501   453   451   0 10:11AM ttys000    0:00.08 -zsh
macbook@macbooks-MacBook-Pro ~ %
```

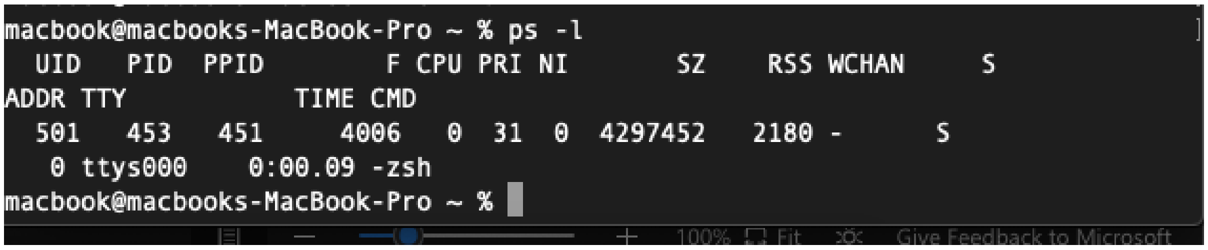
9. Examine the output, which provides detailed information about each process, including UID, PID, PPID, CPU%, MEM%, START, and CMD

   The output from ps -f provides detailed information about processes, including:

   - UID: The user who owns the process.
   - PID: Unique process identifier.
   - PPID: Parent process ID, indicating which process started the current one.
   - CPU%: Percentage of CPU usage.
   - MEM%: Percentage of memory usage.
   - START: When the process started.
   - CMD: The command that initiated the process.

   This information helps monitor system resources and understand process relationships.

10. Use the 'ps' command with the '-l' option to display a long listing format of processes. Execute the following command ps - l



```
macbook@macbooks-MacBook-Pro ~ % ps -l
  UID   PID  PPID        F CPU PRI NI       SZ    RSS WCHAN     S
ADDR TTY           TIME CMD
  501   453   451     4006   0  31  0  4297452   2180 -      S
   0 ttys000    0:00.09 -zsh
macbook@macbooks-MacBook-Pro ~ %
```

11. Analyze the output and observe the columns displayed, including F, S, UID, PID, PPID, PRI, NI, ADDR, SZ, RSS, WCHAN, STAT, TTY, TIME, and CMD.

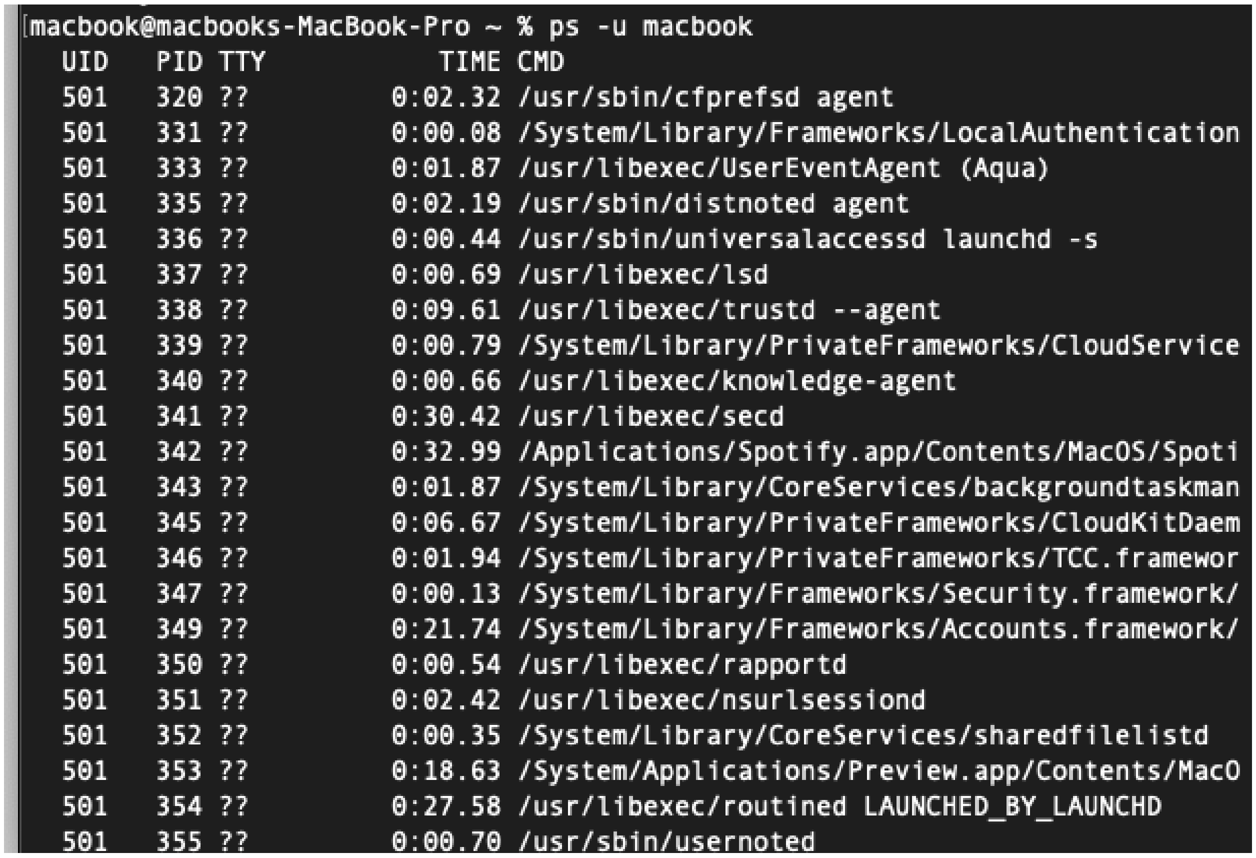
   - **F**: Flags that represent the state of the process (e.g., superuser privileges).
   - **S**: Process state (R for running, S for sleeping, Z for zombie).
   - **UID**: User ID of the owner.
   - **PID**: Process ID.
   - **PPID**: Parent Process ID, indicating the process that started it.
   - **PRI**: Priority of the process (lower value = higher priority).
   - **NI**: Nice value, which adjusts the process priority (negative = higher priority).
   - **ADDR**: Memory address of the process (rarely used).
   - **SZ**: Size of the process in memory (in pages).
   - **RSS**: Resident Set Size, indicating how much memory is currently used by the process.
   - **WCHAN**: The name of the kernel function the process is currently waiting on (if applicable).

- **STAT**: Process status (combination of state and other flags).
- **TTY**: Terminal associated with the process (if any).
- **TIME**: Total CPU time the process has consumed.
- **CMD**: The command that started the process.

Analysis:

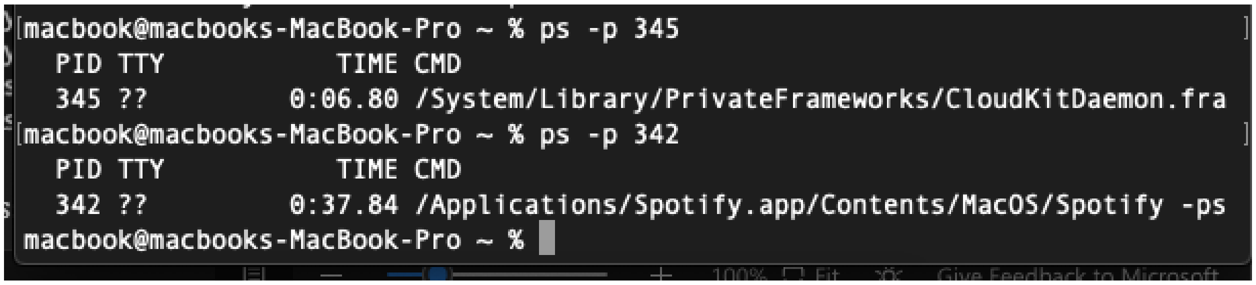
- F and S help determine the process's status and privilege level.
- PRI and NI give insight into how the process is prioritized for CPU scheduling.
- RSS and SZ indicate memory usage.

12. Use the '-u' option followed by a username to display processes owned by that user.



```
[macbook@macbooks-MacBook-Pro ~ % ps -u macbook
  UID   PID TTY           TIME CMD
  501   320 ??         0:02.32 /usr/sbin/cfprefsd agent
  501   331 ??         0:00.08 /System/Library/Frameworks/LocalAuthentication
  501   333 ??         0:01.87 /usr/libexec/UserEventAgent (Aqua)
  501   335 ??         0:02.19 /usr/sbin/distnoted agent
  501   336 ??         0:00.44 /usr/sbin/universalaccessd launchd -s
  501   337 ??         0:00.69 /usr/libexec/lsd
  501   338 ??         0:09.61 /usr/libexec/trustd --agent
  501   339 ??         0:00.79 /System/Library/PrivateFrameworks/CloudService
  501   340 ??         0:00.66 /usr/libexec/knowledge-agent
  501   341 ??         0:30.42 /usr/libexec/secd
  501   342 ??         0:32.99 /Applications/Spotify.app/Contents/MacOS/Spoti
  501   343 ??         0:01.87 /System/Library/CoreServices/backgroundtaskman
  501   345 ??         0:06.67 /System/Library/PrivateFrameworks/CloudKitDaem
  501   346 ??         0:01.94 /System/Library/PrivateFrameworks/TCC.framewor
  501   347 ??         0:00.13 /System/Library/Frameworks/Security.framework/
  501   349 ??         0:21.74 /System/Library/Frameworks/Accounts.framework/
  501   350 ??         0:00.54 /usr/libexec/rapportd
  501   351 ??         0:02.42 /usr/libexec/nsurlsessiond
  501   352 ??         0:00.35 /System/Library/CoreServices/sharedfilelistd
  501   353 ??         0:18.63 /System/Applications/Preview.app/Contents/MacO
  501   354 ??         0:27.58 /usr/libexec/routined LAUNCHED_BY_LAUNCHD
  501   355 ??         0:00.70 /usr/sbin/usernoted
```

13. Use the '-p ' option followed by a process ID (PID) to display information about a specific process.

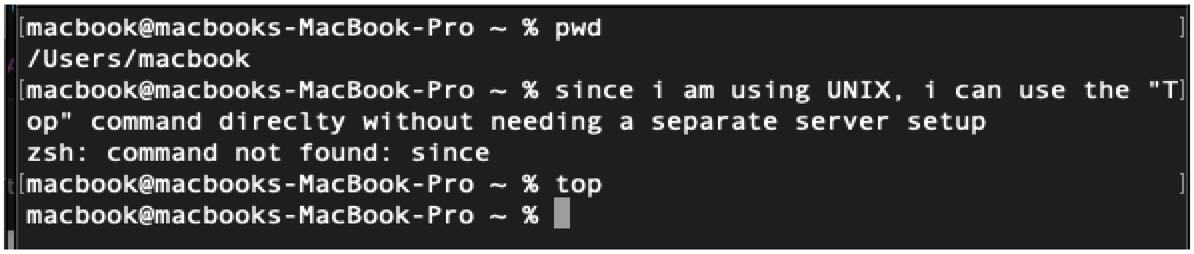


```
macbook@macbooks-MacBook-Pro ~ % ps -p 345
  PID TTY           TIME CMD
  345 ??         0:06.80 /System/Library/PrivateFrameworks/CloudKitDaemon.fra
macbook@macbooks-MacBook-Pro ~ % ps -p 342
  PID TTY           TIME CMD
  342 ??         0:37.84 /Applications/Spotify.app/Contents/MacOS/Spotify -ps
macbook@macbooks-MacBook-Pro ~ %
```

Activity 2: Monitoring Linux Processes with top command

The **top** command is a powerful tool in Linux used to monitor and manage system resources in real-time. It provides a dynamic view of CPU usage, memory utilization, running processes, and other essential system metrics.

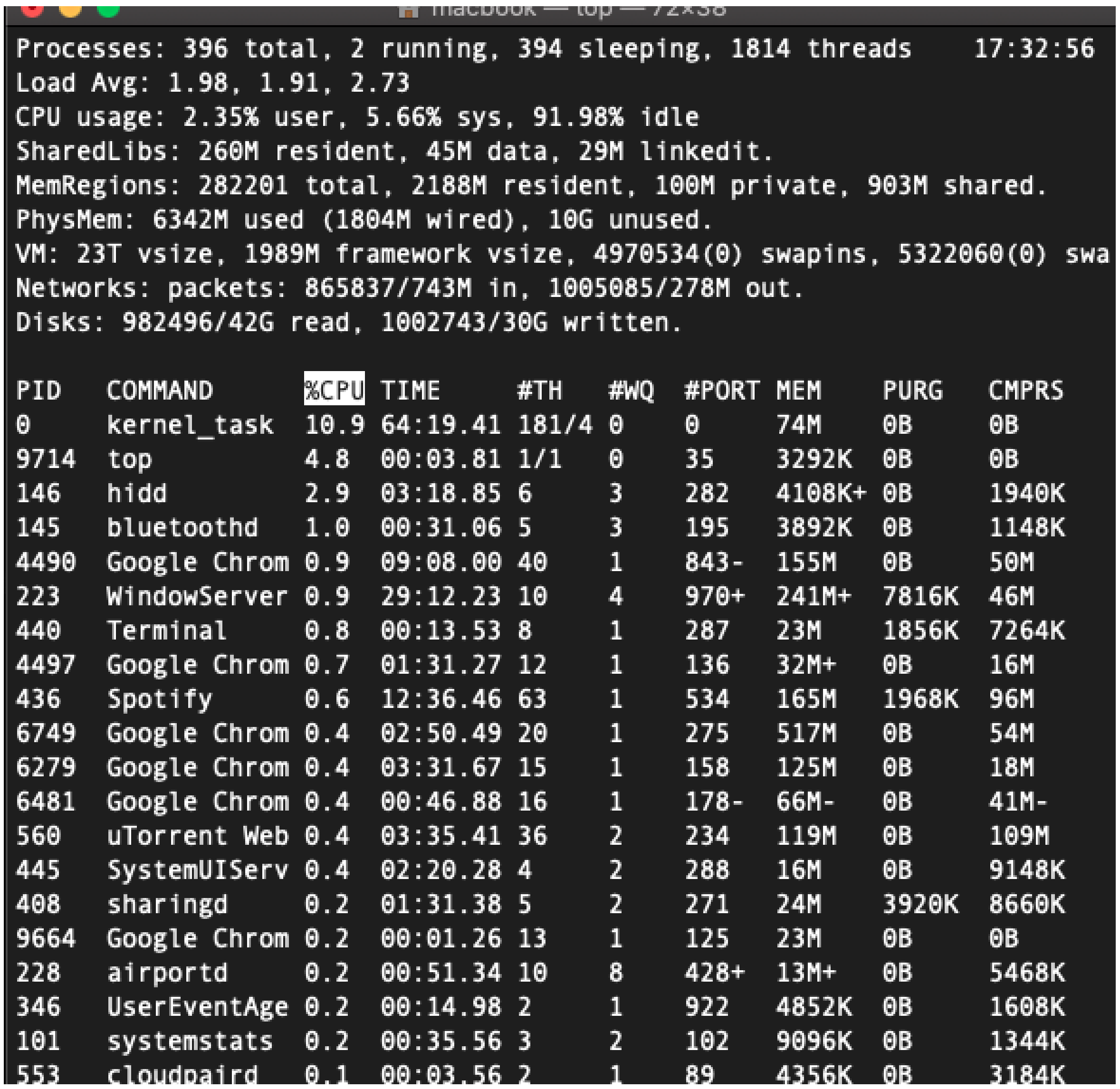
In this activity, sign in to matrix, and experiment with this command to understand resource usage. You might want to refer to the top(1) man page.



```
macbook@macbooks-MacBook-Pro ~ % pwd
/Users/macbook
macbook@macbooks-MacBook-Pro ~ % since i am using UNIX, i can use the "T
op" command direclty without needing a separate server setup
zsh: command not found: since
macbook@macbooks-MacBook-Pro ~ % top
macbook@macbooks-MacBook-Pro ~ %
```

1. Run the command **top**in your terminal. What output do you observe, paste the

screen shot here



```
Processes: 396 total, 2 running, 394 sleeping, 1814 threads    17:32:56
Load Avg: 1.98, 1.91, 2.73
CPU usage: 2.35% user, 5.66% sys, 91.98% idle
SharedLibs: 260M resident, 45M data, 29M linkedit.
MemRegions: 282201 total, 2188M resident, 100M private, 903M shared.
PhysMem: 6342M used (1804M wired), 10G unused.
VM: 23T vsize, 1989M framework vsize, 4970534(0) swapins, 5322060(0) swa
Networks: packets: 865837/743M in, 1005085/278M out.
Disks: 982496/42G read, 1002743/30G written.

PID   COMMAND      %CPU TIME     #TH   #WQ  #PORT MEM    PURG   CMPRS
0     kernel_task  10.9 64:19.41 181/4 0    0     74M    0B     0B
9714  top          4.8  00:03.81 1/1   0    35    3292K  0B     0B
146   hidd         2.9  03:18.85 6     3    282   4108K+ 0B     1940K
145   bluetoothd   1.0  00:31.06 5     3    195   3892K  0B     1148K
4490  Google Chrom 0.9  09:08.00 40    1    843-  155M   0B     50M
223   WindowServer 0.9  29:12.23 10    4    970+  241M+  7816K  46M
440   Terminal     0.8  00:13.53 8     1    287   23M    1856K  7264K
4497  Google Chrom 0.7  01:31.27 12    1    136   32M+   0B     16M
436   Spotify      0.6  12:36.46 63    1    534   165M   1968K  96M
6749  Google Chrom 0.4  02:50.49 20    1    275   517M   0B     54M
6279  Google Chrom 0.4  03:31.67 15    1    158   125M   0B     18M
6481  Google Chrom 0.4  00:46.88 16    1    178-  66M-   0B     41M-
560   uTorrent Web 0.4  03:35.41 36    2    234   119M   0B     109M
445   SystemUIServ 0.4  02:20.28 4     2    288   16M    0B     9148K
408   sharingd     0.2  01:31.38 5     2    271   24M    3920K  8660K
9664  Google Chrom 0.2  00:01.26 13    1    125   23M    0B     0B
228   airportd     0.2  00:51.34 10    8    428+  13M+   0B     5468K
346   UserEventAge 0.2  00:14.98 2     1    922   4852K  0B     1608K
101   systemstats  0.2  00:35.56 3     2    102   9096K  0B     1344K
553   cloudpaird   0.1  00:03.56 2     1    89    4356K  0B     3184K
```

2. Once the top command is running, you'll see a continuously updated display with various sections and columns.

3. Explain what information the following columns give.
a. PR
b. NI
c. VIRT
d. RES
e. %CPU

f. %MEM
g. TIME+

1. **PR (Priority)**: Shows the scheduling priority of the process. Lower values indicate higher priority.
2. **NI (Nice Value)**: Represents the process's "niceness" level, which affects its priority. Lower values give the process more CPU time.
3. **VIRT (Virtual Memory)**: Displays the total virtual memory used by the process, including both physical memory and swap space.
4. **RES (Resident Memory)**: Shows the actual physical memory (RAM) used by the process.
5. **%CPU (CPU Usage)**: Indicates the percentage of the CPU's capacity that the process is using.
6. **%MEM (Memory Usage)**: Shows the percentage of the system's physical memory that the process is consuming.
7. **TIME+ (CPU Time)**: Displays the total CPU time the process has used since it started, in minutes and seconds.

4. The top command provides interactive features to customize the display and perform actions. Press 'P' to sort processes by CPU usage, 'M' to sort by memory usage, and 'N' to sort by PID.

5. To exit the top command, simply press 'q'. This will close the top display and return you to the terminal prompt.

Activity 3: Sending signals to processes

In Linux processes, system admins can send signals to communicate with processes and request specific actions. Signals are software interrupts delivered to a process by the operating system or another process. Signals allow processes to respond to various events, such as the termination of another process, user input, or changes in system conditions.

Signals are identified by unique numbers, known as signal numbers. Each signal number corresponds to a specific event or action.

Each signal has a default action associated with it, which determines what the process does when it receives that signal. Common default actions include termination, stopping, or ignoring the signal.Common Signals: Linux systems have a set of standard signals defined, each with its own

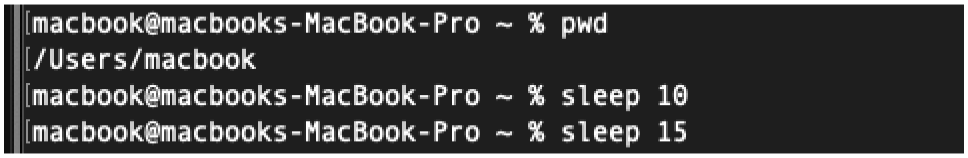
signal number. Some commonly used signals include:

- SIGTERM (Signal 15): This is the default signal sent by the kill command to request a process to terminate gracefully.
- SIGKILL (Signal 9): This signal immediately terminates a process. It cannot be caught or ignored.
- SIGSTOP (Signal 19): This signal pauses a process, suspending its execution until a
- SIGCONT signal is received.
- SIGCONT (Signal 18): This signal resumes the execution of a process that was previously stopped by a SIGSTOP signal.
- SIGHUP (Signal 1): This signal is typically sent to inform a process that the controlling terminal has been disconnected.

Signals can be sent to processes using the **kill command.**

Login to your system in Linux

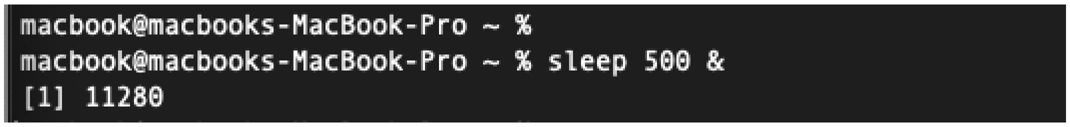
1. Issue the following command and observe that you must wait for the process (command) to finish after 10 seconds: **sleep 10**



```
[macbook@macbooks-MacBook-Pro ~ % pwd
[/Users/macbook
[macbook@macbooks-MacBook-Pro ~ % sleep 10
[macbook@macbooks-MacBook-Pro ~ % sleep 15
```

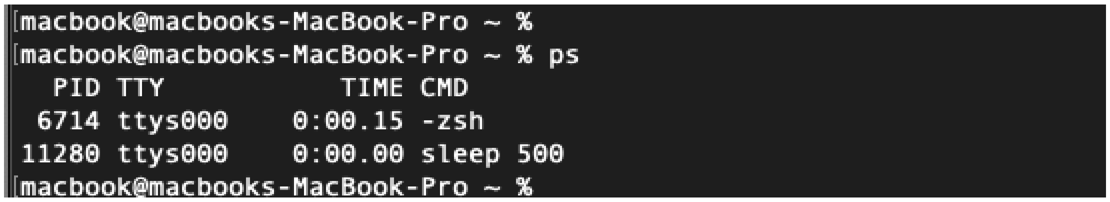
2. Issue the following command: **sleep 500 &**

The "sleep" command in Linux is a utility that allows you to pause the execution of a script or command for a specified amount of time. We will be using this command to simulate the behavior of a "long-running" process. This process will run in the background for **500 seconds**, and is not forcing the user to **wait** until this process finishes. A process that is **running in the terminal** is referred to as a **foreground process.** A process that is running in the background does not block the user.



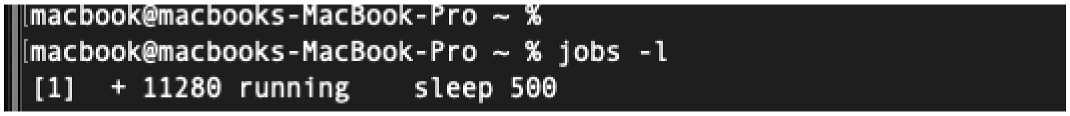
```
macbook@macbooks-MacBook-Pro ~ %
macbook@macbooks-MacBook-Pro ~ % sleep 500 &
[1] 11280
```

3. Run the command: **ps**



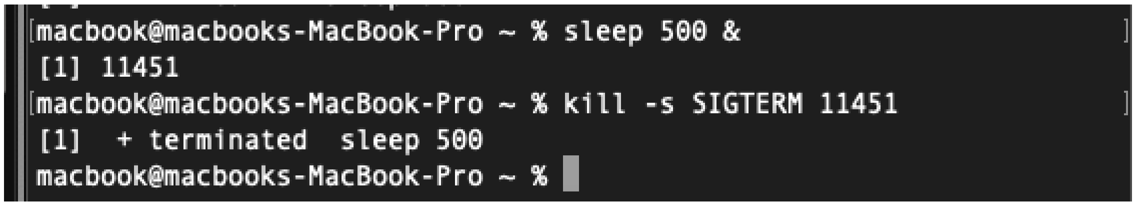
```
macbook@macbooks-MacBook-Pro ~ %
macbook@macbooks-MacBook-Pro ~ % ps
  PID TTY           TIME CMD
 6714 ttys000    0:00.15 -zsh
11280 ttys000    0:00.00 sleep 500
macbook@macbooks-MacBook-Pro ~ %
```

4. Note the process id of the background sleep command.
11280
5. Run the command: jobs -l



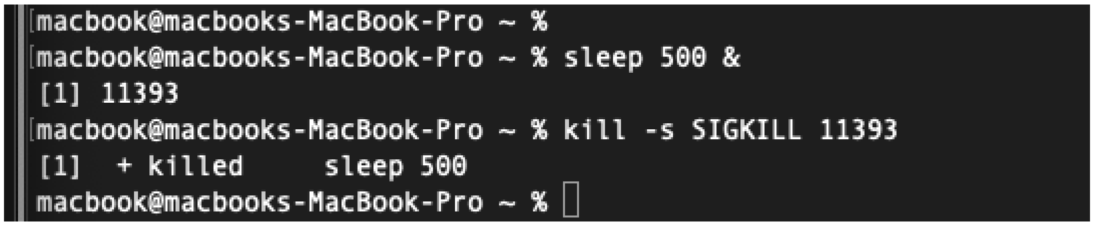
```
macbook@macbooks-MacBook-Pro ~ %
macbook@macbooks-MacBook-Pro ~ % jobs -l
[1]  + 11280 running    sleep 500
```

6. Note that it also shows the same process id for the sleep command.
11280
7. Run the command: **kill PID** (replace PID with process id)
By default, the kill command sends the SIGTERM signal (signal number 15) to the process, requesting it to terminate gracefully. However, you can specify a different signal using the -s option followed by the signal number or signal name. What output you see? Paste a screenshot of the output below.



```
macbook@macbooks-MacBook-Pro ~ % sleep 500 &
[1] 11451
macbook@macbooks-MacBook-Pro ~ % kill -s SIGTERM 11451
[1]  + terminated  sleep 500
macbook@macbooks-MacBook-Pro ~ %
```

Run the command "sleep 500 &" another time and this time send the SIGKILL signal to this process. What output you see? Paste a screenshot of the output below.



```
macbook@macbooks-MacBook-Pro ~ %
macbook@macbooks-MacBook-Pro ~ % sleep 500 &
[1] 11393
macbook@macbooks-MacBook-Pro ~ % kill -s SIGKILL 11393
[1]  + killed     sleep 500
macbook@macbooks-MacBook-Pro ~ %
```

8. What difference you noticed in SIGTERM and SIGKILL signals?

- **SIGTERM** allows a process to terminate gracefully, giving it time to clean up resources.

  [1]+ terminated   sleep 500

- **SIGKILL** forcefully kills the process immediately
  [1]+ Killed   sleep 500

Activity 4: Foreground and background processes

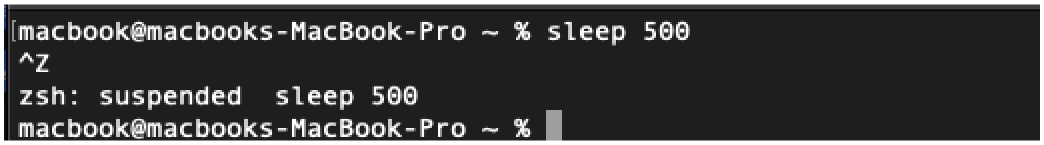
1. Again, use the following command:

   sleep 500

The Unix/Linux system is designed to allow users to send **preemptive signals** to manage those processes.

2. Press the following key combination to interrupt the process running on the terminal: ctrl-z.   This sends a SIGSTOP signal to the process.

3. You should see output similar to what is displayed below:

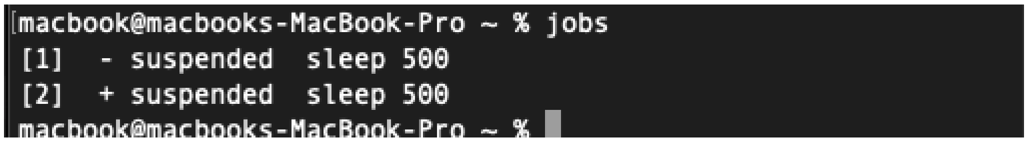


```
macbook@macbooks-MacBook-Pro ~ % sleep 500
^Z
zsh: suspended  sleep 500
macbook@macbooks-MacBook-Pro ~ %
```

4. This indicates that this process has been placed into the **background**. This is useful in order to "**free-up**" the terminal to run other Linux commands

5. Issue the following Linux command: jobs

You should see the following output:



```
macbook@macbooks-MacBook-Pro ~ % jobs
[1]  - suspended  sleep 500
[2]  + suspended  sleep 500
macbook@macbooks-MacBook-Pro ~ %
```

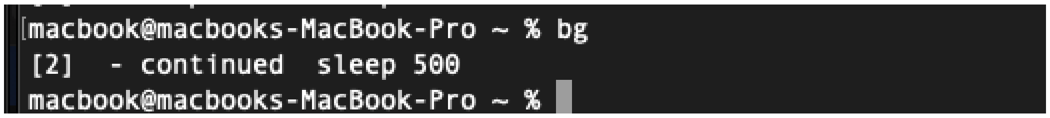
This display indicates that this process (that is now in the background) has **stopped**. In other words, the *sleep* command is NOT counting-down to zero to terminate.

6. The plus sign "+" indicates the most recent process placed into the background.

7. Sometimes you would like to run the process you stopped in the background. You

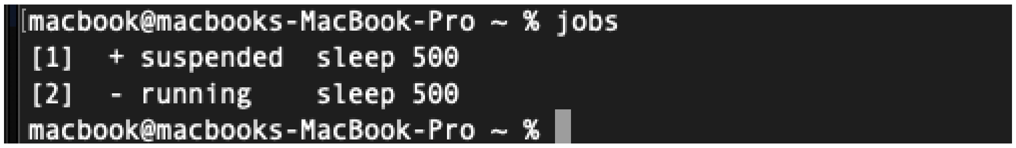
can use bg command without arguments to run in background the most recent process that was stopped.

8. Run the command: bg



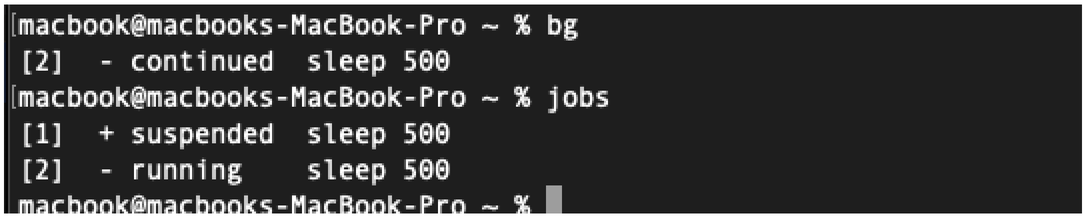
```
macbook@macbooks-MacBook-Pro ~ % bg
[2]  - continued  sleep 500
macbook@macbooks-MacBook-Pro ~ %
```

9. Issue the command: jobs



```
macbook@macbooks-MacBook-Pro ~ % jobs
[1]  + suspended  sleep 500
[2]  - running    sleep 500
macbook@macbooks-MacBook-Pro ~ %
```

10. You should see the following output similar to what was displayed above

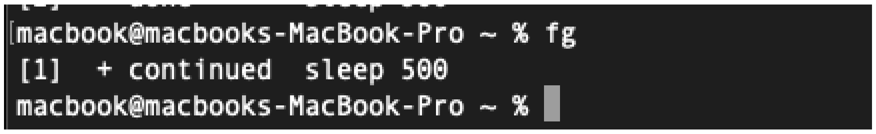


```
macbook@macbooks-MacBook-Pro ~ % bg
[2]  - continued  sleep 500
macbook@macbooks-MacBook-Pro ~ % jobs
[1]  + suspended  sleep 500
[2]  - running    sleep 500
macbook@macbooks-MacBook-Pro ~ %
```

11. The & sign indicates that the process is now running in the background.

12. You can also bring this process to foreground using fg command.

13. Issue the command fg. This will make the sleep process run in foreground.

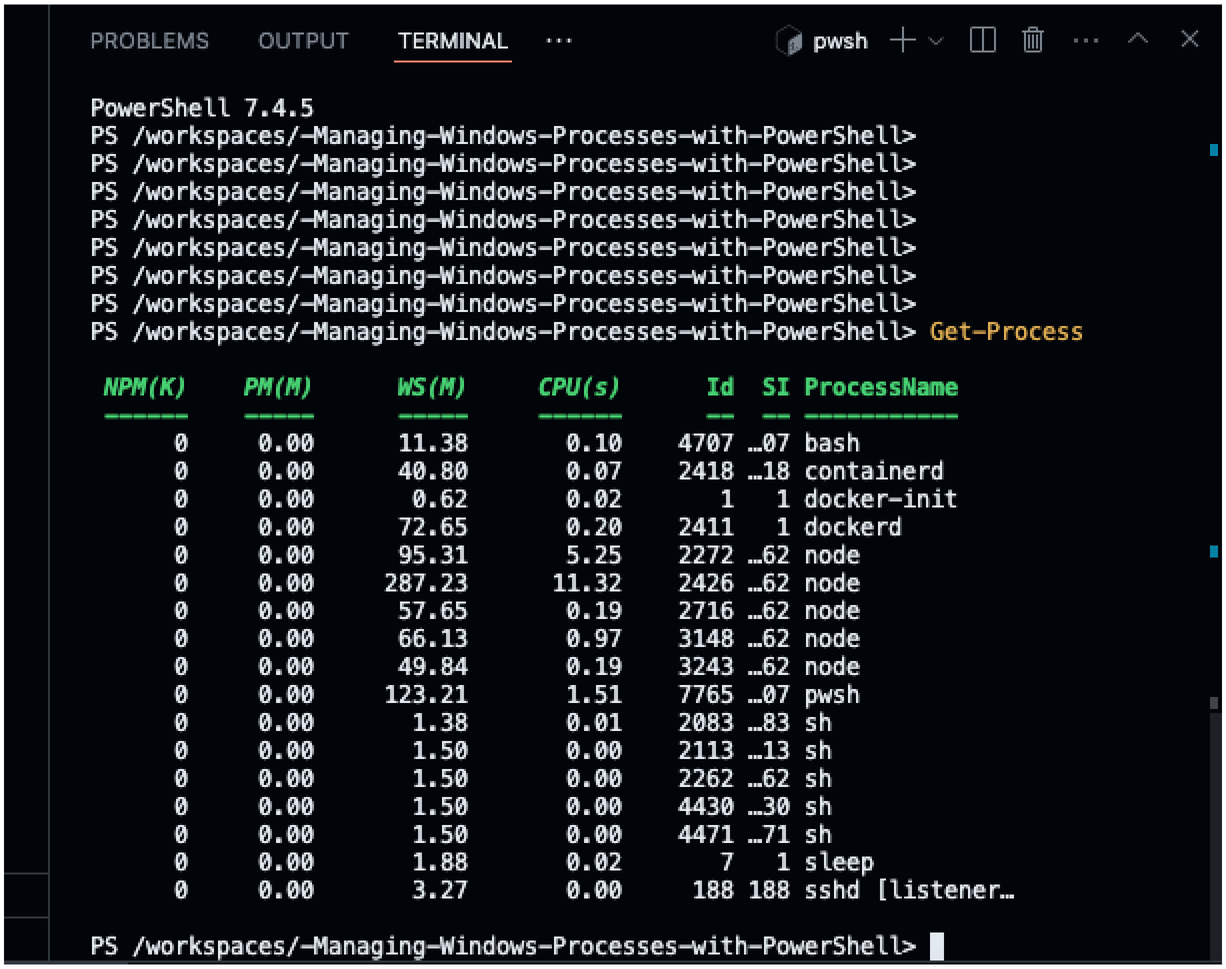


```
macbook@macbooks-MacBook-Pro ~ % fg
[1]  + continued  sleep 500
macbook@macbooks-MacBook-Pro ~ %
```

Activity 5: Managing Windows Processes with PowerShell

Mostly Task Manager application is used for managing processes on Windows. However, Windows PowerShell does provide some commands for process management. The main command used to get information about process is called '**Get-Process**'. In the following tasks use this command in Windows PowerShell to get information about the process.

1. Run the **Get-Process** command in PowerShell and explain the output



```
PowerShell 7.4.5
PS /workspaces/-Managing-Windows-Processes-with-PowerShell>
PS /workspaces/-Managing-Windows-Processes-with-PowerShell>
PS /workspaces/-Managing-Windows-Processes-with-PowerShell>
PS /workspaces/-Managing-Windows-Processes-with-PowerShell>
PS /workspaces/-Managing-Windows-Processes-with-PowerShell>
PS /workspaces/-Managing-Windows-Processes-with-PowerShell>
PS /workspaces/-Managing-Windows-Processes-with-PowerShell>
PS /workspaces/-Managing-Windows-Processes-with-PowerShell> Get-Process

 NPM(K)    PM(M)      WS(M)     CPU(s)      Id  SI ProcessName
 ------    -----      -----     ------      --  -- -----------
      0     0.00      11.38       0.10    4707 …07 bash
      0     0.00      40.80       0.07    2418 …18 containerd
      0     0.00       0.62       0.02       1   1 docker-init
      0     0.00      72.65       0.20    2411   1 dockerd
      0     0.00      95.31       5.25    2272 …62 node
      0     0.00     287.23      11.32    2426 …62 node
      0     0.00      57.65       0.19    2716 …62 node
      0     0.00      66.13       0.97    3148 …62 node
      0     0.00      49.84       0.19    3243 …62 node
      0     0.00     123.21       1.51    7765 …07 pwsh
      0     0.00       1.38       0.01    2083 …83 sh
      0     0.00       1.50       0.00    2113 …13 sh
      0     0.00       1.50       0.00    2262 …62 sh
      0     0.00       1.50       0.00    4430 …30 sh
      0     0.00       1.50       0.00    4471 …71 sh
      0     0.00       1.88       0.02       7   1 sleep
      0     0.00       3.27       0.00     188 188 sshd [listener…

PS /workspaces/-Managing-Windows-Processes-with-PowerShell>
```

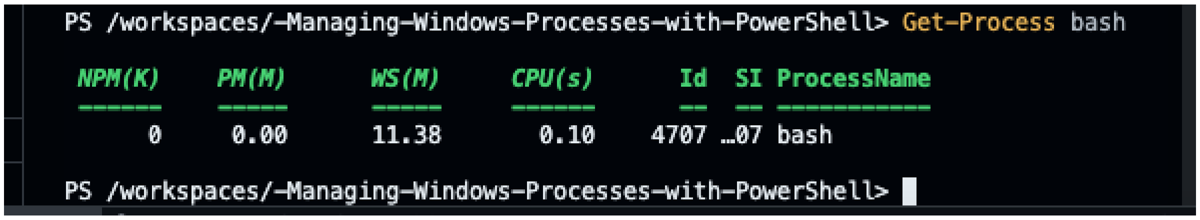
This command list of all processes running in my Codespace

2. Explain the meaning of column headers of the information output by this command

- **NPM(K)**: Non-paged memory used by the process (in kilobytes) that remains in RAM.
- **PM(M)**: Paged memory used by the process (in megabytes) that can be written to disk when not in use.
- **WS(M)**: Working Set size (in megabytes), indicating the physical memory currently being used by the process.
- **CPU(s)**: Total CPU time used by the process (in seconds).
- **Id**: Unique Process ID (PID) assigned to the process.
- **SI**: Session ID, indicating the session in which the process is running.
- **ProcessName**: The name of the process or executable file that started the process.

3. To get information about specific process you can use the syntax **Get-Process <process-name>**. For example, to get information about firefox process you can run command **Get-Process firefox**. Run this command to get information about a

process of your choosing and take and save a screenshot for later submission.



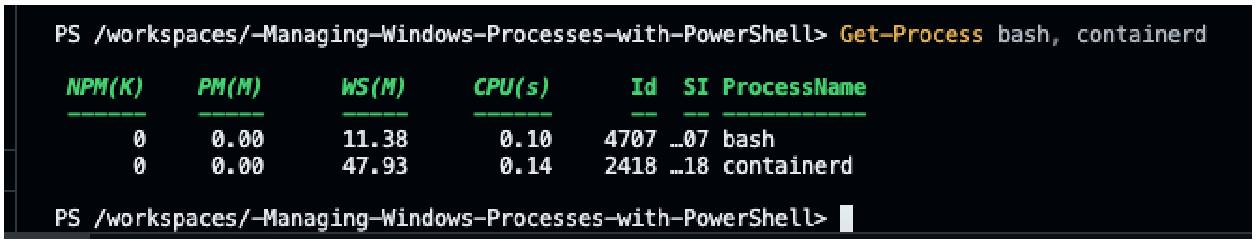
```
PS /workspaces/-Managing-Windows-Processes-with-PowerShell> Get-Process bash

 NPM(K)    PM(M)      WS(M)     CPU(s)      Id  SI ProcessName
 ------    -----      -----     ------      --  -- -----------
      0     0.00      11.38       0.10    4707 …07 bash

PS /workspaces/-Managing-Windows-Processes-with-PowerShell>
```

4. Using this same command describe with example how you can get information about multiple processes

To get information about multiple processes in PowerShell using the Get-Process command, you can specify multiple process names separated by commas



```
PS /workspaces/-Managing-Windows-Processes-with-PowerShell> Get-Process bash, containerd

 NPM(K)    PM(M)      WS(M)     CPU(s)      Id  SI ProcessName
 ------    -----      -----     ------      --  -- -----------
      0     0.00      11.38       0.10    4707 …07 bash
      0     0.00      47.93       0.14    2418 …18 containerd

PS /workspaces/-Managing-Windows-Processes-with-PowerShell>
```

5. To stop a process, you can use **Stop-Process** command with syntax **Stop- Process <process-name>**. In this task use this command to stop some process and show the screenshot below

The comment Stop-Process -Name bash-Force end the session on codespace

6. There are two other commands of this class to manage processes. These commands are **Wait-Process** and **Debug-Process**. Search and read about these commands and provide examples.

Wait-Process: This command pauses execution until a specified process has stopped running. Eg Wait-Process -Name notepad

Debug-Process: The Debug-Process command attaches a debugger to the specified process, allowing you to troubleshoot and analyze its behavior. Eg Debug-Process -Id 1234

Further Practice Questions.

Answer the following questions based on your knowledge of process management in

Linux. You do not need to submit these answers – these questions are to reinforce your understanding of the material.

1. What is a process in Linux? Answer:
A process is an instance of a running program, which includes the program code, its current activity, and allocated resources.

2. Name three different states a process can be in, and briefly describe each state.
a) State 1: Description: **Running**: The process is actively executing on the CPU.
b) State 2: Description: **Sleeping**: The process is idle, waiting for an event (like I/O) to occur.
c) State 3: Description: **Zombie**: The process has completed execution but still has an entry in the process table to report its exit status.

3. Which command is used to list processes in Linux? Provide an example of its usage.
Command: ps

Example: ps –aux lists all running processes with detailed information.

4. Explain the meaning of the following columns displayed by the ps command:

a) PID: Process ID, a unique identifier for each running process.
b) CPU%: Percentage of CPU resources used by the process.

c) MEM%: Percentage of physical memory used by the process.

5. How can you terminate a process in Linux? Describe two different methods.
1: Method - kill <PID> to send a termination signal to a specific process.
2: Method - Use killall <process-name> to terminate all instances of a process by name.

6. What is the purpose of the top command in Linux? How can you sort processes using top?
Purpose of top: To provide a real-time view of system processes and resource usage.
Sorting processes in top: Press P to sort by CPU usage or M to sort by memory usage.

7. Why is it important to exercise caution when terminating processes in Linux?

Explain briefly.

Terminating processes can lead to data loss, corruption, or system instability, especially if critical or system processes are stopped.

8. Briefly explain the difference between the kill and killall commands in Linux.
kill sends a signal to a specific process by its PID, while *killall* sends a signal to all processes with a given name.

9. True or False: Terminating a process with SIGKILL allows it to perform cleanup operations before termination. Answer: False

10. Name two signals that can be sent to a process using the kill command, and briefly describe their effects.

Signal 1: Effect: **SIGTERM -** Politely requests a process to terminate, allowing cleanup
Signal 2: Effect: SIGKILL - Forcefully terminates a process without allowing cleanup.