OS LAB ASSIGNMENT - 2

Done By	Roll Number
N Sree Dhyuti	CED19I027

---

(1) Implement FCFS (First Come First Serve) Algorithm.

CODE:

```
// N Sree Dhyuti
// CED19I027
// Lab 2 : Q1
// Inclusion of required libraries
#include <stdio.h>

// Function to sort the elements in 3 arrays consecutively : Bubble Sort
// Here arr1 is the array with respect to which the sorting is happening
// Eg : sort (AT, BT, PID, 1, num -1) means we are sorting the 3 arrays wrt Arrival Time and in the array range 1 to num-1
void sort(float* arr1, float* arr2, float* arr3, int start, int end)
{
        int a, b;
        for(a = start; a < end - 1; a++)
        {
                for(b = a; b < end; b++)
                {
                        if(arr1[a] > arr1[b])
                        {
                                float temp;
                                // Swap
                                temp = arr1[b]; arr1[b] = arr1[a]; arr1[a] = temp;

                                temp = arr2[b]; arr2[b] = arr2[a]; arr2[a] = temp;

                                temp = arr3[b]; arr3[b] = arr3[a]; arr3[a] = temp;
                        }
                }
        }
}
```

```
// Main
int main()
{
	// Define required variables
	int num;
	float current_time=0,avg_wt = 0, avg_tat = 0;

	printf("Number of Processes : ");
	scanf("%d", &num);

	// Incase the user types a negative value for num
	if(num < 0)
	{
		printf("Invalid Number of processes. Try again\n");
		main();
	}

	// Create arrays for storing Process ID, Arrival Time and Burst Time
	// CT (Completion Time, WT (Waiting Time), TAT (Turn Around Time)
	float PID[num], AT[num], BT[num], CT[num], WT[num], TAT[num];

	// Take all necessary inputs from user
	for(int i = 0; i < num; i++)
	{
		printf("Enter PID :");
		scanf("%f", &PID[i]);

		printf("Enter Arrival Time of Process %f :", PID[i]);
		scanf("%f", &AT[i]);

		printf("Enter Burst Time of Process %f :", PID[i]);
		scanf("%f", &BT[i]);
	}

	// Sort the arrays w.r.t Arrival Time
	sort(AT,PID,BT,0,num);

	for(int i = 0; i < num; i++)
	{
		// When there are no performable processes at current time, move
forward
		while (current_time < AT[i])
		{
			current_time = current_time + 1;
```

```
                }

                // To check number of processes with same Arrival Time
                int j = i;
                while (AT[j] == AT[j + 1])
                        j++;

                // Sort those processes based on PID
                sort(PID, AT, BT, i, j + 1);

        // Final Calculations
                current_time = current_time + BT[i];

                CT[i] = current_time;

                TAT[i] = CT[i] - AT[i];

                avg_tat = avg_tat + TAT[i];

                WT[i] = TAT[i] - BT[i];

                avg_wt = avg_wt + WT[i];
        }

        // Print Outputs
        printf("----------------------------------------------------------------\n \n");
        printf("PID          AT          BT          CT          TAT          WT\n");
        for (int i = 0; i < num; i++)
        {
                printf("%f     %f     %f     %f     %f     %f\n",PID[i], AT[i], BT[i], CT[i],
TAT[i], WT[i]);
        }
        printf("----------------------------------------------------------------\n \n");

        printf("Average Waiting Time : %f\n", avg_wt / num);
        printf("Average Turn Around Time : %f\n", avg_tat / num);

        return 0;
}
```

OS Lab Assignment-2

N. Sree Dhyuti
CED19I027

13/8/21

(1) FCFS Algorithm Code Explanation:

Step1: Take inputs from user (No of processes, PID, AT, BT)

Step2: Sort all the 3 arrays (PID, AT and BT) w.r.t Arrival Time (AT)

Step3: Choose the process with least Arrival Time and let it undergo completion.
If 2 or more processes have same arrival time, choose the process with least PID.

Step4: For every iteration, update time ~~x Completed~~ completion time, Waiting time, TurnAround time
~~step5~~ and store these values in arrays

Step5: Display outputs

Arrival Time: Time at which process arrives in queue

Burst Time: Time required in milliseconds by a process for its execution

Completion Time: Time at which a process completes its execution.

Turn around Time: Time diff b/w completion Time & Arrival Time

$$TAT = CT - AT$$

Waiting Time: Time taken for a process to start after it arrived in queue.

$$WT = TAT - BT$$

$$\text{average Waiting Time} = \frac{\sum_{i=1}^{n} WT_i}{n}, \quad n - \text{no of processes}$$

$$\text{average Turn Around Time} = \frac{\sum_{i=1}^{n} TAT_i}{n}, \quad n - \text{no of processes}$$

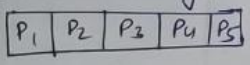
Note: When all arrival times are 0, the code will proceed w.r.t process ID

N. Sree Dhyuti
CED19I027

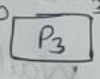
Example:
Schedule the following processes

Process ID	Arrival Time	Burst Time
1	4	5
2	6	4
3	0	3
4	6	2
5	5	4

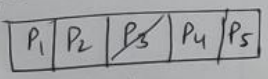
check array:



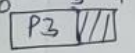
Sol) Here, the process with least Arrival Time = Process 3
So, we undergo process 3 first
Gantt chart:



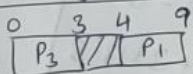
updated check array



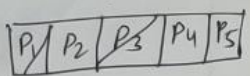
Now at time = 3, we have no processes in due
So, the next one second will be empty
Updated Gantt chart:



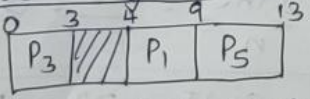
At time = 4, we have process 1
updated Gantt chart:



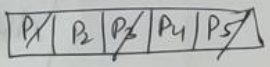
updated ~~Gantt~~ check array:



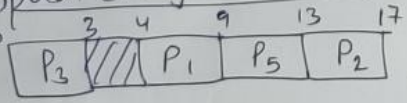
Before time = 9, we have 3 processes, where Process 5 has least arrival time.
So, we undergo process 5
updated Gantt Chart:



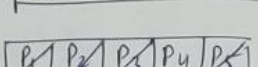
updated Check list:



Next we have 2 processes with same AT, we choose the one with less value of PID.
updated Gantt chart:



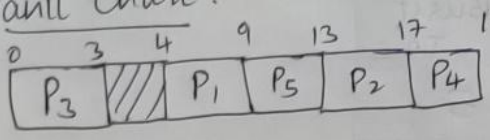
updated Checklist:



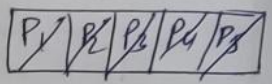
N. Sree Dhyuti
CED19I027

Finally, we undergo the last process P_4

Gantt chart:



check list:



~~Completion Times of~~

CT P_1 = 9

P_2 =

Completion Times:

$CT_{(P_1)} = 9$

$CT_{(P_2)} = 17$

$CT_{(P_3)} = 3$

$CT_{(P_4)} = 19$

$CT_{(P_5)} = 13$

Turn Around Times: (TAC)

$TAC_{(P_1)} = 9 - 4 = 5$

$TAC_{(P_2)} = 17 - 6 = 11$

$TAC_{(P_3)} = 3 - 0 = 3$

$TAC_{(P_4)} = 19 - 6 = 13$

$TAC_{(P_5)} = 13 - 5 = 8$

$\Sigma TAC = 5 + 11 + 3 + 13 + 8$

$\boxed{\Sigma TAC = 40}$

Waiting Time:

$WT_{(P_1)} = 5 - 5 = 0$

$WT_{(P_2)} = 11 - 4 = 7$

$WT_{(P_3)} = 3 - 3 = 0$

$WT_{(P_4)} = 13 - 2 = 11$

$WT_{(P_5)} = 8 - 4 = 4$

$\Sigma WT = 0 + 7 + 0 + 11 + 4$

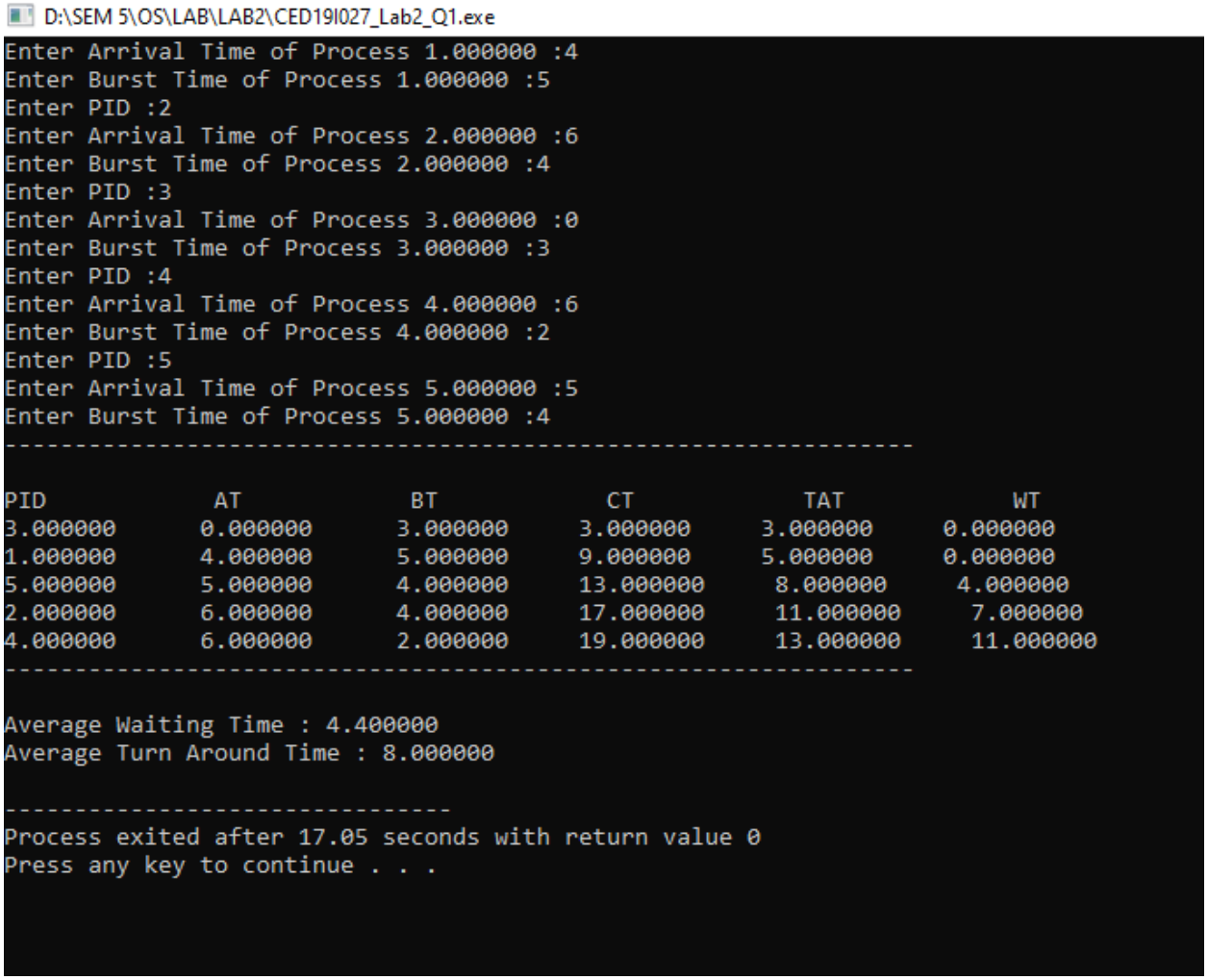
$= 22$

$\boxed{\Sigma WT = 22}$

$\boxed{\text{Average TAC} = \frac{\Sigma TAC}{n} = \frac{40}{5} = 8}$

$\boxed{\text{Average WT} = \frac{\Sigma WT}{n} = \frac{22}{5} = 4.4}$

CODE OUTPUT FOR THE ABOVE EXAMPLE :



---

(2) Implement SJF Algorithm.

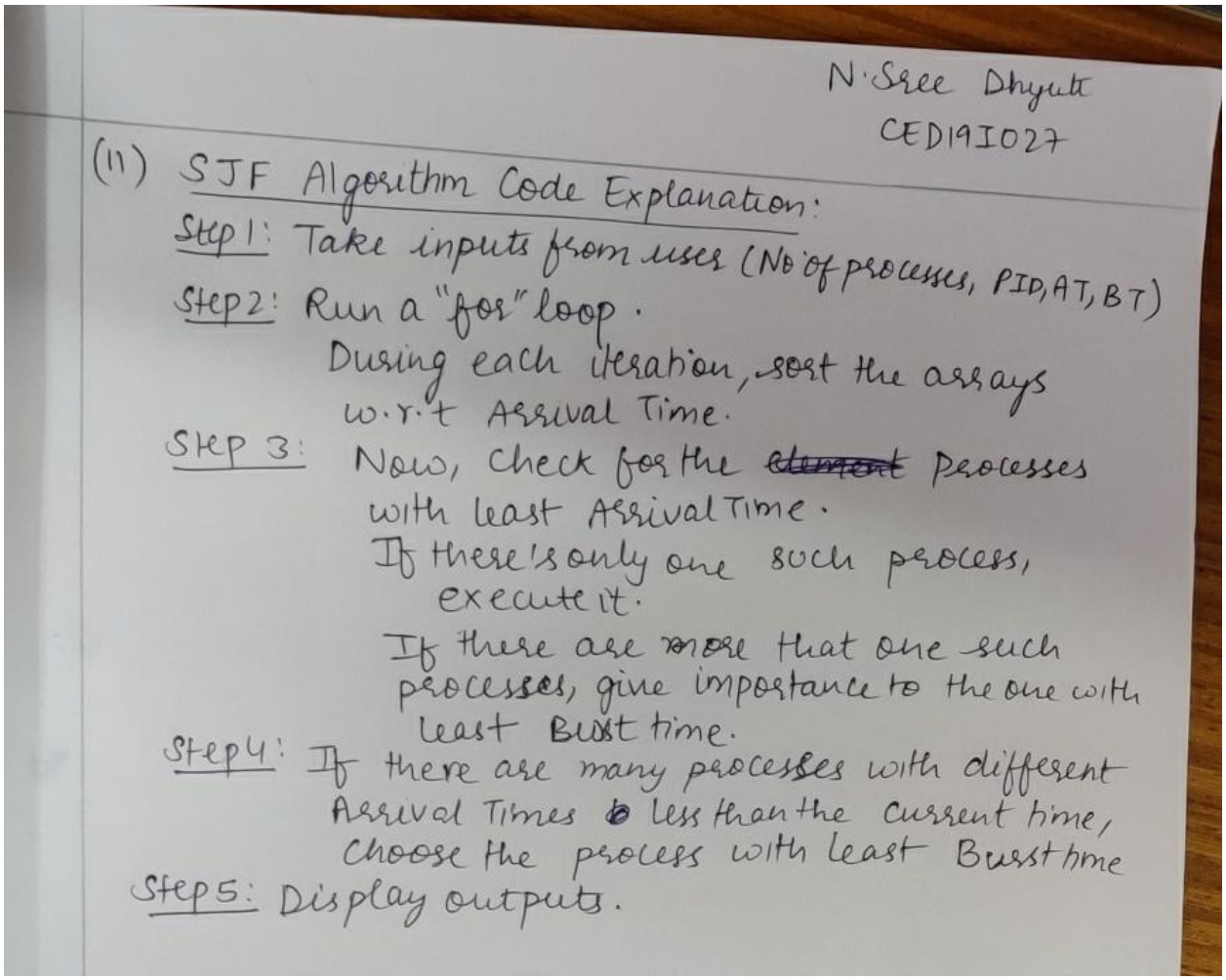
CODE:

```
// N Sree Dhyuti
// CED19I027
// Lab 2 : Q2

// Inclusion of required libraries
#include <stdio.h>


// Function to sort the elements in 3 arrays consecutively : Bubble Sort
// Here arr1 is the array with respect to which the sorting is happening
```

```
// Eg : sort (AT, BT, PID, 1, num -1) means we are sorting the 3 arrays wrt Arrival Time and in the array range 1 to num-1
void sort(float* arr1, float* arr2, float* arr3, int start, int end)
{
	int a, b;
	for(a = start; a < end - 1; a++)
	{
		for(b = a; b < end; b++)
		{
			if(arr1[a] > arr1[b])
			{
				float temp;
				// Swap
				temp = arr1[b]; arr1[b] = arr1[a]; arr1[a] = temp;

				temp = arr2[b]; arr2[b] = arr2[a]; arr2[a] = temp;

				temp = arr3[b]; arr3[b] = arr3[a]; arr3[a] = temp;
			}
		}
	}
}

// Main
int main()
{
	// Define required variables
	int num, j;
	float current_time=0,avg_wt = 0, avg_tat = 0;
	printf("Number of Processes : ");
	scanf("%d", &num);

	// Incase the user types a negative value for num
	if(num < 0)
	{
		printf("Invalid Number of processes. Try again\n");
		main();
	}

	// Create arrays for storing Process ID, Arrival Time and Burst Time
	// CT (Completion Time, WT (Waiting Time), TAT (Turn Around Time)
	float PID[num], AT[num], BT[num], CT[num], WT[num], TAT[num];
```

```
// Take all necessary inputs from user
for(int i = 0; i < num; i++)
{
        printf("Enter PID :");
        scanf("%f", &PID[i]);

        printf("Enter Arrival Time of Process %f :", PID[i]);
        scanf("%f", &AT[i]);

        printf("Enter Burst Time of Process %f :", PID[i]);
        scanf("%f", &BT[i]);
}

for(int i = 0; i < num; i++)
{
       // Sort the arrays w.r.t Arrival Time
       // Bubble Sort used here
        sort(AT,PID,BT,i,num);

        while (current_time < AT[i])
        {
                current_time = current_time + 1;
        }
   // Find the process with least BT which has AT <= current_time
   for(j = i; j < num; j++)
   {
        if(AT[j] > current_time)
                break;
        }
        if(i + j - 1 != num)
                j = j - 1;

        //Sort the selected range wrt Burst Time
        sort(BT, PID, AT, i, j + 1);

        current_time = current_time + BT[i];
        CT[i] = current_time;
   TAT[i] = CT[i] - AT[i];
   avg_tat = avg_tat + TAT[i];
   WT[i] = TAT[i] - BT[i];
   avg_wt = avg_wt + WT[i];
}
// Print all details
```

```
        printf("-----------------------------------------------------------------\n \n");
        printf("PID          AT          BT          CT          TAT          WT\n");
        for (int i = 0; i < num; i++)
        {
                printf("%f     %f     %f     %f     %f     %f\n",PID[i], AT[i], BT[i], CT[i], TAT[i],
WT[i]);
        }
        printf("-----------------------------------------------------------------\n \n");
        printf("Average Waiting Time : %f\n", avg_wt / num);
        printf("Average Turn Around Time : %f\n", avg_tat / num);
        return 0;
}
```



N. Sree Dhyuti
CED19I027

(11) SJF Algorithm Code Explanation:

Step 1: Take inputs from user (No of processes, PID, AT, BT)

Step 2: Run a "for" loop.
During each iteration, sort the arrays w.r.t Arrival Time.

Step 3: Now, check for the ~~element~~ processes with least Arrival Time.
If there's only one such process, execute it.
If there are more that one such processes, give importance to the one with least Burst time.

Step 4: If there are many processes with different Arrival Times ~~is~~ less than the current time, choose the process with least Burst time

Step 5: Display outputs.

N. Sree Dhyuti
CED19I027

SJF or Shortest Job First Algorithm is more preffered over FCFS Scheduling Algorithm as it overcomes "Convoy Effect" and reduces overall waiting time.

Convoy effect:

If processes with higher Burst Time arrive before processes with lesser burst time, then smaller processes have to wait for longer processes to release the CPU.

N. Sree Dhyuti
CED19I027

Example: Schedule the following processes in SJF.

(same example used for FCFS also)

Process ID	Arrival Time	Busst Time
1	4	5
2	6	4
3	0	3
4	6	2
5	5	4

check array:

P1	P2	P3	P4	P5

We initially find & execute the process with least arrival Time ⇒ Process 3

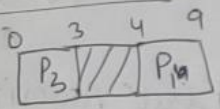
Gantt chart:

updated Check array:

P1	P2	~~P3~~	P4	P5

Now at time = 3, there are no processes in due.
So we move to time = 4.
Here we have process 1.

updated Gantt chart:

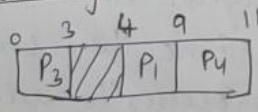


updated check array:

~~P1~~	P2	~~P3~~	P4	P5

at time = 9, we have Process 2, 4 & 5 who have already arrived in queue.
Of them, the process with least Busst time is P4.
So we execute P4

updated Gantt chart:

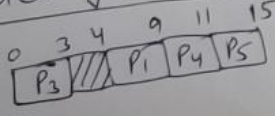


updated Check array:

~~P1~~	P2	~~P3~~	~~P4~~	P5

Next process with least Busst Time is P5 & least Arrival Time

updated Gantt Chart:



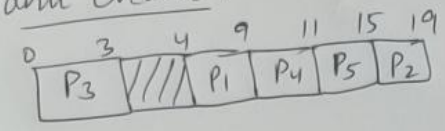
updated Check array:

~~P1~~	P2	~~P3~~	~~P4~~	~~P5~~

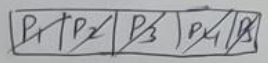
N. Sree Dhyuti
CED19I027

We then finish the final left Process P_2

Gantt Chart:



Check array:



Completion Time:

$CT(P_1) = 9$

$CT(P_2) = 19$

$CT(P_3) = 3$

$CT(P_4) = 11$

$CT(P_5) = 15$

Turn Around Time:

$TAT(P_1) = 9 - 4 = 5$

$TAT(P_2) = 19 - 6 = 13$

$TAT(P_3) = 3 - 0 = 3$

$TAT(P_4) = 11 - 6 = 5$

$TAT(P_5) = 15 - 5 = 10$

$\Sigma TAT = 5 + 13 + 3 + 5 + 10$

$\boxed{\Sigma TAT = 36}$

Waiting Time

$WT(P_1) = 5 - 5 = 0$

$WT(P_2) = 13 - 4 = 9$

$WT(P_3) = 3 - 3 = 0$

$WT(P_4) = 5 - 2 = 3$

$WT(P_5) = 10 - 4 = 6$

$\Sigma WT = 0 + 9 + 0 + 3 + 6$

$\boxed{\Sigma WT = 18}$

$$\boxed{\text{Avg. } TAT = \frac{\Sigma TAT}{n} = \frac{36}{5} = 7.2}$$

$$\boxed{\text{Avg. } WT = \frac{\Sigma WT}{n} = \frac{18}{5} = 3.6}$$

Notice that for the same example problem, FCFS algorithm resulted in greater values of avg TAT & avg. WT.

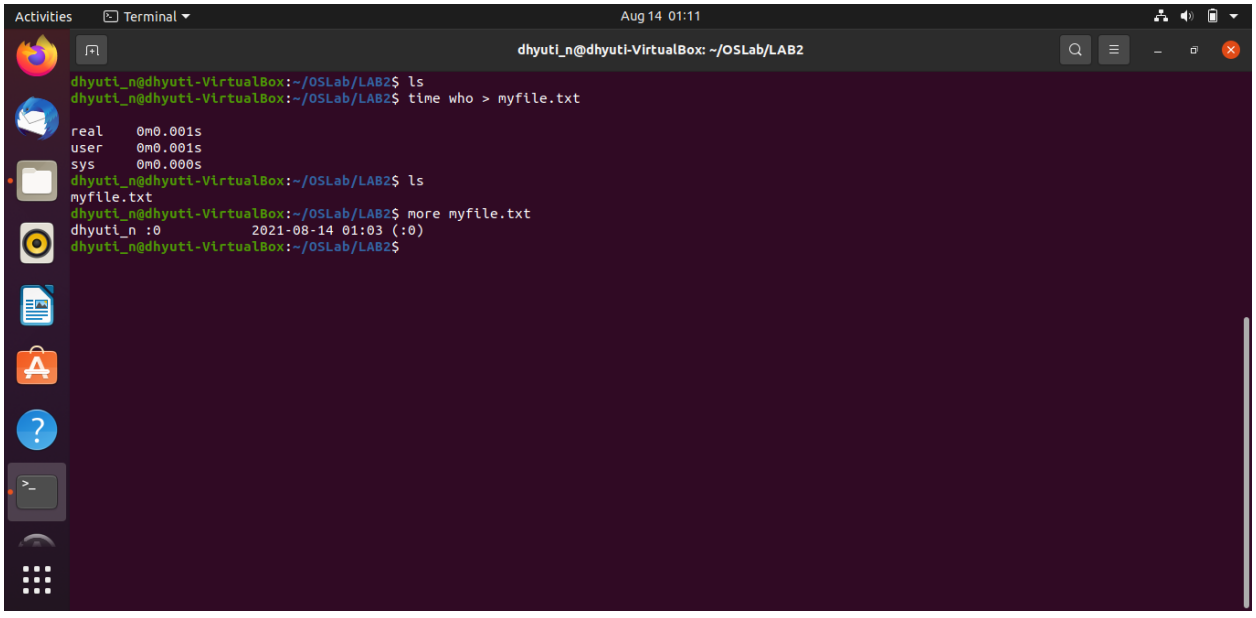
CODE OUTPUT FOR THE ABOVE EXAMPLE :

D:\SEM 5\OS\LAB\LAB2\CED19I027_Lab2_Q2.exe

```
Enter Arrival Time of Process 1.000000 :4
Enter Burst Time of Process 1.000000 :5
Enter PID :2
Enter Arrival Time of Process 2.000000 :6
Enter Burst Time of Process 2.000000 :4
Enter PID :3
Enter Arrival Time of Process 3.000000 :0
Enter Burst Time of Process 3.000000 :3
Enter PID :4
Enter Arrival Time of Process 4.000000 :6
Enter Burst Time of Process 4.000000 :2
Enter PID :5
Enter Arrival Time of Process 5.000000 :5
Enter Burst Time of Process 5.000000 :4
-----------------------------------------------------------------

PID             AT             BT             CT             TAT             WT
3.000000       0.000000       3.000000       3.000000       3.000000       0.000000
1.000000       4.000000       5.000000       9.000000       5.000000       0.000000
4.000000       6.000000       2.000000       11.000000       5.000000       3.000000
5.000000       5.000000       4.000000       15.000000       10.000000       6.000000
2.000000       6.000000       4.000000       19.000000       13.000000       9.000000
-----------------------------------------------------------------

Average Waiting Time : 3.600000
Average Turn Around Time : 7.200000

--------------------------------
Process exited after 18.42 seconds with return value 0
Press any key to continue . . .
```

---

(3)



N. Sree Dhyuti
CED19I027

(3) Use 'time' and 'who' commands in sequence (in one line) such that the output of time will display on the screen and output of who will be redirected to a file called myfile. Use more command to check the contents of a file.

'Time' is a command which displays the real time, usertime & system time spent on executing a command that follows it.

syntax: time [option] [command]

The command "time who" basically gives the information about who is currently logged in the LINUX OS & the time taken to execute this command.

To redirect this information to another file the command will be

"time who > myfile.txt"

This will display time on the screen & redirect "who" value to the file. If 'myfile.txt' is not available, it creates one for itself.

(4) ps

CED19I027
N. Sree Dhyuti

(4) Explore process management commands like ps, top, glances, kill, pkill, Pgrep etc.

ps:
- command for viewing process status
- gives information about
  - PID: Process ID
  - TTY: Terminal Type of User
  - TIME: amt of CPU in minutes & seconds that the process is running
  - CMD: name of the command that launched the process.

syntax: ps [options]

ps -A } - show all running processes
ps -e }

ps -a } - shows all processes except both session leaders & processes that aren't associated with a terminal

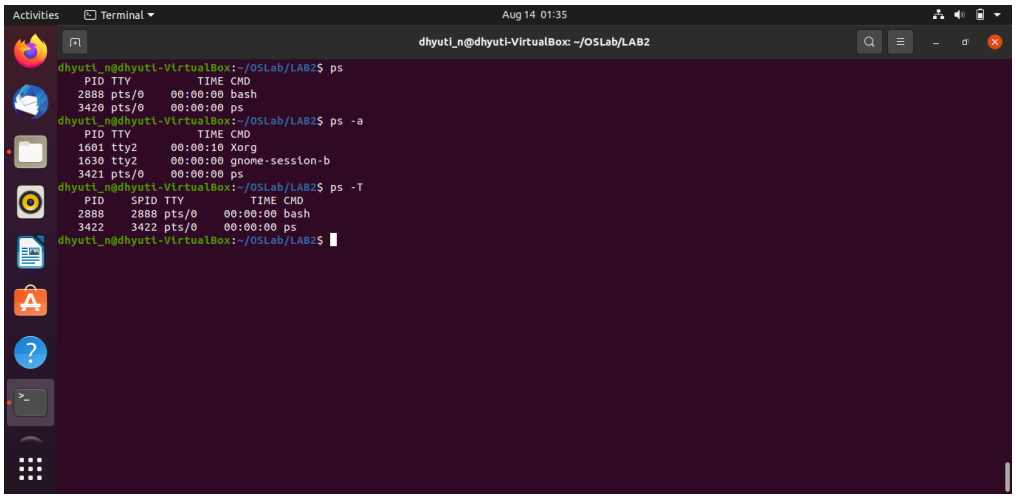
ps -d } - all processes except session leaders

ps -T } - all processes in this terminal

ps -r } - all running processes

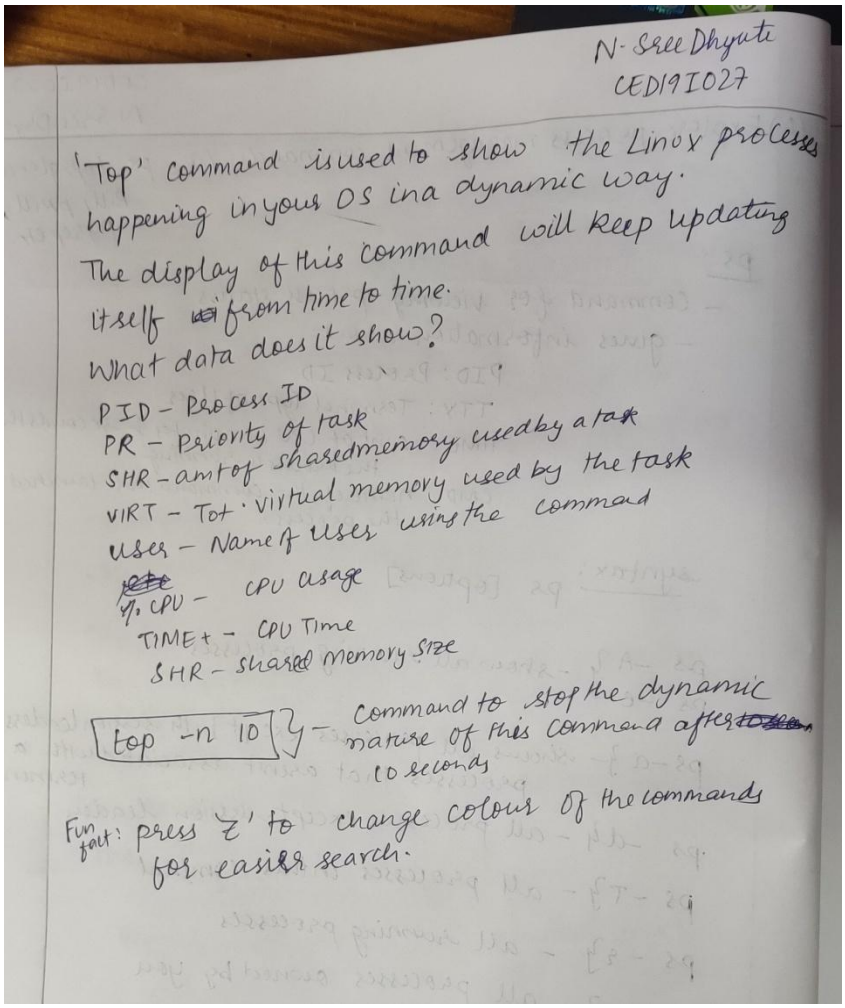
ps -x } - all processes owned by you

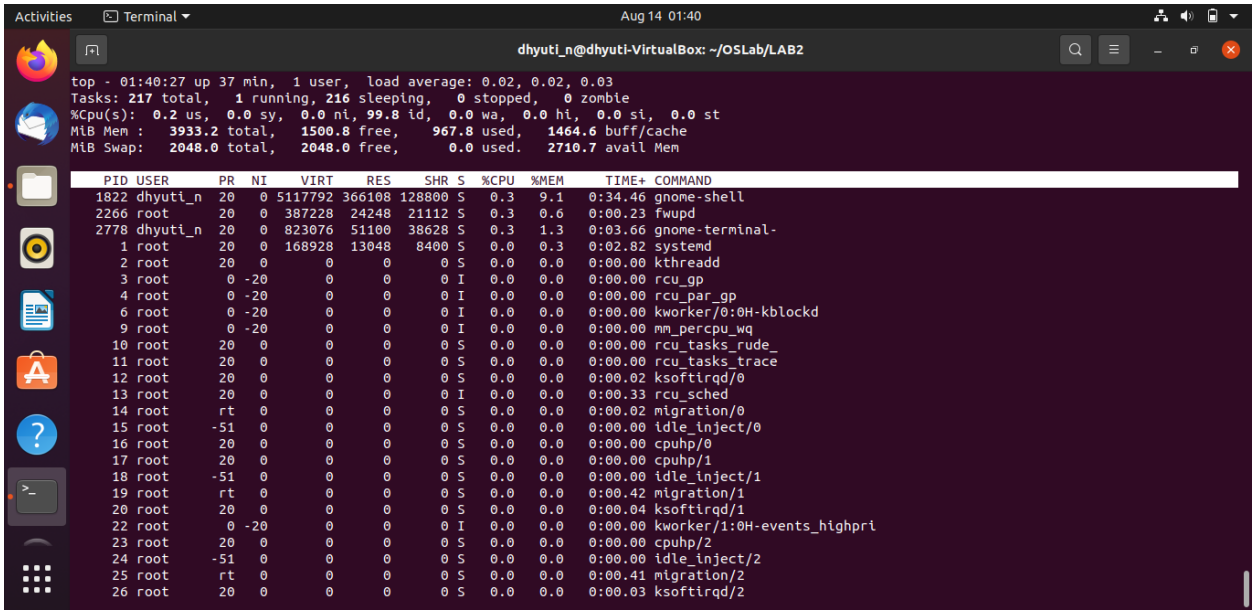
Session Leaders are the process ID of 1st member of the session.


Activities
Terminal
Aug 14 01:35
dhyuti_n@dhyuti-VirtualBox: ~/OSLab/LAB2
dhyuti_n@dhyuti-VirtualBox:~/OSLab/LAB2$ ps
PID TTY TIME CMD
2888 pts/0 00:00:00 bash
3420 pts/0 00:00:00 ps
dhyuti_n@dhyuti-VirtualBox:~/OSLab/LAB2$ ps -a
PID TTY TIME CMD
1601 tty2 00:00:10 Xorg
1630 tty2 00:00:00 gnome-session-b
3421 pts/0 00:00:00 ps
dhyuti_n@dhyuti-VirtualBox:~/OSLab/LAB2$ ps -T
PID SPID TTY TIME CMD
2888 2888 pts/0 00:00:00 bash
3422 3422 pts/0 00:00:00 ps
dhyuti_n@dhyuti-VirtualBox:~/OSLab/LAB2$

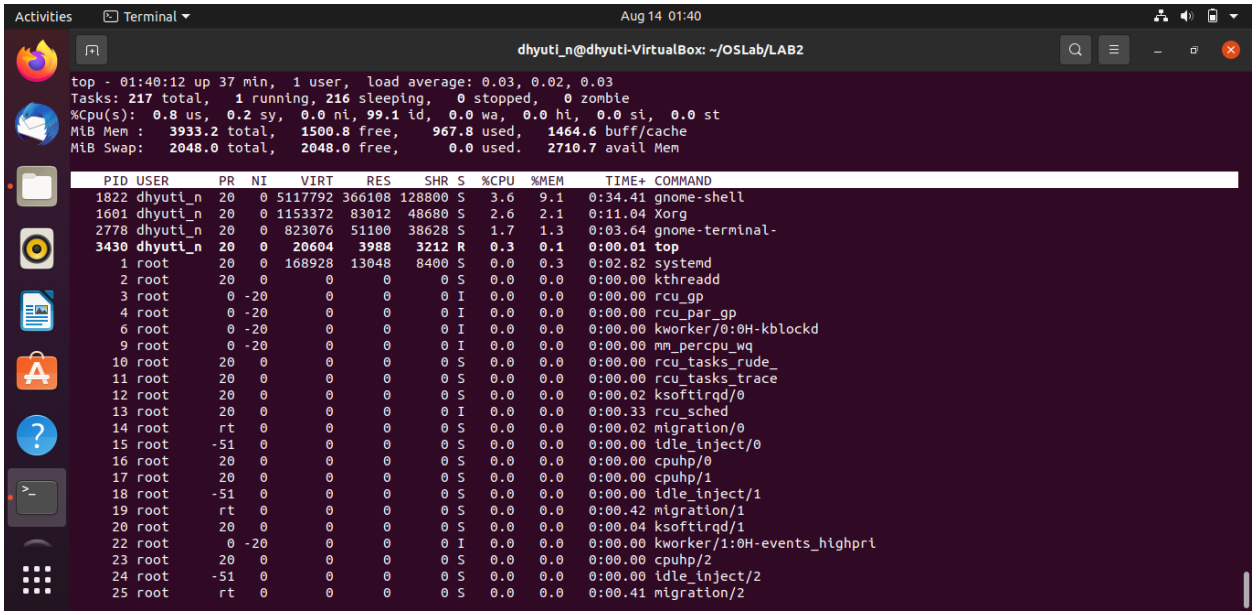
---

(4) top


N. Sree Dhyuti
CED19I027
'Top' command is used to show the Linux processes happening in your OS in a dynamic way.
The display of this command will keep updating itself from time to time.
What data does it show?
PID - Process ID
PR - Priority of task
SHR - amt of shared memory used by a task
VIRT - Tot. virtual memory used by the task
User - Name of user using the command
%CPU - CPU usage
TIME+ - CPU Time
SHR - Shared memory size
top -n 10 - Command to stop the dynamic nature of this command after 10 seconds
Fun fact: press 'z' to change colour of the commands for easier search.

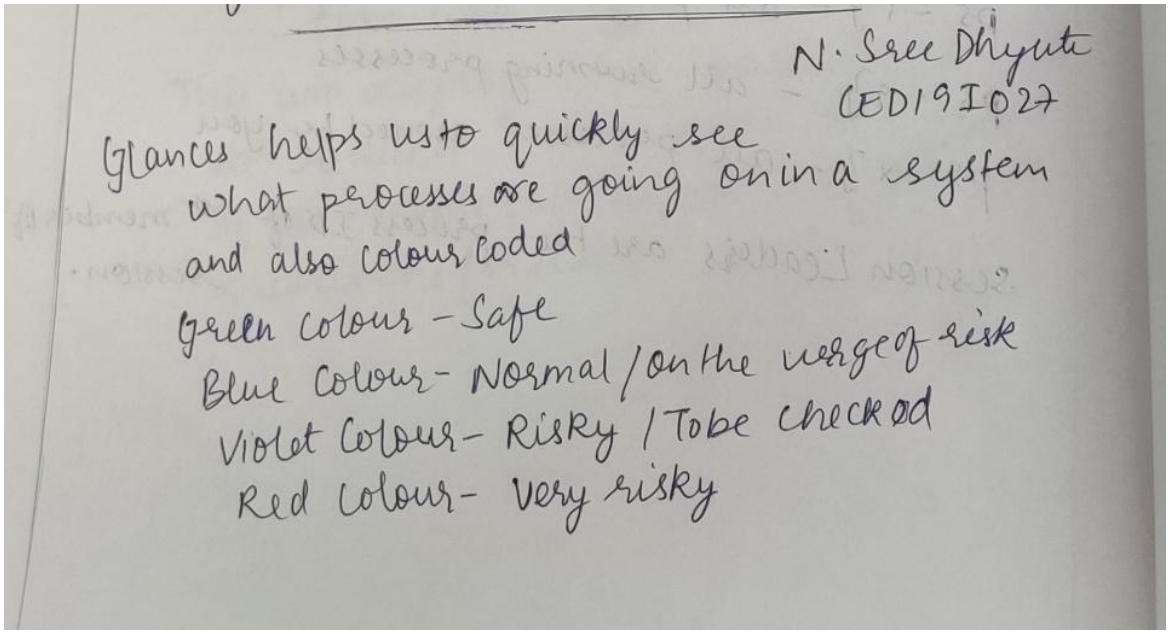


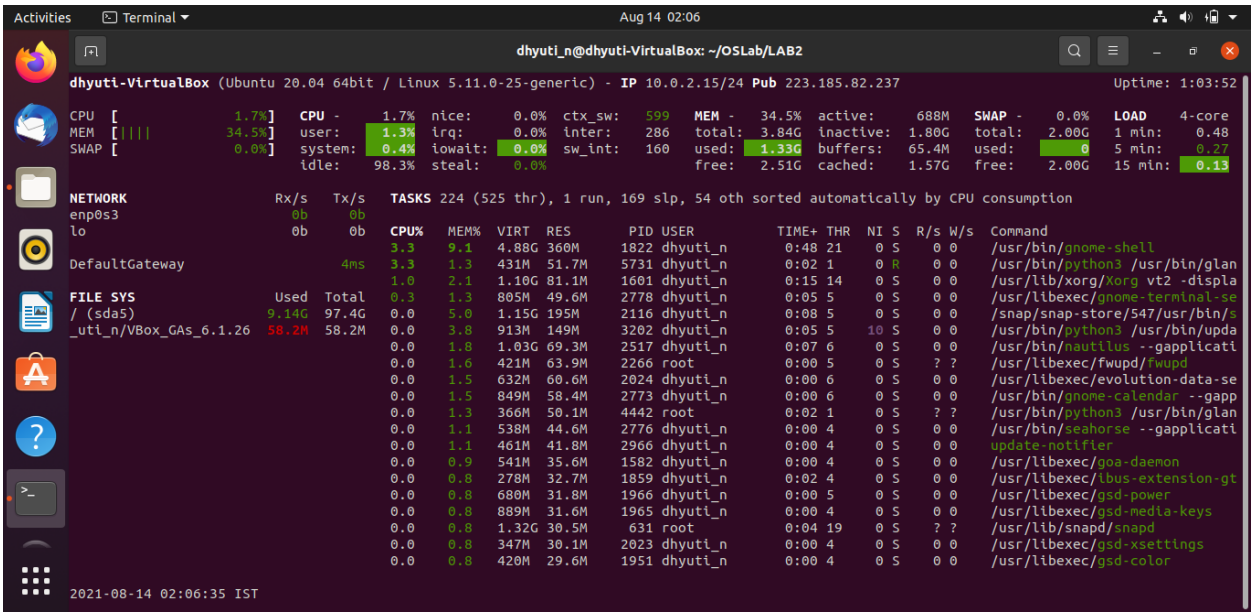
Notice the change in display after a few seconds...



---

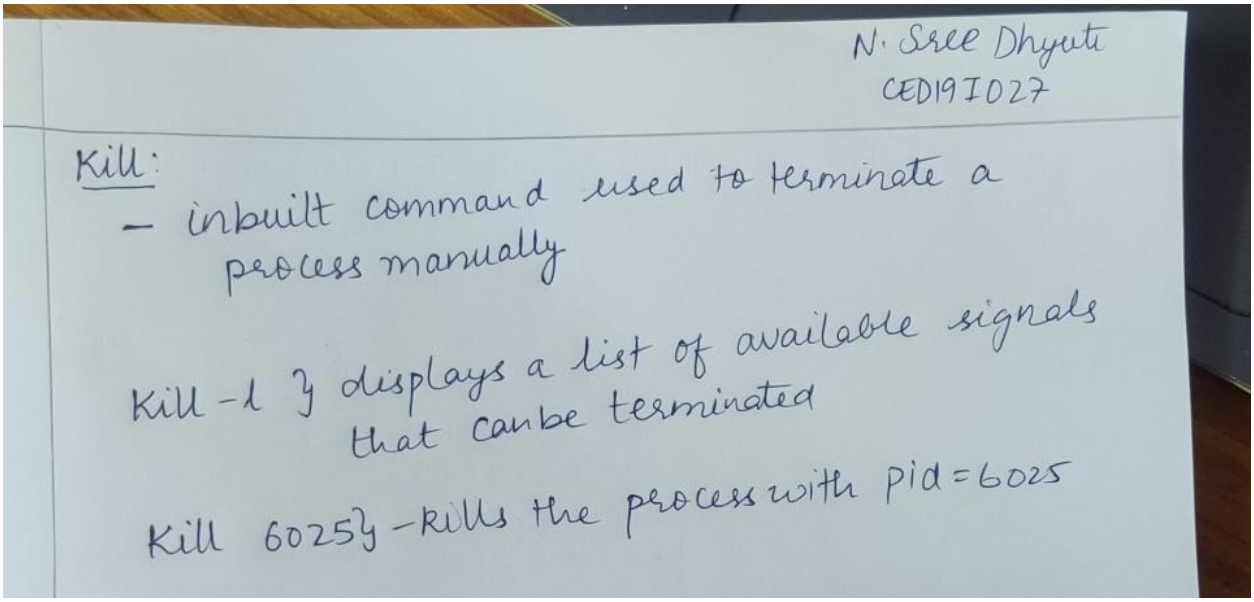
(4) glances

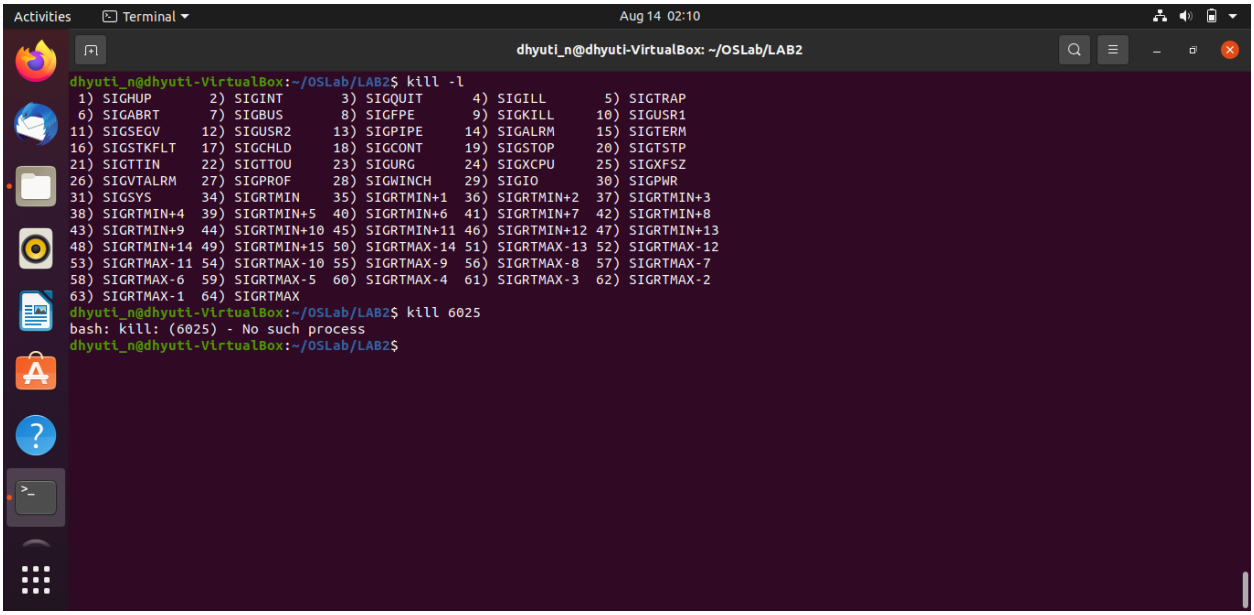

N. Sree Dhyuti
CED19I027
Glances helps us to quickly see what processes are going on in a system and also colour coded
Green colour - Safe
Blue Colour - Normal / on the verge of risk
Violet Colour - Risky / To be checked
Red Colour - Very risky


Activities
Terminal
Aug 14 02:06
dhyuti_n@dhyuti-VirtualBox: ~/OSLab/LAB2
dhyuti-VirtualBox (Ubuntu 20.04 64bit / Linux 5.11.0-25-generic) - IP 10.0.2.15/24 Pub 223.185.82.237
Uptime: 1:03:52
CPU [1.7%] CPU - 1.7% nice: 0.0% ctx_sw: 599 MEM - 34.5% active: 688M SWAP - 0.0% LOAD 4-core
MEM [|||| 34.5%] user: 1.3% irq: 0.0% inter: 286 total: 3.84G inactive: 1.80G total: 2.00G 1 min: 0.48
SWAP [0.0%] system: 0.4% iowait: 0.0% sw_int: 160 used: 1.33G buffers: 65.4M used: 0 5 min: 0.27
idle: 98.3% steal: 0.0% free: 2.51G cached: 1.57G free: 2.00G 15 min: 0.13
NETWORK Rx/s Tx/s
enp0s3 0b 0b
lo 0b 0b
DefaultGateway 4ms
FILE SYS Used Total
/ (sda5) 9.14G 97.4G
_uti_n/VBox_GAs_6.1.26 58.2M 58.2M
TASKS 224 (525 thr), 1 run, 169 slp, 54 oth sorted automatically by CPU consumption
CPU% MEM% VIRT RES PID USER TIME+ THR NI S R/s W/s Command
3.3 9.1 4.88G 360M 1822 dhyuti_n 0:48 21 0 S 0 0 /usr/bin/gnome-shell
3.3 1.3 431M 51.7M 5731 dhyuti_n 0:02 1 0 R 0 0 /usr/bin/python3 /usr/bin/glan
1.0 2.1 1.10G 81.1M 1601 dhyuti_n 0:15 14 0 S 0 0 /usr/lib/xorg/Xorg vt2 -displa
0.3 1.3 805M 49.6M 2778 dhyuti_n 0:05 5 0 S 0 0 /usr/libexec/gnome-terminal-se
0.0 5.0 1.15G 195M 2116 dhyuti_n 0:08 5 0 S 0 0 /snap/snap-store/547/usr/bin/s
0.0 3.8 913M 149M 3202 dhyuti_n 0:05 5 10 S 0 0 /usr/bin/python3 /usr/bin/upda
0.0 1.8 1.03G 69.3M 2517 dhyuti_n 0:07 6 0 S 0 0 /usr/bin/nautilus --gapplicati
0.0 1.6 421M 63.9M 2266 root 0:00 5 0 S ? ? /usr/libexec/fwupd/fwupd
0.0 1.5 632M 60.6M 2024 dhyuti_n 0:00 6 0 S 0 0 /usr/libexec/evolution-data-se
0.0 1.5 849M 58.4M 2773 dhyuti_n 0:00 6 0 S 0 0 /usr/bin/gnome-calendar --gapp
0.0 1.3 366M 50.1M 4442 root 0:02 1 0 S ? ? /usr/bin/python3 /usr/bin/glan
0.0 1.1 538M 44.6M 2776 dhyuti_n 0:00 4 0 S 0 0 /usr/bin/seahorse --gapplicati
0.0 1.1 461M 41.8M 2966 dhyuti_n 0:00 4 0 S 0 0 update-notifier
0.0 0.9 541M 35.6M 1582 dhyuti_n 0:00 4 0 S 0 0 /usr/libexec/goa-daemon
0.0 0.8 278M 32.7M 1859 dhyuti_n 0:02 4 0 S 0 0 /usr/libexec/ibus-extension-gt
0.0 0.8 680M 31.8M 1966 dhyuti_n 0:00 5 0 S 0 0 /usr/libexec/gsd-power
0.0 0.8 889M 31.6M 1965 dhyuti_n 0:00 4 0 S 0 0 /usr/libexec/gsd-media-keys
0.0 0.8 1.32G 30.5M 631 root 0:04 19 0 S ? ? /usr/lib/snapd/snapd
0.0 0.8 347M 30.1M 2023 dhyuti_n 0:00 4 0 S 0 0 /usr/libexec/gsd-xsettings
0.0 0.8 420M 29.6M 1951 dhyuti_n 0:00 4 0 S 0 0 /usr/libexec/gsd-color
2021-08-14 02:06:35 IST

---

(4) Kill


N. Sree Dhyuti
CED19I027
Kill:
– inbuilt command used to terminate a process manually
Kill -l } displays a list of available signals that can be terminated
Kill 6025} – kills the process with pid = 6025


Activities
Terminal
Aug 14 02:10
dhyuti_n@dhyuti-VirtualBox: ~/OSLab/LAB2
dhyuti_n@dhyuti-VirtualBox:~/OSLab/LAB2$ kill -l
1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL 5) SIGTRAP
6) SIGABRT 7) SIGBUS 8) SIGFPE 9) SIGKILL 10) SIGUSR1
11) SIGSEGV 12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP
21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU 25) SIGXFSZ
26) SIGVTALRM 27) SIGPROF 28) SIGWINCH 29) SIGIO 30) SIGPWR
31) SIGSYS 34) SIGRTMIN 35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
dhyuti_n@dhyuti-VirtualBox:~/OSLab/LAB2$ kill 6025
bash: kill: (6025) - No such process
dhyuti_n@dhyuti-VirtualBox:~/OSLab/LAB2$

---

(4)pkill

N. Sree Dhyuti
CED19I027

pkill:

- part of "procps" package, which is pre-installed

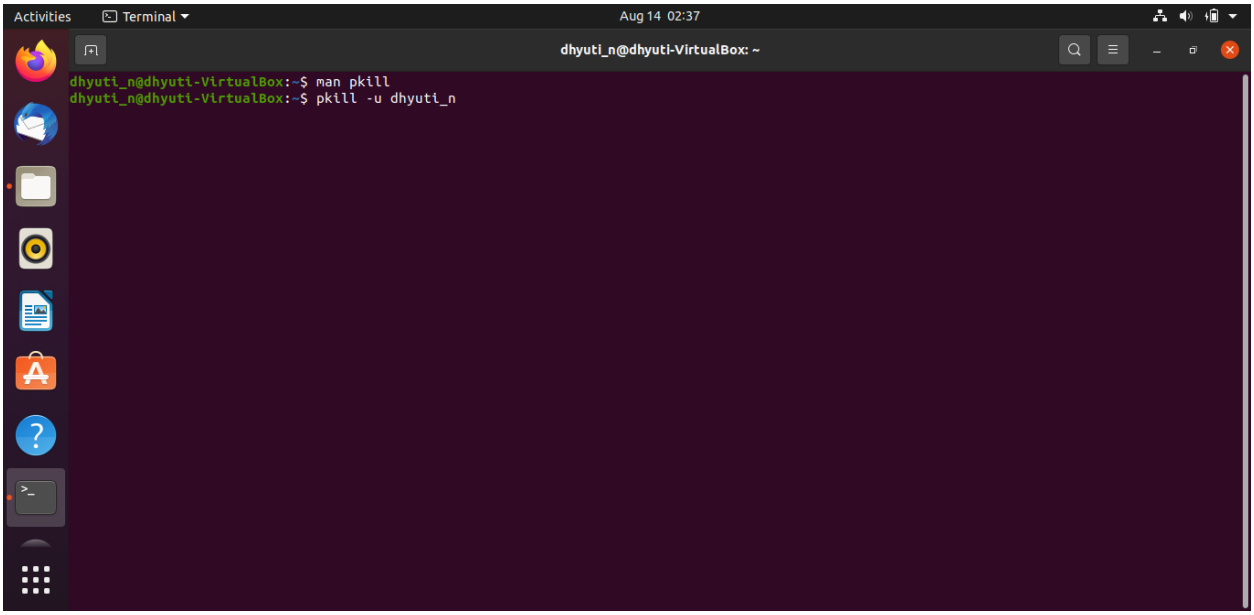
syntax:

pkill [options] <pattern>

pkill -u dhyuti_n } command to completely kill all running processes by the user dhyuti_n

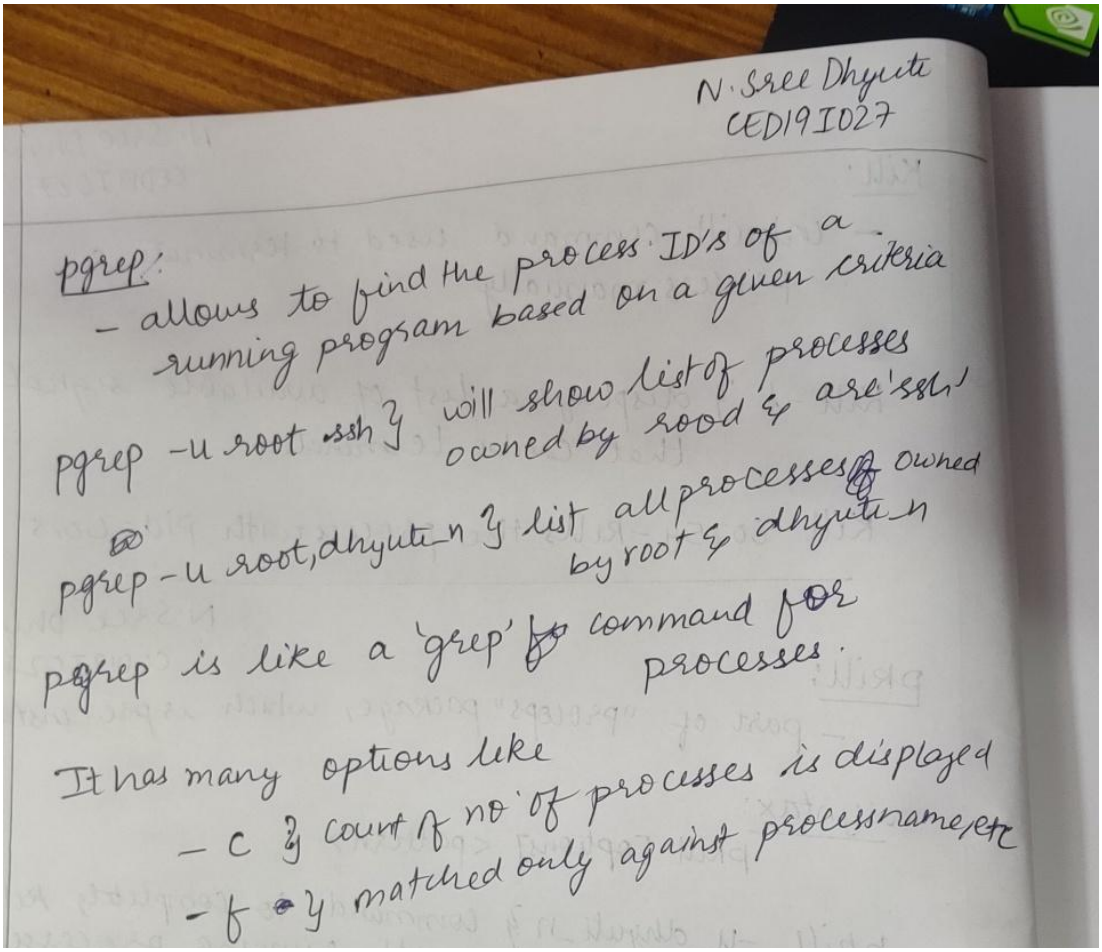
pkill -9 -n screen } - kill most recently created screen

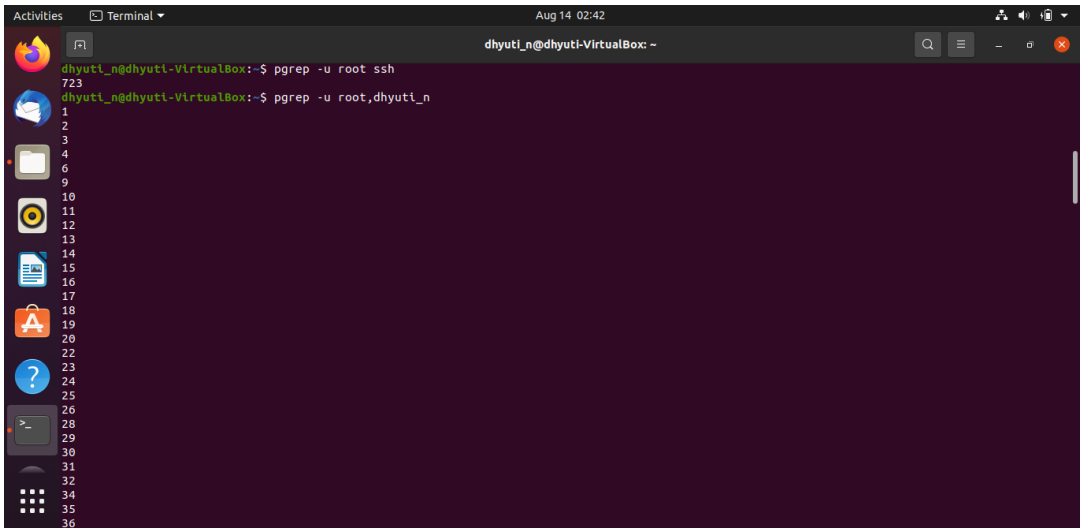
PKill sends the specified signal to each process instead of listing them



---

(4) pgrep


N. Sree Dhyuti
CED19I027
pgrep:
- allows to find the process ID's of a running program based on a given criteria
pgrep -u root ssh ; will show list of processes owned by root & are 'ssh'
pgrep -u root,dhyuti_n ; list all processes owned by root & dhyuti_n
pgrep is like a 'grep' command for processes.
It has many options like
-c ; count of no. of processes is displayed
-f ; matched only against processname, etc


Activities
Terminal
Aug 14 02:42
dhyuti_n@dhyuti-VirtualBox: ~
dhyuti_n@dhyuti-VirtualBox:~$ pgrep -u root ssh
723
dhyuti_n@dhyuti-VirtualBox:~$ pgrep -u root,dhyuti_n
1
2
3
4
6
9
10
11
12
13
14
15
16
17
18
19
20
22
23
24
25
26
28
29
30
31
32
34
35
36

THE END