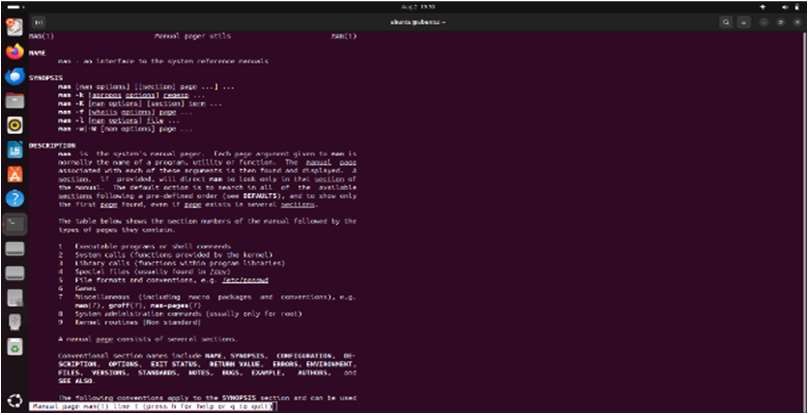
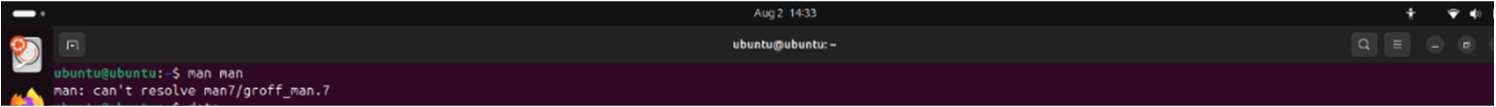
ASSIGNMENT 1

Perform the following tasks in the Linux shell with the help of appropriate commands:

1. Check the manual for the ‘man’ command.

>> man man



1. Display the current date and time.

>>date



1. Display the calendar of the current month.

>>cal

1. Display the current logged-in user.

>>whoami



1. Display more information about the current logged-in user.

>>id

1. Display the current working directory.

>>pwd



1. Check the contents of the current working directory.

>> ls



1. Make a new directory ‘Folder1’ in your current working directory.

>>mkdir Folder1

1. Enter the newly created directory Folder1 and go up one level.

>>cd Folder1

>>cd ..

1. Delete the newly created directory Folder1.

>>rmdir Folder1



1. Concatenate 2 files and display the resulting content:
   * Create 2 new files, name them ‘File1.txt’ and ‘File2.txt’ respectively.
   * Write ‘Hello’ in File1.txt and ‘World!’ in File2.txt.
   * Concatenate these 2 files, display the content, and save the result in ‘File3.txt’.
   * Delete File2.txt. Now compare File1.txt and File3.txt.

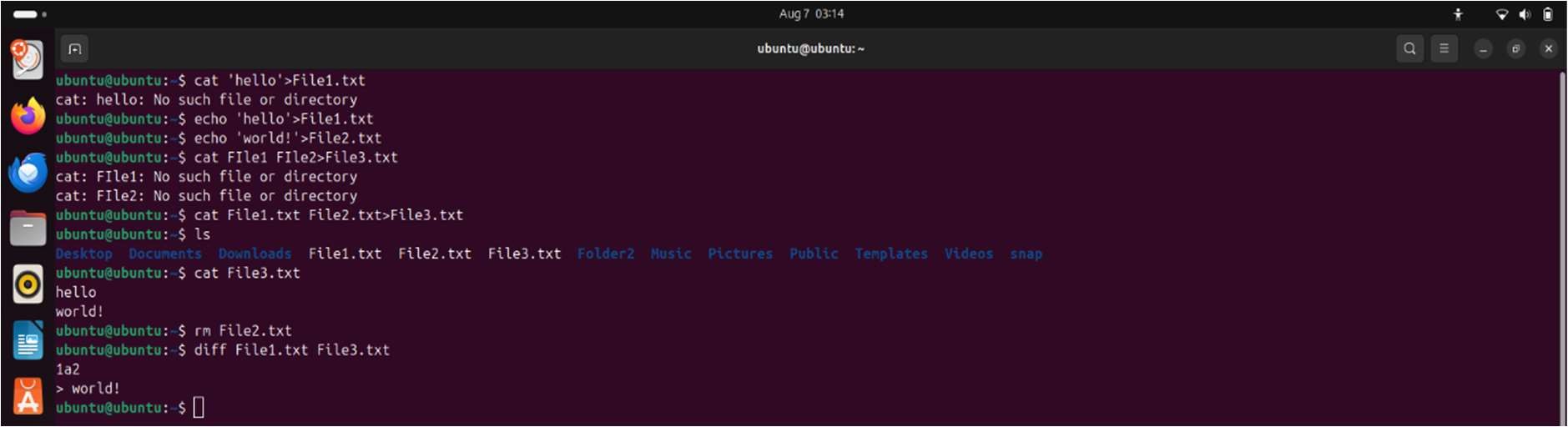
>>echo ‘hello’>File1.txt

>>echo ‘world!’>File2.txt

>>cat File1.txt File2.txt>File3.txt

>>cat File3.txt

>>rm File2.txt

>>diff File1.txt File3.txt

2

1. Check and display the word count in File3.txt.

>> wc File3.txt



1. Lexicographically sort the files and folders in the current working directory using the pipe operator along with other appropriate commands.

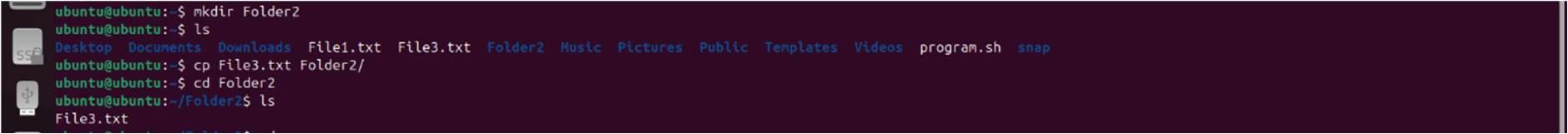
>> ls|sort



1. Make a new directory ‘Folder2’ in your current working directory. Copy and paste File3.txt into the aforementioned directory.

>>mkdir Folder2

>>cp File3.txt Folder2/



1. Check the read / write permission of File3.txt.

>>ls -l File3.txt



1. Set the following permissions for File1.txt:
   * File owner permission as read-only.
   * Group file owner permission as read and write.
   * Other user permission as read, write and execute.
   * All other system user permissions as read, write and execute.

>>chmod 467 File1.txt



3

ASSIGNMENT 2

1. Use the ‘ps’ command to perform the following operations:
   1. Display the process status of the current shell.
   2. Display the process status of all the running processes.
   3. Display the process status of all the running processes in a full format.
   4. Display the process status of all the running processes except session leader.
   5. View all the running processes.

>> ps $$



>> ps -e



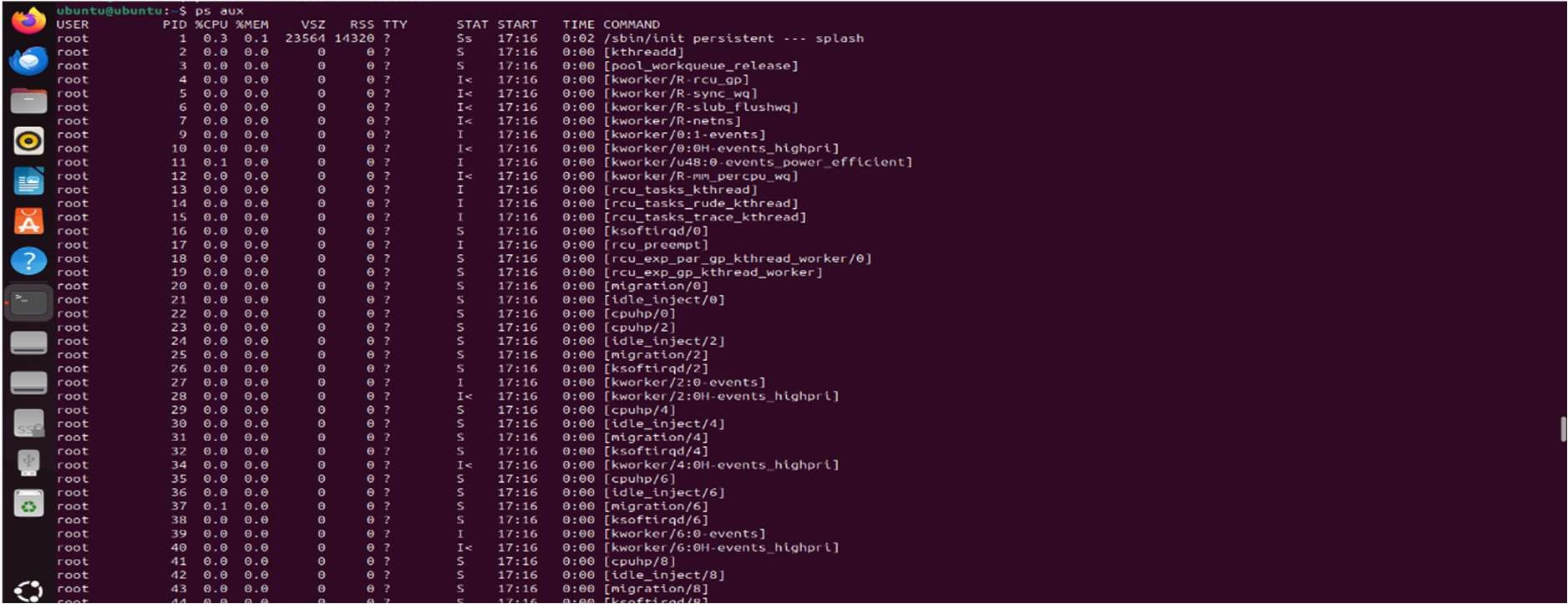
>> ps -ef



>> ps -e –no-headers --deselect



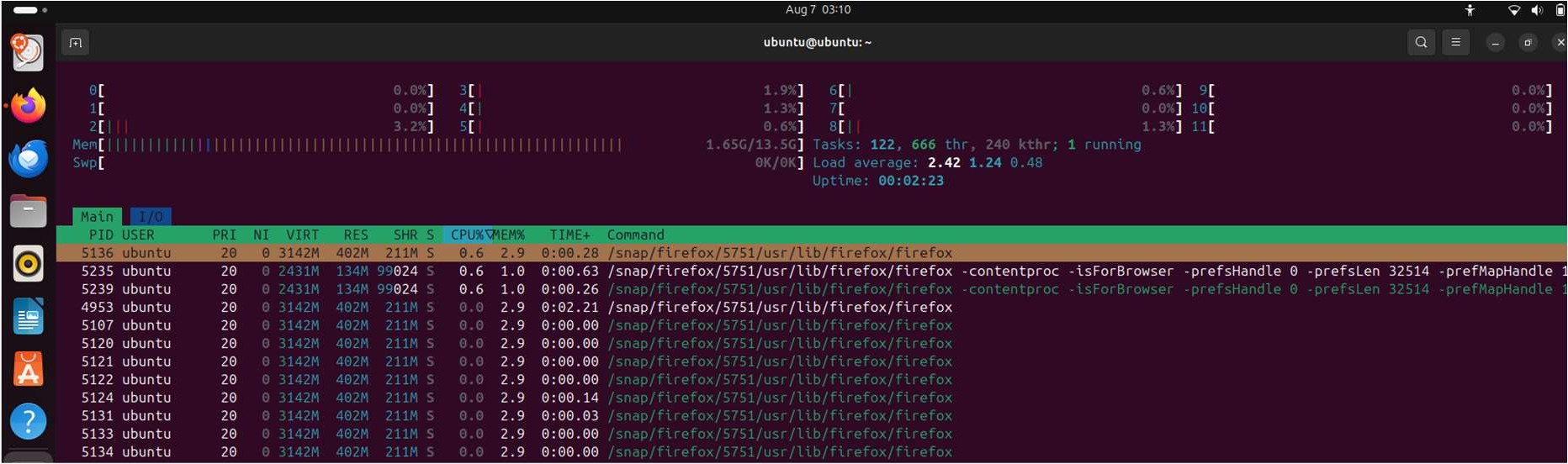
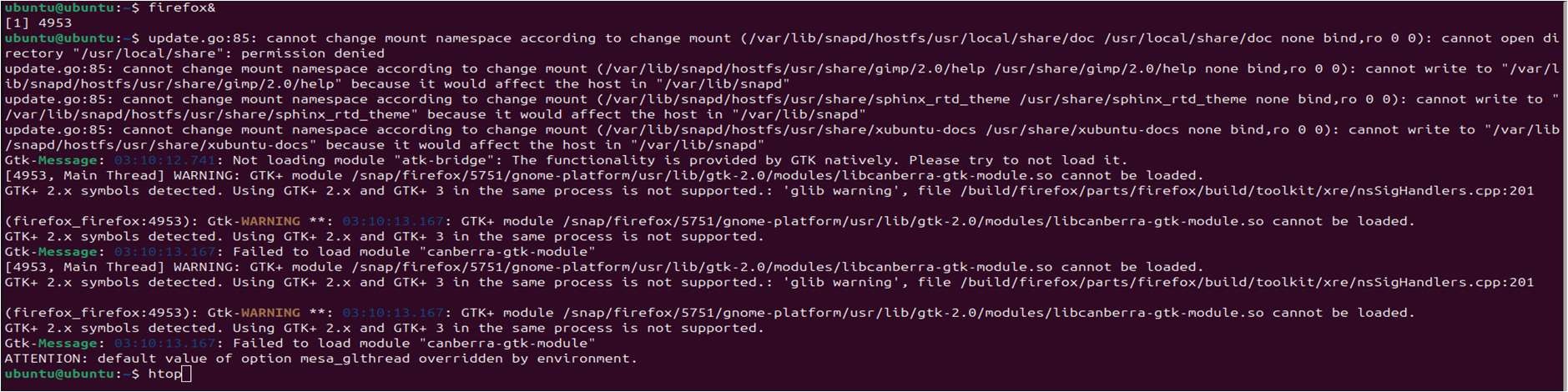
>>ps aux



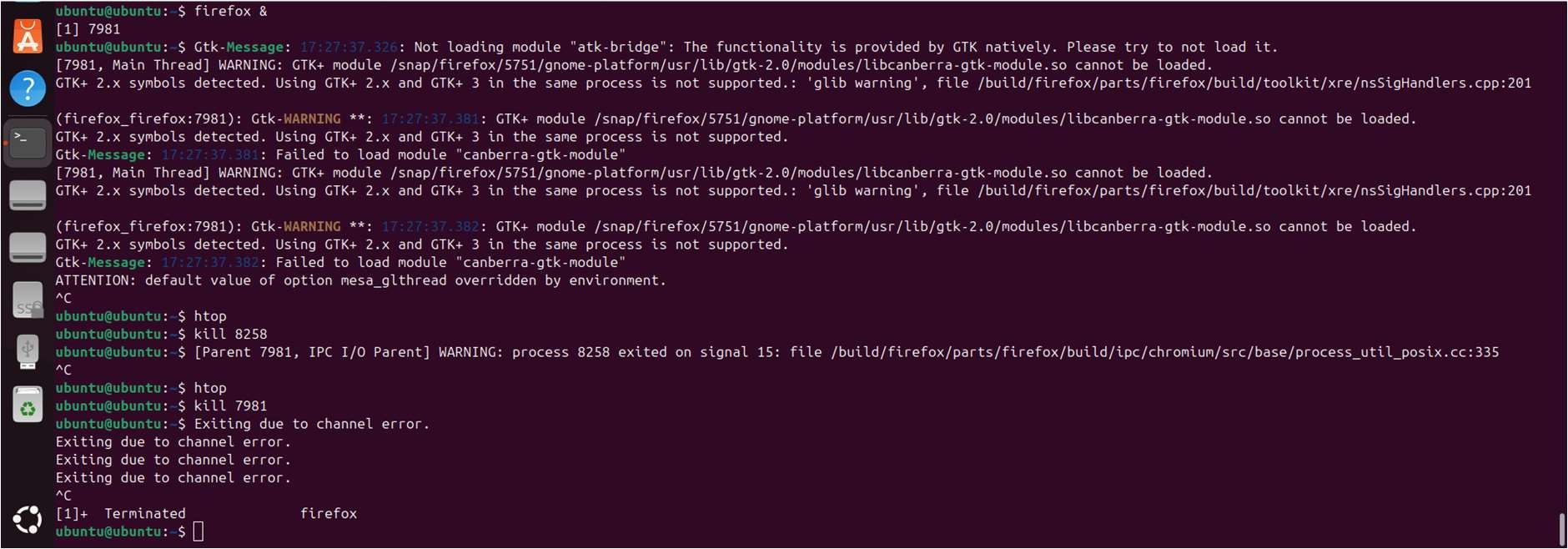
1. Open the Firefox browser. Now using the ‘htop’ program, display the following:
   1. The PID(s) of the Firefox process / processes.
   2. The owner of the process / processes.
   3. Virtual memory being consumed by the process / processes.
   4. The percentage of the processor time used by the process / processes.
   5. The percentage of physical RAM used by the process / processes.
   6. The name of the command that started the process / processes.

>>firefox &

>>htop



1. Kill a specific process by using its PID.

>> kill <PID>

1. Kill all the processes in the system.

>> sudo kill -9 -1

1. Open the Firefox browser. Observe and state the following using appropriate commands:
   1. Parent process

>>ps -o pid

* 1. Child process / processes

>>pgrep -p 10023

* 1. PID of the associated processes

>>pidof firefox

* 1. PPID of the associated processes

>>ps -o ppid= -p $(pidof firefox)

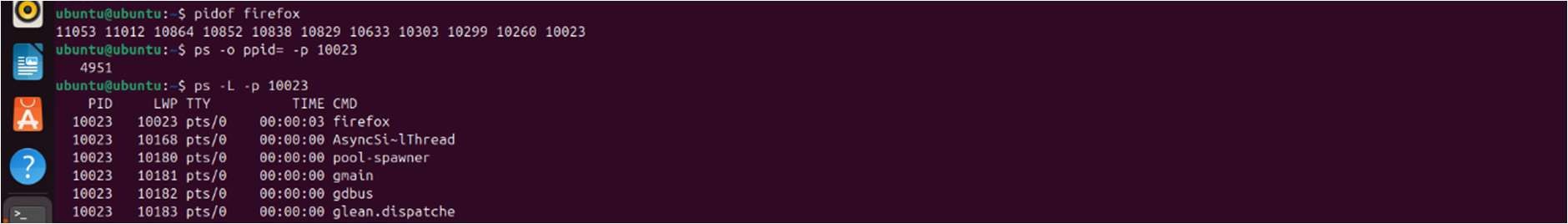
* 1. TID of the associated processes

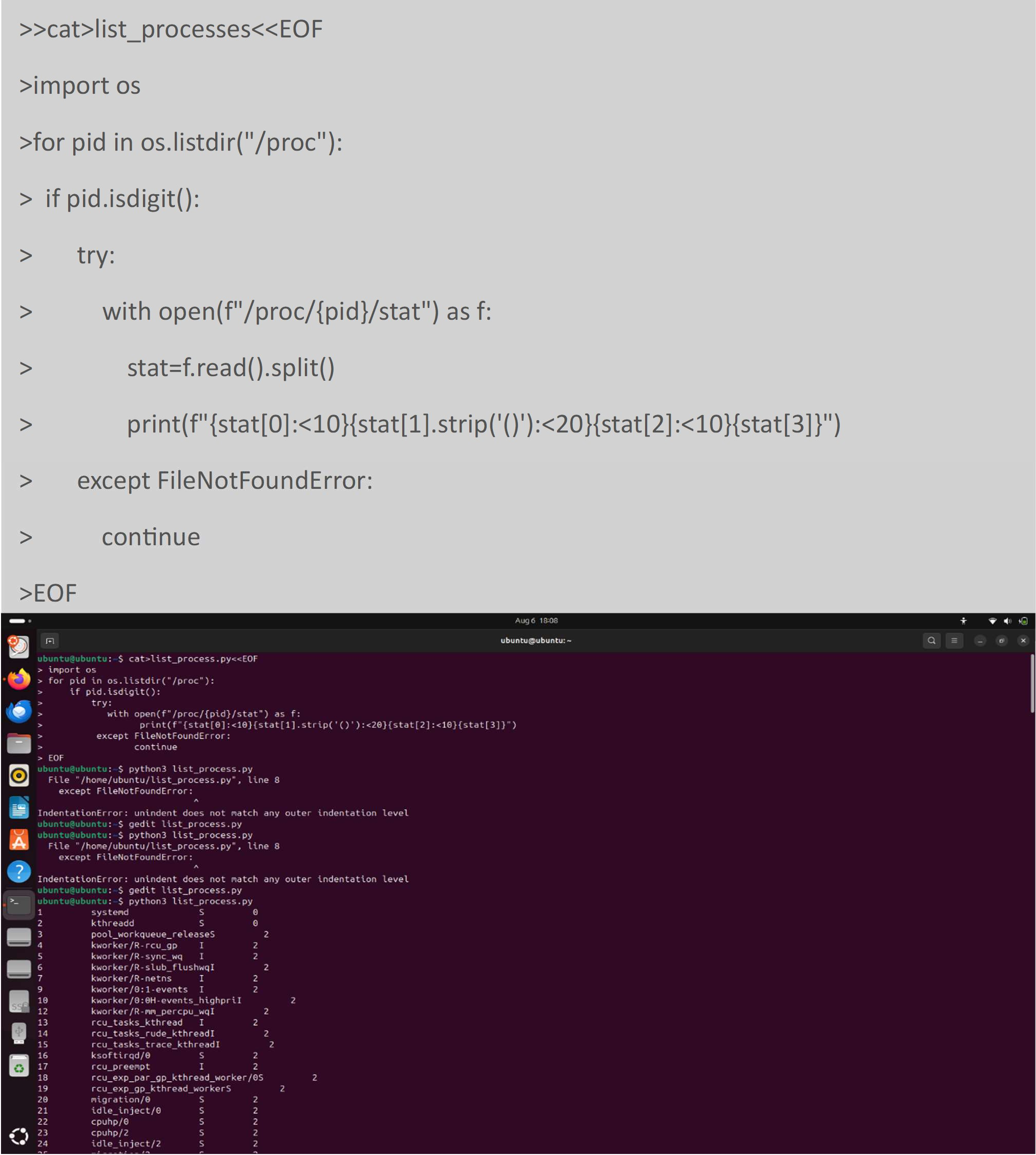
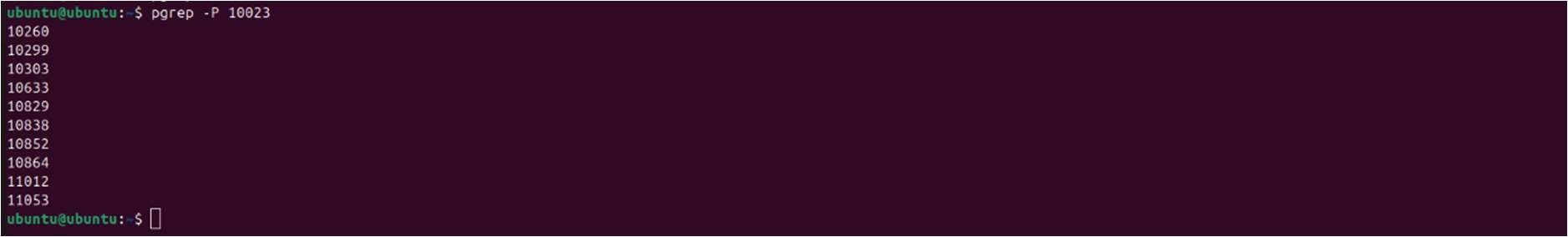
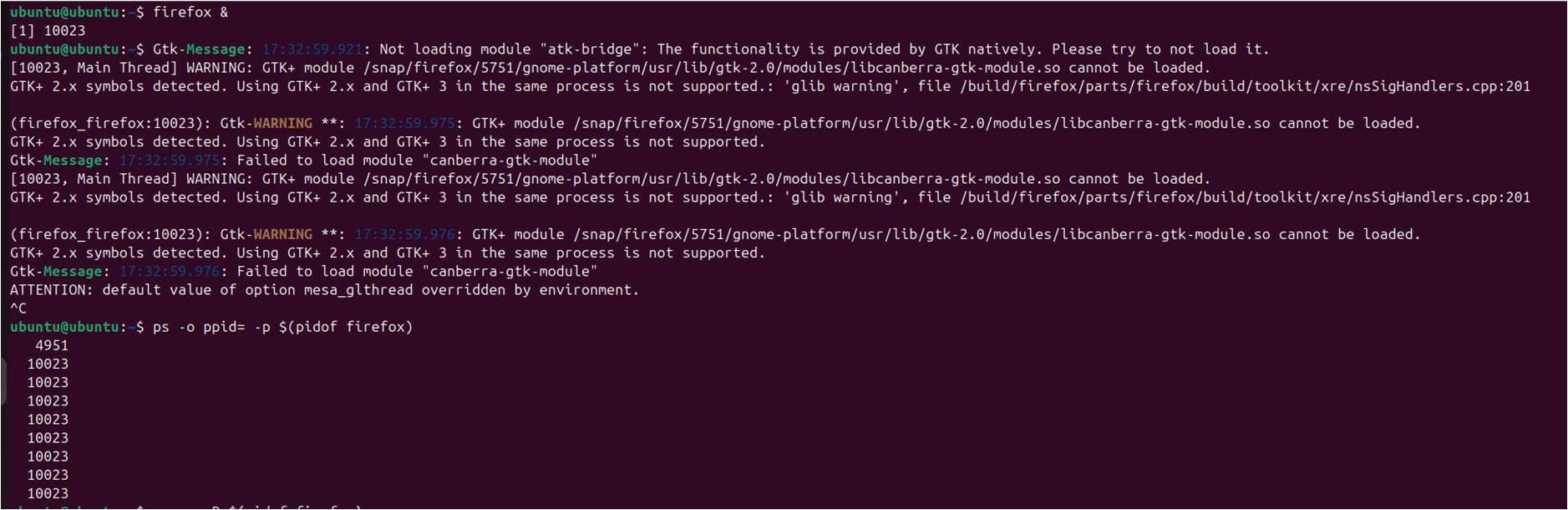
>>ps -L -p $(pidof firefox)

* 1. Kill a child process

>>kill10260

* 1. Kill the parent process

>>kill 10023



1. Using CAT command, write a program in Python to get a list of all the running processes.

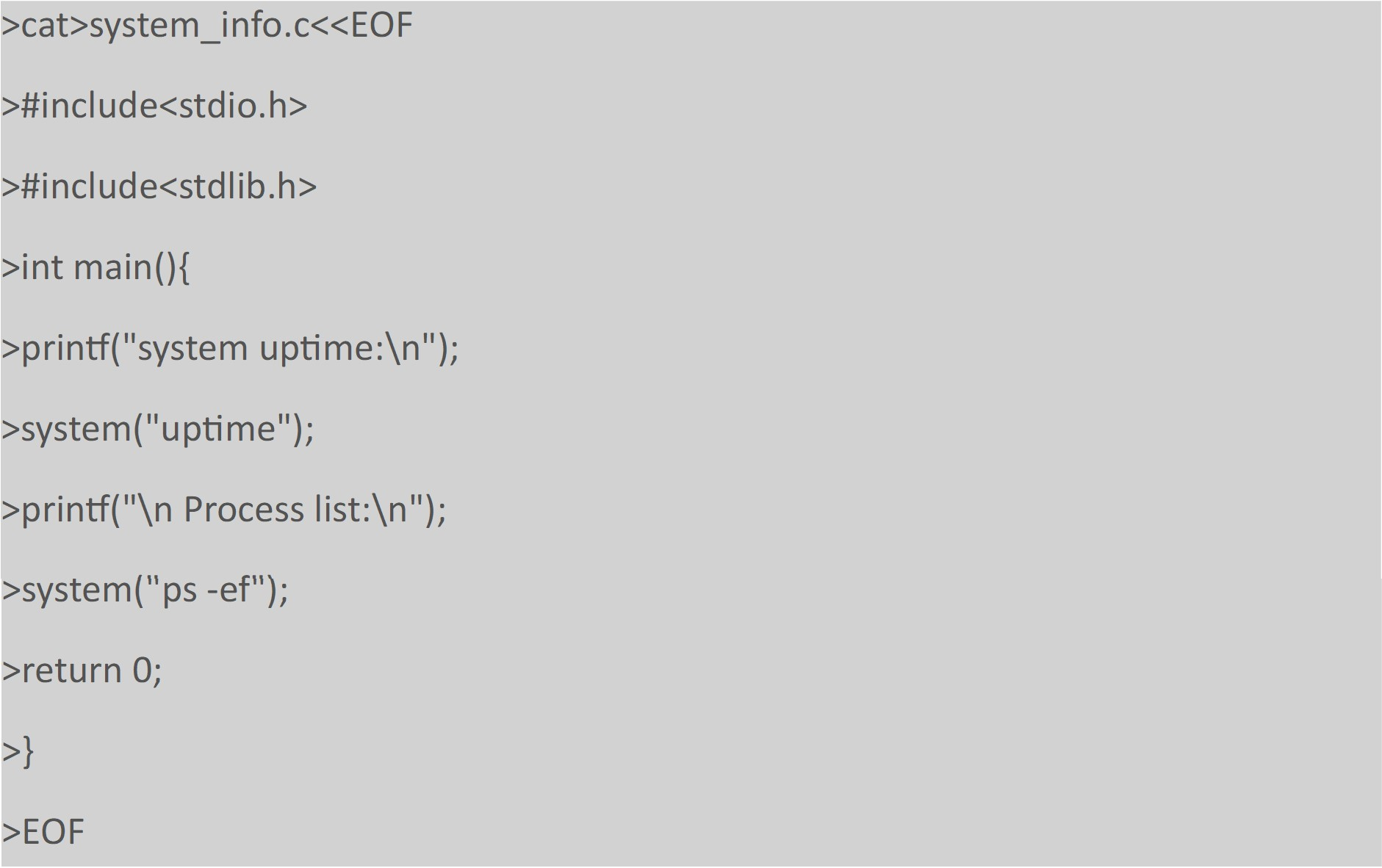
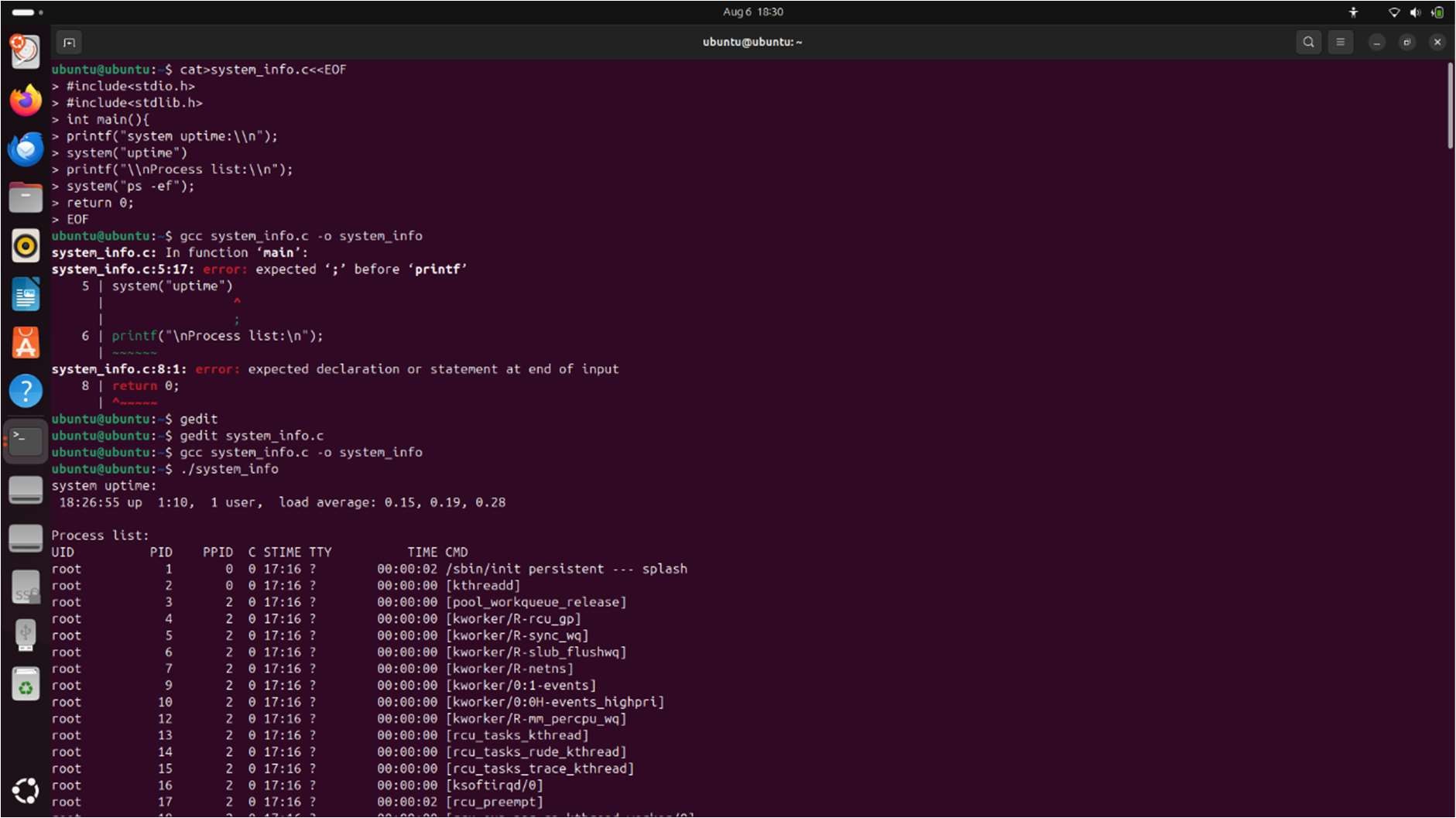
>>cat>list\_processes<<EOF

>import os

>for pid in os.listdir("/proc"):

* if pid.isdigit():
* try:
* with open(f"/proc/{pid}/stat") as f:
* stat=f.read().split()
* print(f"{stat[0]:<10}{stat[1].strip('()'):<20}{stat[2]:<10}{stat[3]}")
* except FileNotFoundError:
* continue

>EOF



1. Using CAT command, write a program in C to get the system and process information.

>cat>system\_info.c<<EOF

>#include<stdio.h>

>#include<stdlib.h>

>int main(){

>printf("system uptime:\n");

>system("uptime");

>printf("\n Process list:\n");

>system("ps -ef");

>return

ASSIGNMENT 4

FCFS

import matplotlib.pyplot as plt print("FIRST COME FIRST SERVE SCHEDULING")

def calculate\_metrics(processes, n, bt): wt = [0] \* n

tat = [0] \* n rt = [0] \* n

ct = [0] \* n ct[0] = bt[0]

for i in range(1, n):

ct[i] = ct[i - 1] + bt[i]

for i in range(n):

tat[i] = ct[i] - processes[i][1]

wt[i] = tat[i] - processes[i][2] rt[i] = ct[i]

avg\_tat = sum(tat) / n avg\_wt = sum(wt) / n avg\_rt = sum(rt) / n

return ct, tat, wt, rt, avg\_tat, avg\_wt, avg\_rt

def main():

n = int(input("Enter the number of processes: ")) processes = []

for i in range(n):

arrival\_time = int(input(f"Enter arrival time for process {i +

1}: ")) "))

burst\_time = int(input(f"Enter burst time for process {i + 1}: processes.append((i + 1, arrival\_time, burst\_time))

processes.sort(key=lambda x: x[1])

burst\_time\_list = [process[2] for process in processes]

completion\_time, turnaround\_time, waiting\_time, response\_time, avg\_turnaround\_time, avg\_waiting\_time, avg\_response\_time = calculate\_metrics(processes, n, burst\_time\_list)

print("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\tResponse Time")

for i in range(n): print(f"P{i +

1}\t{processes[i][1]}\t\t{processes[i][2]}\t\t{completion\_time[i]}\t\t{ turnaround\_time[i]}\t\t{waiting\_time[i]}\t\t{response\_time[i]}")

print(f"\nAverage Turnaround Time: {avg\_turnaround\_time}") print(f"Average Waiting Time: {avg\_waiting\_time}") print(f"Average Response Time: {avg\_response\_time}")

'''plt.figure(figsize=(10, 4)) for i in range(n):

plt.barh(f'P{processes[i][0]}', completion\_time[i] - processes[i][1], left=processes[i][1], color='tab:blue')

plt.xlabel('Time') plt.ylabel('Processes') plt.title('Gantt Chart (FCFS)')

plt.grid(True, linestyle='--', alpha=0.6) plt.show()'''

plt.figure(figsize=(10, 4)) plt.title("Gantt Chart - FCFS Scheduling")

for i in range(n):

plt.barh(y=0, width=burst\_time\_list[i], left=completion\_time[i]

- burst\_time\_list[i], height=0.5)

plt.yticks([]) plt.xlabel("Time")

plt.show() main()

OUTPUT

FIRST COME FIRST SERVE SCHEDULING

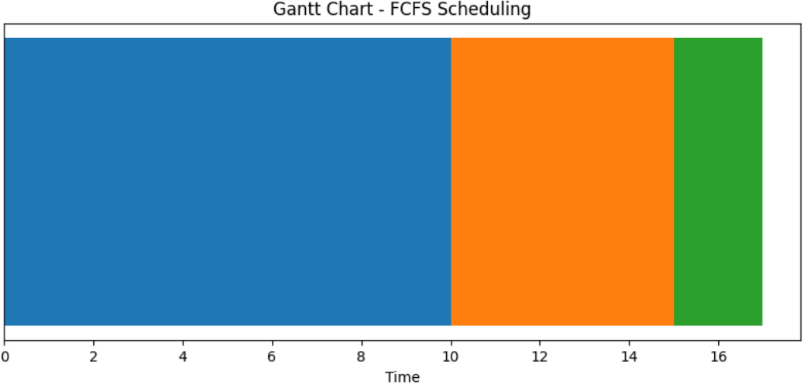
Enter the number of processes: 3 Enter arrival time for process 1: 0 Enter burst time for process 1: 10 Enter arrival time for process 2: 1 Enter burst time for process 2: 5 Enter arrival time for process 3: 3 Enter burst time for process 3: 2

Process Arrival Time Burst Time Completion Time Turnaround Time Waiting Time Response Time

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P1 0 | 10 | 10 | 10 | 0 | 10 |
| P2 1 | 5 | 15 | 14 | 9 | 15 |
| P3 3 | 2 | 17 | 14 | 12 | 17 |

Average Turnaround Time: 12.666666666666666 Average Waiting Time: 7.0

Average Response Time: 14.0



# SJF

import matplotlib.pyplot as plt

def findWaitingTime(processes, n, wt, ct, gantt\_chart): rt = [0] \* n

for i in range(n):

rt[i] = processes[i][2]

complete = 0

t = 0

min\_rt = float('inf') short = 0

check = False

while complete != n: for j in range(n):

if processes[j][1] <= t and rt[j] < min\_rt and rt[j] > 0: min\_rt = rt[j]

short = j check = True

if not check: t += 1

continue

rt[short] -= 1 min\_rt = rt[short]

if min\_rt == 0:

min\_rt = float('inf')

complete += 1 check = False ct[short] = t + 1

wt[short] = ct[short] - processes[short][1] - processes[short][2]

if wt[short] < 0: wt[short] = 0

gantt\_chart.append((t, t + 1, processes[short][0])) t += 1

def findTurnAroundTime(processes, n, wt, tat): for i in range(n):

tat[i] = processes[i][2] + wt[i]

def findavgTime(processes, n): wt = [0] \* n

tat = [0] \* n ct = [0] \* n gantt\_chart = []

findWaitingTime(processes, n, wt, ct, gantt\_chart) findTurnAroundTime(processes, n, wt, tat)

print("Processes\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n): total\_wt += wt[i] total\_tat += tat[i]

print(f" {processes[i][0]}\t\t{processes[i][1]}\t\t{processes[ i][2]}\t\t{ct[i]}\t\t{wt[i]}\t\t{tat[i]}")

avg\_wt = total\_wt / n avg\_tat = total\_tat / n

print(f"\nAverage Waiting Time: {avg\_wt:.5f}") print(f"Average Turnaround Time: {avg\_tat:.5f}")

plot\_gantt\_chart(gantt\_chart, n) def plot\_gantt\_chart(gantt\_chart, n):

fig, gnt = plt.subplots()

gnt.set\_ylim(0, n + 1) gnt.set\_xlim(0, gantt\_chart[-1][1] + 1)

gnt.set\_xlabel('Time') gnt.set\_ylabel('Processes')

unique\_processes = list(set(process for \_, \_, process in gantt\_chart))

unique\_processes.sort()

gnt.set\_yticks(unique\_processes) gnt.set\_yticklabels([f'P{process}' for process in

unique\_processes])

for i in range(len(gantt\_chart)): start, end, process = gantt\_chart[i]

gnt.broken\_barh([(start, end - start)], (process - 0.4, 0.8), facecolors=('tab:blue', 'tab:orange', 'tab:green', 'tab:red')[process % 4])

plt.title('Gantt Chart') plt.show()

def main():

n = int(input("Enter the number of processes: ")) processes = []

for i in range(n):

arrival\_time = int(input(f"Enter arrival time for process {i +

1}: ")) "))

burst\_time = int(input(f"Enter burst time for process {i + 1}: processes.append([i + 1, arrival\_time, burst\_time])

processes.sort(key=lambda x: x[1]) findavgTime(processes, n)

main()

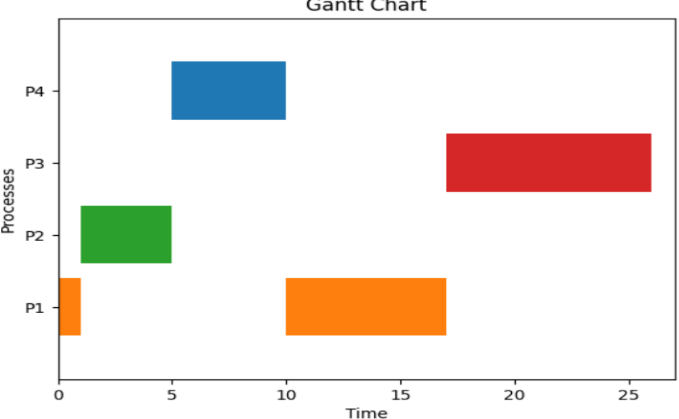
# OUTPUT

Enter the number of processes: 4 Enter arrival time for process 1: 0 Enter burst time for process 1: 8 Enter arrival time for process 2: 1 Enter burst time for process 2: 4 Enter arrival time for process 3: 2 Enter burst time for process 3: 9 Enter arrival time for process 4: 3 Enter burst time for process 4: 5

Processes Arrival Time Burst Time Completion Time Waiting Time Turn-Around Time

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 8 | 17 | 9 | 17 |
| 2 | 1 | 4 | 5 | 0 | 4 |
| 3 | 2 | 9 | 26 | 15 | 24 |
| 4 | 3 | 5 | 10 | 2 | 7 |

Average Waiting Time: 6.50000 Average Turnaround Time: 13.00000



# SRTF

def calculate\_metrics(processes): n = len(processes)

wt, tat, ct, rt = [0] \* n, [0] \* n, [0] \* n, [0] \* n

completed = [False] \* n time = 0

for i in range(n): min\_bt = float('inf') shortest = None

for j in range(n):

if not completed[j] and processes[j][1] <= time: if processes[j][2] < min\_bt:

min\_bt = processes[j][2] shortest = j

if shortest is None:

for j in range(n):

if not completed[j]:

time = processes[j][1] break

completed[shortest] = True

rt[shortest] = time - processes[shortest][1] time += processes[shortest][2]

ct[shortest] = time

tat[shortest] = ct[shortest] - processes[shortest][1] wt[shortest] = tat[shortest] - processes[shortest][2]

avg\_tat = sum(tat) / n avg\_wt = sum(wt) / n avg\_rt = sum(rt) / n

return ct, tat, wt, rt, avg\_tat, avg\_wt, avg\_rt def plot\_gantt\_chart(processes, ct):

fig, ax = plt.subplots(figsize=(10, 4))

sorted\_processes = sorted(processes, key=lambda x: x[2]) for i, (pid, at, bt) in enumerate(sorted\_processes):

start\_time = max(at, ct[i - 1] if i > 0 else 0) ax.barh(f'P{pid}', left=start\_time, width=bt, label=f'P{pid}') ct[i] = start\_time + bt

ax.set\_xlabel('Time')

ax.set\_yticks([f'P{p[0]}' for p in sorted\_processes]) ax.set\_title('Gantt Chart')

plt.legend(loc='upper right') plt.show()

def main():

n = int(input("Enter the number of processes: "))

processes = [(i + 1, int(input(f"Arrival time for P{i + 1}: ")), int(input(f"Burst time for P{i + 1}: "))) for i in range(n)]

ct, tat, wt, rt, avg\_tat, avg\_wt, avg\_rt = calculate\_metrics(processes)

print("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\tResponse Time")

for i in range(n):

pid, at, bt = processes[i] print(f"P{pid}\t{at}\t\t{bt}\t\t{ct[i]}\t\t{tat[i]}\t\t{wt[i]}\

t\t{rt[i]}")

print(f"\nAverage Turnaround Time: {avg\_tat}") print(f"Average Waiting Time: {avg\_wt}") print(f"Average Response Time: {avg\_rt}") plot\_gantt\_chart(processes, ct)

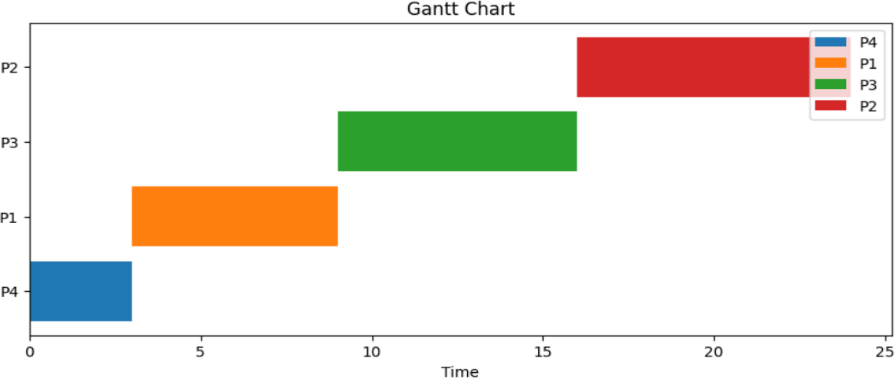
main()

# OUTPUT

Enter the number of processes: 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Arrival time for P1: Burst time for P1: 6 Arrival time for P2: Burst time for P2: 8 Arrival time for P3: Burst time for P3: 7 Arrival time for P4: Burst time for P4: 3 | | | 0  0  0  0 |  |  |  |
| Process  P1 | Arrival Waiting 0 | Time Time | Burst Time Response Time 6 | Completion  9 | Time Turnaround  9 | Time |
| 3 |
| P2 | 0 | 3 | 8 | 24 | 24 | 16 |
| P3 | 0 | 16 | 7 | 16 | 16 | 9 |
| P4 | 0 | 9 | 3 | 3 | 3 | 0 |
| 0 | | | |  |  |  |
| Average Turnaround Time: 13.0 | | | |  |  |  |

Average Waiting Time: 7.0 Average Response Time: 7.0



# ROUND ROBIN

import matplotlib.pyplot as plt

def round\_robin(processes, burst\_time, arrival\_time, quantum):

n = len(processes) waiting\_time = [0] \* n turnaround\_time = [0] \* n

remaining\_time = burst\_time.copy() completion\_time = [0] \* n

time = 0 timeline = [] gantt\_chart = []

while any(remaining\_time): for i in range(n):

if remaining\_time[i] > 0 and arrival\_time[i] <= time: if remaining\_time[i] > quantum:

timeline.append(processes[i]) gantt\_chart.append(quantum) time += quantum remaining\_time[i] -= quantum

else:

timeline.append(processes[i]) gantt\_chart.append(remaining\_time[i]) time += remaining\_time[i] completion\_time[i] = time waiting\_time[i] = time - burst\_time[i] -

arrival\_time[i]

remaining\_time[i] = 0

for i in range(n):

turnaround\_time[i] = completion\_time[i] - arrival\_time[i]

avg\_waiting\_time = sum(waiting\_time) / n avg\_turnaround\_time = sum(turnaround\_time) / n

return avg\_waiting\_time, avg\_turnaround\_time, timeline, gantt\_chart, completion\_time, waiting\_time, turnaround\_time

def main():

n = int(input("Enter the number of processes: ")) processes = [f"P{i}" for i in range(n)]

arrival\_time = [int(input(f"Enter arrival time for {processes[i]}: ")) for i in range(n)]

burst\_time = [int(input(f"Enter burst time for {processes[i]}: ")) for i in range(n)]

quantum = int(input("Enter time quantum: "))

avg\_waiting\_time, avg\_turnaround\_time, timeline, gantt\_chart, completion\_time, waiting\_time, turnaround\_time = round\_robin(processes, burst\_time, arrival\_time, quantum)

print("Process\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time")

for i in range(n): print(f"{processes[i]}\t\t{arrival\_time[i]}\t\t{burst\_time[i]}\

t\t{completion\_time[i]}\t\t{waiting\_time[i]}\t\t{turnaround\_time[i]}")

print(f"\nAverage Waiting Time: {avg\_waiting\_time:.2f}") print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

plt.figure(figsize=(8, 3))

plt.bar(range(len(gantt\_chart)), gantt\_chart, tick\_label=timeline) plt.xlabel('Time')

plt.ylabel('Burst Time')

plt.title(f'Round Robin Gantt Chart (Time Quantum = {quantum})') plt.show()

quantum\_values = list(range(1, 11)) avg\_waiting\_times = []

for quantum\_value in quantum\_values:

avg\_waiting\_time , \_, \_, \_, \_, \_, \_ = round\_robin(processes, burst\_time, arrival\_time, quantum\_value)

avg\_waiting\_times.append(avg\_waiting\_time)

plt.figure(figsize=(8, 4))

plt.plot(quantum\_values, avg\_waiting\_times, marker='o') plt.xlabel('Time Quantum')

plt.ylabel('Average Waiting Time')

plt.title('Average Waiting Time (AWT) vs. Time Quantum') plt.grid(True)

plt.show() avg\_turnaround\_times = []

for quantum\_value in quantum\_values:

\_, avg\_turnaround\_time, \_, \_, \_, \_, \_ = round\_robin(processes, burst\_time, arrival\_time, quantum\_value)

avg\_turnaround\_times.append(avg\_turnaround\_time)

plt.figure(figsize=(8, 4))

plt.plot(quantum\_values, avg\_turnaround\_times, marker='x', color='orange')

plt.xlabel('Time Quantum') plt.ylabel('Average Turnaround Time')

plt.title('Average Turnaround Time (ATAT) vs. Time Quantum') plt.grid(True)

plt.show()

main()

# OUTPUT

Enter the number of processes: 4 Enter arrival time for P0: 0 Enter arrival time for P1: 1 Enter arrival time for P2: 2 Enter arrival time for P3: 4 Enter burst time for P0: 10 Enter burst time for P1: 5 Enter burst time for P2: 2 Enter burst time for P3: 8 Enter time quantum: 2

Process Arrival Time Burst Time Completion Time Waiting Time Turnaround Time

P0 0 10 25 15

25

P1 1 5 17 11

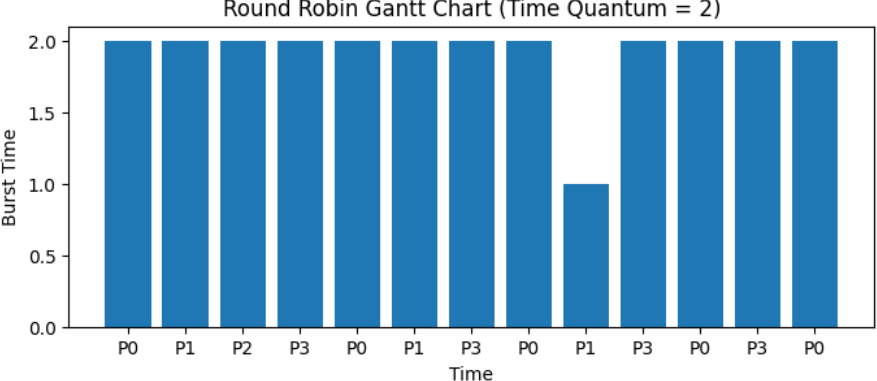
16

P2 2 2 6 2

4

P3 4 8 23 11

19

Average Waiting Time: 9.75 Average Turnaround Time: 16.00

ASSIGNMENT 8

First Fit

from tabulate import tabulate

PDTm = [

    ["memory block no", "size", "status"],

    [0, 200, 1],

    [1, 100, 0],

    [10, 500, 0],

    [11, 200, 0],

    [100, 300, 0],

    [101, 600, 0],

]

print("Initial Memory Table (PDTm):")

print(tabulate(PDTm, headers="firstrow"))

print()

%pip install tabulate

PDTp = [

    ["process", "size"],

    ["P1", 210],

    ["P2", 410],

    ["P3", 110],

    ["P4", 420],

]

print("Process Table (PDTp):")

print(tabulate(PDTp, headers="firstrow"))

print()

unallocated\_processes = []

allocated\_mapping = {}

internal\_frag = []

for process\_index in range(1, len(PDTp)):

    process = PDTp[process\_index]

    process\_allocated = False

    for i in range(1, len(PDTm)):

        if PDTm[i][2] == 0 and PDTm[i][1] >= process[1]:

            PDTm[i][2] = 1

            allocated\_mapping[PDTm[i][0]] = process[0]

            frag = PDTm[i][1] - process[1]

            if frag > 0:

                internal\_frag.append([PDTm[i][0], frag])

            process\_allocated = True

            break

    if not process\_allocated:

        unallocated\_processes.append(process)

print("\nCurrent PDTm:")

print(tabulate(PDTm, headers="firstrow"))

print("\nUnallocated processes:")

if unallocated\_processes:

    print(tabulate(unallocated\_processes, headers=["process", "size"]))

else:

    print("None — all processes were allocated.")

total\_internal\_frag = sum(frag[1] for frag in internal\_frag)

print(f"\nTotal Internal Fragmentation: {total\_internal\_frag}")

external\_frag = 0

if unallocated\_processes:

    largest\_unalloc\_size = max(p[1] for p in unallocated\_processes)

    for i in range(1, len(PDTm)):

        if PDTm[i][2] == 0 and PDTm[i][1] < largest\_unalloc\_size:

            external\_frag += PDTm[i][1]

print(f"\nExternal Fragmentation: {external\_frag}")

Initial Memory Table (PDTm):

memory block no size status

----------------- ------ --------

0 200 1

1 100 0

10 500 0

11 200 0

100 300 0

101 600 0

Process Table (PDTp):

process size

--------- ------

P1 210

P2 410

P3 110

P4 420

Current PDTm:

memory block no size status

----------------- ------ --------

0 200 1

1 100 0

10 500 1

11 200 1

100 300 0

101 600 1

Unallocated processes:

process size

--------- ------

P4 420

Total Internal Fragmentation: 570

External Fragmentation: 400

Best Fit

from tabulate import tabulate

PDTm = [

    ["memory block no", "size", "status"],

    [0, 200, 1],

    [1, 100, 0],

    [10, 500, 0],

    [11, 200, 0],

    [100, 300, 0],

    [101, 600, 0],

]

print("Initial Memory Table (PDTm):")

print(tabulate(PDTm, headers="firstrow"))

print()

PDTp = [

    ["process", "size"],

    ["P1", 210],

    ["P2", 410],

    ["P3", 110],

    ["P4", 420],

]

print("Process Table (PDTp):")

print(tabulate(PDTp, headers="firstrow"))

print()

unallocated\_processes = []

allocated\_mapping = {}

internal\_frag = []

for process\_index in range(1, len(PDTp)):

    process = PDTp[process\_index]

    process\_size = process[1]

    process\_allocated = False

    best\_index = -1

    best\_size = float('inf')

    for i in range(1, len(PDTm)):

        if PDTm[i][2] == 0 and PDTm[i][1] >= process\_size:

            if PDTm[i][1] < best\_size:

                best\_size = PDTm[i][1]

                best\_index = i

    if best\_index != -1:

        PDTm[best\_index][2] = 1

        allocated\_mapping[PDTm[best\_index][0]] = process[0]

        frag = PDTm[best\_index][1] - process\_size

        if frag > 0:

            internal\_frag.append([PDTm[best\_index][0], frag])

        process\_allocated = True

    else:

        unallocated\_processes.append(process)

print("\nCurrent PDTm:")

print(tabulate(PDTm, headers="firstrow"))

print("\nProcess to Memory Block Mapping:")

if allocated\_mapping:

    mapping\_table = [[block, allocated\_mapping[block]] for block in allocated\_mapping]

    print(tabulate(mapping\_table, headers=["Block No", "Allocated Process"]))

else:

    print("No allocations were made.")

print("\nUnallocated Processes:")

if unallocated\_processes:

    print(tabulate(unallocated\_processes, headers=["Process", "Size"]))

else:

    print("None — all processes were allocated.")

print("\nInternal Fragmentation per Block:")

if internal\_frag:

    print(tabulate(internal\_frag, headers=["Block No", "Internal Fragmentation"]))

else:

    print("None")

total\_internal\_frag = sum(frag[1] for frag in internal\_frag)

print(f"\nTotal Internal Fragmentation: {total\_internal\_frag}")

external\_frag = 0

if unallocated\_processes:

    largest\_unalloc\_size = max(p[1] for p in unallocated\_processes)

    for i in range(1, len(PDTm)):

        if PDTm[i][2] == 0 and PDTm[i][1] < largest\_unalloc\_size:

            external\_frag += PDTm[i][1]

print(f"\nExternal Fragmentation: {external\_frag}")

Initial Memory Table (PDTm):

memory block no size status

----------------- ------ --------

0 200 1

1 100 0

10 500 0

11 200 0

100 300 0

101 600 0

Process Table (PDTp):

process size

--------- ------

P1 210

P2 410

P3 110

P4 420

Memory allocation attempt complete (BEST-FIT).

Current PDTm:

memory block no size status

----------------- ------ --------

0 200 1

1 100 0

10 500 1

11 200 1

100 300 1

101 600 1

Process to Memory Block Mapping:

Block No Allocated Process

---------- -------------------

100 P1

10 P2

11 P3

101 P4

Unallocated Processes:

None — all processes were allocated.

Internal Fragmentation per Block:

Block No Internal Fragmentation

---------- ------------------------

100 90

10 90

11 90

101 180

Total Internal Fragmentation: 450

External Fragmentation: 0

from tabulate import tabulate

PDTm = [

    ["memory block no", "size", "status"],

    [0, 200, 1],

    [1, 100, 0],

    [10, 500, 0],

    [11, 200, 0],

    [100, 300, 0],

    [101, 600, 0],

]

print("Initial Memory Table (PDTm):")

print(tabulate(PDTm, headers="firstrow"))

print()

PDTp = [

    ["process", "size"],

    ["P1", 210],

    ["P2", 410],

    ["P3", 110],

    ["P4", 420],

]

Worse Fit

print("Process Table (PDTp):")

print(tabulate(PDTp, headers="firstrow"))

print()

unallocated\_processes = []

allocated\_mapping = {}

internal\_frag = []

for process\_index in range(1, len(PDTp)):

    process = PDTp[process\_index]

    process\_size = process[1]

    process\_allocated = False

    worst\_index = -1

    worst\_size = -1

    for i in range(1, len(PDTm)):

        if PDTm[i][2] == 0 and PDTm[i][1] >= process\_size:

            if PDTm[i][1] > worst\_size:

                worst\_size = PDTm[i][1]

                worst\_index = i

    if worst\_index != -1:

        PDTm[worst\_index][2] = 1

        allocated\_mapping[PDTm[worst\_index][0]] = process[0]

        frag = PDTm[worst\_index][1] - process\_size

        if frag > 0:

            internal\_frag.append([PDTm[worst\_index][0], frag])

        process\_allocated = True

    else:

        unallocated\_processes.append(process)

print("\nCurrent PDTm:")

print(tabulate(PDTm, headers="firstrow"))

print("\nUnallocated Processes:")

if unallocated\_processes:

    print(tabulate(unallocated\_processes, headers=["Process", "Size"]))

else:

    print("None — all processes were allocated.")

total\_internal\_frag = sum(frag[1] for frag in internal\_frag)

print(f"\nTotal Internal Fragmentation: {total\_internal\_frag}")

external\_frag = 0

if unallocated\_processes:

    largest\_unalloc\_size = max(p[1] for p in unallocated\_processes)

    for i in range(1, len(PDTm)):

        if PDTm[i][2] == 0 and PDTm[i][1] < largest\_unalloc\_size:

            external\_frag += PDTm[i][1]

print(f"\nExternal Fragmentation: {external\_frag}")

Initial Memory Table (PDTm):

memory block no size status

----------------- ------ --------

0 200 1

1 100 0

10 500 0

11 200 0

100 300 0

101 600 0

Process Table (PDTp):

process size

--------- ------

P1 210

P2 410

P3 110

P4 420

Current PDTm:

memory block no size status

----------------- ------ --------

0 200 1

1 100 0

10 500 1

11 200 0

100 300 1

101 600 1

Unallocated Processes:

Process Size

--------- ------

P4 420

Total Internal Fragmentation: 670

External Fragmentation: 300