Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Operating Systems (ENCS3390)

Project Report

Prepared by:

Student's Name: Abdalkarim Eiss Student's Id: 1200015

Student's Name: Mahmoud Khatib Student's Id: 1200275

Instructor: Dr. Yazan Abu Farha

Section: 1

Date: 12/2/2023

Table of Contents

List of Figures

Code

The below codes are implemented by using PyCharm software in python language.

Process Class

```
class Process:
    def __init__(self, pid, arrival_time):
        self.__pid = pid
        self.__arrivalTime = arrival_time
        self.__bursts = []
        self.__workingTime = 0
        self.__finishTime = 0
        self.__count = 0
        self.__predictTime = 20 # tau(n)

    def get_pid(self):
        return self.__pid

    def set_pid(self, pid):
        self.__pid = pid

    def get_arrival_time(self):
        return self.__arrivalTime

    def set_arrival_time(self, arrival_time):
        self.__arrivalTime = arrival_time

    def get_bursts(self) -> list:
        return self.__bursts

    def set_bursts(self, bursts: list):
        self.__bursts = bursts

    def get_working_time(self):
        return self.__workingTime

    def set_working_time(self, t):
        self.__workingTime = t

    def get_finish_time(self):
        return self.__finishTime

    def set_finish_time(self, finish_time):
        self.__finishTime = finish_time

    def get_count(self):
        return self.__count

    def get_predict_time(self):
        return self.__predictTime
```

```
    def set_predict_time(self, predict_time):
        self.__predictTime = predict_time

    def set_count(self, count):
        self.__count = count

    def __lt__(self, other):
        return self.__arrivalTime < other.__arrivalTime

    def __repr__(self):
        process_str = '{}\t{}\t{}'.format(self.__pid, self.__arrivalTime,
"\t".join(str(burst) for burst in self.__bursts))
        return process_str

    def __str__(self):
        return f'pid: {self.__pid}\nArrival Time:
{self.__arrivalTime}\nBursts: {self.__bursts}\n'
```

Main Code

```
import sys
from process import *
from random import randint
from colorama import Fore


def workload_generator(max_processes, max_arrive_time, max_cpu_bursts,
min_io, max_io, min_cpu, max_cpu):
    processes = []

    for pid in range(max_processes):
        arriveTime = randint(0, max_arrive_time)
        cpuBurstNum = randint(1, max_cpu_bursts)
        p = Process(pid, arriveTime)

        for burst in range(cpuBurstNum):
            p.get_bursts().append(randint(min_cpu, max_cpu))

            if burst < cpuBurstNum-1:
                p.get_bursts().append(randint(min_io, max_io))
        processes.append(p)

    with open('process.txt', "w") as f:
        for p in processes:
            f.write(p.__repr__() + '\n')
            print(p)

    print('Process generation is complete')


def read_from_file(file_name):
    jobQueue = []

    with open(file_name, 'r') as f:
        for i in f:
            print(i)
            line = i.split()
            p = Process(int(line[0]), int(line[1]))
            p.set_bursts([int(burst) for burst in line[2:]])
            jobQueue.append(p)

    jobQueue.sort()
    return jobQueue


# returns the process that has the minimum wait time left in higher priority
queues
# returns max size(maxInt) if there is no processes in IO that have a higher
priority than specified
def find_min(d: dict, queue_priority=5):
    processes = [p for p in d.keys() if d[p] < queue_priority]
    if processes:
```

```
        bursts = [p.get_bursts() for p in processes]
        return min(bursts, key=lambda x: x[0])[0]
    return sys.maxsize


def elapse_io_round(q1, q2, q3, q4, io: dict[Process], curr_time, prev_time):
    for p in io.copy():
        p.get_bursts()[0] -= (curr_time - prev_time)
        if p.get_bursts()[0] <= 0:
            print(f'P{p.get_pid()} has finished its IO burst at time
{curr_time + p.get_bursts().pop(0)}')

            if io.get(p) == 1:
                q1.append(p)
            elif io.get(p) == 2:
                q2.append(p)
            elif io.get(p) == 3:
                q3.append(p)
            elif io.get(p) == 4:
                q4.append(p)
            del io[p]


def avg_waiting_time(finished_proc: list[Process]):
    totalWaitTime = 0
    for i in finished_proc:
        totalWaitTime += (i.get_finish_time() - i.get_arrival_time()) -
i.get_working_time()

    return float(totalWaitTime / len(finished_proc))


def start_simulation(job_queue: list[Process], tq1, tq2, alpha):
    q1 = []
    q2 = []
    q3 = []
    q4 = []
    ganttchart = {}
    io = {}
    finishedProcesses = []
    idleTime = 0    # to calculate the utilization

    currTime = job_queue[0].get_arrival_time()
    prevTime = currTime
    i = 0
    boolean_1 = False
    boolean_2 = False

    while True:
        print(Fore.YELLOW)
        # this loop will not be executed after all the processes has entered
Queue 1
        while i < len(job_queue) and job_queue[i].get_arrival_time() <=
currTime:
            print(f'process P{job_queue[i].get_pid()} entered Queue 1 at
time: {job_queue[i].get_arrival_time()}')
            print(job_queue[i])
```

```
            q1.append(job_queue[i])
            i += 1

        # processes in queue 1
        if len(q1) > 0:
            if q1[0].get_count() == 10:
                print(f'P{q1[0].get_pid()} has entered Queue 2 at time:
{currTime}')
                q1[0].set_count(0)
                q2.append(q1.pop(0))
                if len(q1) == 0:
                    continue

            if boolean_1:
                p = q1.pop(0)
                q1.append(p)
            print(f'P{q1[0].get_pid()} in Queue 1, has started its CPU turn
at time: {currTime}')

            q1[0].get_bursts()[0] -= tq1
            prevTime = currTime

            if q1[0].get_bursts()[0] <= 0:
                currTime += tq1 + q1[0].get_bursts().pop(0)

                q1[0].set_predict_time((alpha * (currTime - prevTime)) + ((1
- alpha) * q1[0].get_predict_time()))
                q1[0].set_working_time(q1[0].get_working_time() + currTime-
prevTime)
                ganttchart[range(prevTime, currTime)] = f'P{q1[0].get_pid()}'
                elapse_io_round(q1, q2, q3, q4, io, currTime, prevTime)
                q1[0].set_count(0)  # reset the counter for that process

                if len(q1[0].get_bursts()) > 0:
                    print(f'process P{q1[0].get_pid()} has entered IO Device
at time: {currTime}')
                    io[q1.pop(0)] = 1
                else:
                    print(f'process P{q1[0].get_pid()} has finished all its
bursts at time {currTime}')
                    q1[0].set_finish_time(currTime)
                    finishedProcesses.append(q1.pop(0))    # process has
finished all its CPU bursts
                boolean_1 = False
            else:
                q1[0].set_count(q1[0].get_count() + 1)
                currTime += tq1
                q1[0].set_predict_time((alpha * (currTime - prevTime)) + ((1
- alpha) * q1[0].get_predict_time()))
                q1[0].set_working_time(q1[0].get_working_time() + currTime -
prevTime)
                ganttchart[range(prevTime, currTime)] = f'P{q1[0].get_pid()}'
                elapse_io_round(q1, q2, q3, q4, io, currTime, prevTime)
                boolean_1 = True

        elif len(q2) > 0:
            if q2[0].get_count() == 10:
```

```
                print(f'P{q2[0].get_pid()} has entered Queue 3 at time:
{currTime}')
                q2[0].set_count(0)
                q3.append(q2.pop(0))
                if len(q2) == 0:
                    continue

            if boolean_2:
                p = q2.pop(0)
                q2.append(p)
            print(f'P{q2[0].get_pid()} in Queue 2, has started its CPU turn
at time: {currTime}')
            prevTime = currTime
            if i != len(job_queue):   # there is remaining processes that
have not arrived yet
                minimum = min(tq2, find_min(io, queue_priority=2),
job_queue[i].get_arrival_time() - currTime, q2[0].get_bursts()[0])
            else:
                minimum = min(tq2, find_min(io, queue_priority=2),
q2[0].get_bursts()[0])
            currTime += minimum

            q2[0].set_predict_time((alpha * (currTime - prevTime)) + ((1 -
alpha) * q2[0].get_predict_time()))
            q2[0].set_working_time(q2[0].get_working_time() + currTime -
prevTime)

            q2[0].get_bursts()[0] -= minimum
            elapse_io_round(q1, q2, q3, q4, io, currTime, prevTime)
            ganttchart[range(prevTime, currTime)] = f'P{q2[0].get_pid()}'

            # the process has finished its current CPU burst
            if q2[0].get_bursts()[0] == 0:
                print(f'P{q2[0].get_pid()} (Queue 2) has finished its CPU
turn at time: {currTime}')
                q2[0].set_count(0)
                q2[0].get_bursts().pop(0)
                # this process has more IO and CPU bursts
                if len(q2[0].get_bursts()) > 0:
                    print(f'P{q2[0].get_pid()} has entered IO Device at time:
{currTime}')
                    io[q2.pop(0)] = 2
                else:
                    print(f'P{q2[0].get_pid()} has finished all its CPU
bursts at time: {currTime}')
                    q2[0].set_finish_time(currTime)
                    finishedProcesses.append(q2.pop(0))
                boolean_2 = False
            else:
                q2[0].set_count(q2[0].get_count() + 1)
                boolean_2 = True

        elif len(q3) > 0:
            if q3[0].get_count() == 3:
                print(f'P{q3[0].get_pid()} has entered Queue 4 at time:
{currTime}')
                q3[0].set_count(0)
```

```
                q4.append(q3.pop(0))
                if len(q3) == 0:
                    continue

            p = min(q3, key=lambda z: z.get_predict_time())
            if q3[0] != p:
                q3[0].set_count(q3[0].get_count() + 1)
                q3.remove(p)
                q3.insert(0, p)

            print(f'P{q3[0].get_pid()} in Queue 3, has started its CPU turn
at time: {currTime}')
            prevTime = currTime
            if i != len(job_queue):
                minimum = min(find_min(io, queue_priority=4),
job_queue[i].get_arrival_time() - currTime, q3[0].get_bursts()[0])
            else:
                minimum = min(find_min(io, queue_priority=4),
q3[0].get_bursts()[0])
            currTime += minimum
            q3[0].get_bursts()[0] -= minimum
            elapse_io_round(q1, q2, q3, q4, io, currTime, prevTime)
            ganttchart[range(prevTime, currTime)] = f'P{q3[0].get_pid()}'

            q3[0].set_predict_time((alpha * (currTime-prevTime)) + ((1-
alpha)*q3[0].get_predict_time()))
            q3[0].set_working_time(q3[0].get_working_time() + currTime -
prevTime)

            if q3[0].get_bursts()[0] == 0:
                print(f'P{q3[0].get_pid()} (Queue 3) has finished its CPU
turn at time: {currTime}')
                q3[0].get_bursts().pop(0)
                if len(q3[0].get_bursts()) > 0:
                    print(f'P{q3[0].get_pid()} has entered IO Device at time:
{currTime}')
                    io[q3.pop(0)] = 3
                else:
                    print(f'P{q3[0].get_pid()} has finished all its CPU
bursts at time: {currTime}')
                    q3[0].set_finish_time(currTime)
                    finishedProcesses.append(q3.pop(0))

        elif len(q4) > 0:
            print(f'P{q4[0].get_pid()} in Queue 4, has started its CPU turn
at time: {currTime}')
            prevTime = currTime
            if i != len(job_queue):
                minimum = min(find_min(io, queue_priority=4),
job_queue[i].get_arrival_time() - currTime, q4[0].get_bursts()[0])
            else:
                minimum = min(find_min(io, queue_priority=4),
q4[0].get_bursts()[0])
            currTime += minimum
            q4[0].get_bursts()[0] -= minimum
            elapse_io_round(q1, q2, q3, q4, io, currTime, prevTime)
            ganttchart[range(prevTime, currTime)] = f'P{q4[0].get_pid()}'
```

```
            q4[0].set_working_time(q4[0].get_working_time() + currTime -
prevTime)

            if q4[0].get_bursts()[0] == 0:
                print(f'P{q4[0].get_pid()} (Queue 4) has finished its CPU
turn at time: {currTime}')
                q4[0].get_bursts().pop(0)
                if len(q4[0].get_bursts()) > 0:
                    print(f'P{q4[0].get_pid()} has entered IO Device at time:
{currTime}')
                    io[q4.pop(0)] = 4
                else:
                    print(f'P{q4[0].get_pid()} has finished all its CPU
bursts at time: {currTime}')
                    q4[0].set_finish_time(currTime)
                    finishedProcesses.append(q4.pop(0))

        # should get the min in any queue
        # should increment the idleTime
        elif len(io) > 0:
            prevTime = currTime
            if i != len(job_queue):
                currTime += min(find_min(io),
job_queue[i].get_arrival_time()-currTime)
            else:
                currTime += find_min(io)
            idleTime += (currTime - prevTime)
            elapse_io_round(q1, q2, q3, q4, io, currTime, prevTime)
        else:
            break

        pause = input(Fore.RESET + 'Pause the simulation to view the Queues
results/info? (enter q to pause, anything else to continue)')
        if pause == 'q':
            print('================================')
            print('Queue 1: ', q1)
            print('Queue 2: ', q2)
            print('Queue 3: ', q3)
            print('Queue 4: ', q4)
            print('IO Buffer: ', io)
            print('================================')

    print(Fore.RESET+ str(ganttchart))
    print(Fore.CYAN+'Average Waiting time: ',
avg_waiting_time(finishedProcesses))
    print(Fore.GREEN+'CPU Utilization: ', float((currTime-idleTime) /
currTime) * 100, '%')


def main():
    print('Please choose an option!')
    jobQueue = []

    while True:
        option = int(input('1- Workload generator\t2-Enter a file\n'))
        if option == 1:
            print('Please enter the following parameters:-')
```

```
            maxProcesses = int(input('Number of process: '))
            maxArrivalTime = int(input('Maximum arrival time: '))
            maxCpuBurstNum = int(input('Maximum number of cpu bursts: '))
            minIo = int(input('Minimum IO burst duration: '))
            maxIo = int(input('Maximum IO burst duration: '))
            minCpu = int(input('Minimum CPU burst duration: '))
            maxCpu = int(input('Maximum CPU burst duration: '))

            workload_generator(maxProcesses, maxArrivalTime, maxCpuBurstNum,
minIo, maxIo, minCpu, maxCpu)
            jobQueue = read_from_file('process.txt')
        elif option == 2:
            fileName = input('Please enter a file name: ')
            jobQueue = read_from_file(fileName)
        else:
            print('This choice is not valid!!')
            continue
        break

    timeQuantum_1 = int(input('Please enter the time quantum for Queue 1: '))
    timeQuantum_2 = int(input('Please enter the time quantum for Queue 2: '))
    alpha = float(input('Please enter alpha: '))
    start_simulation(jobQueue, timeQuantum_1, timeQuantum_2, alpha)


if __name__ == '__main__':
    main()
```

Test Cases

Test Case One

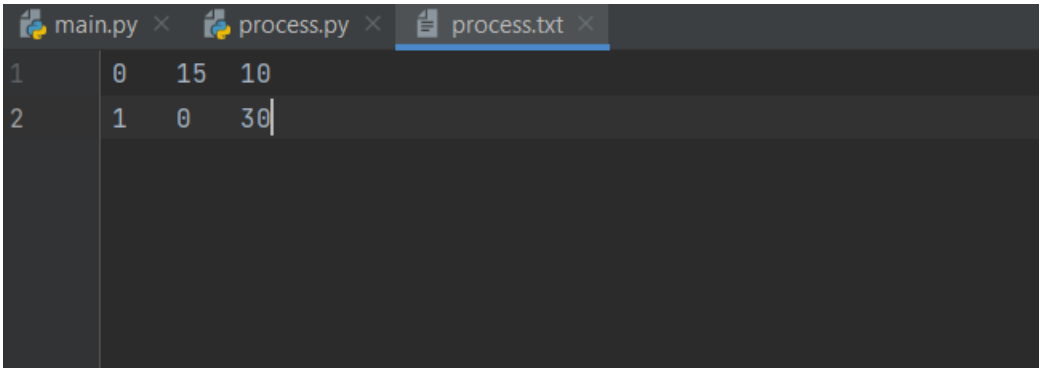

Figure 1: The First Test Case

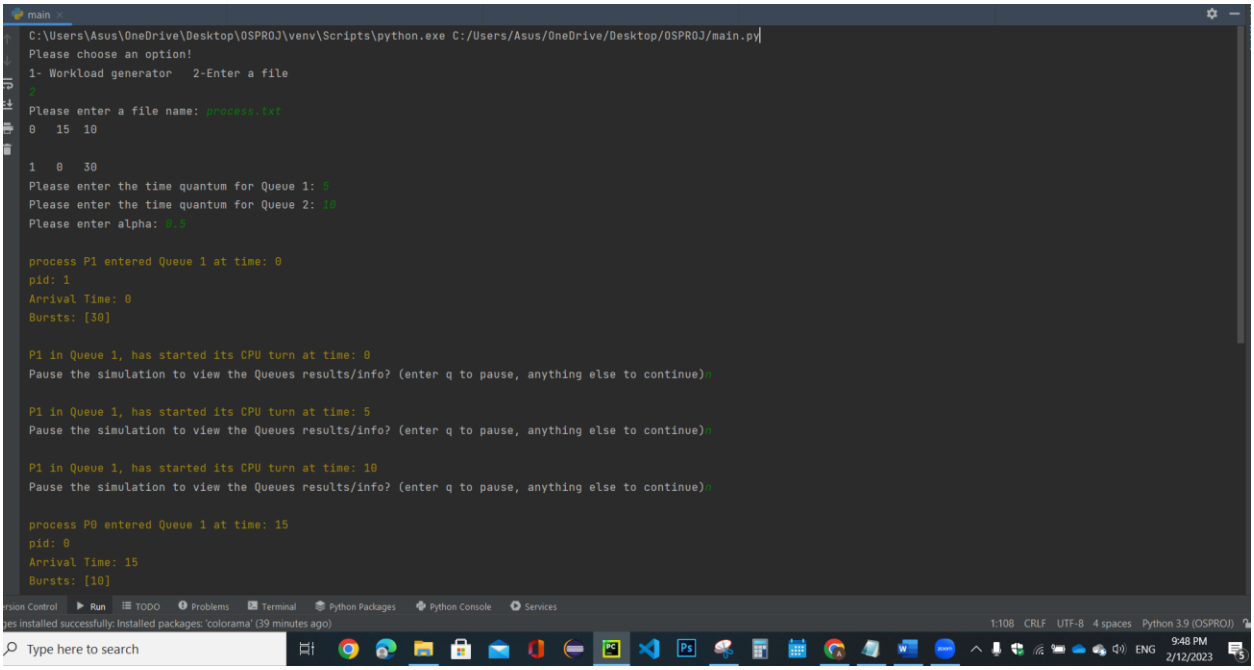

Figure 2: The Execution for the First Test Case-Part One

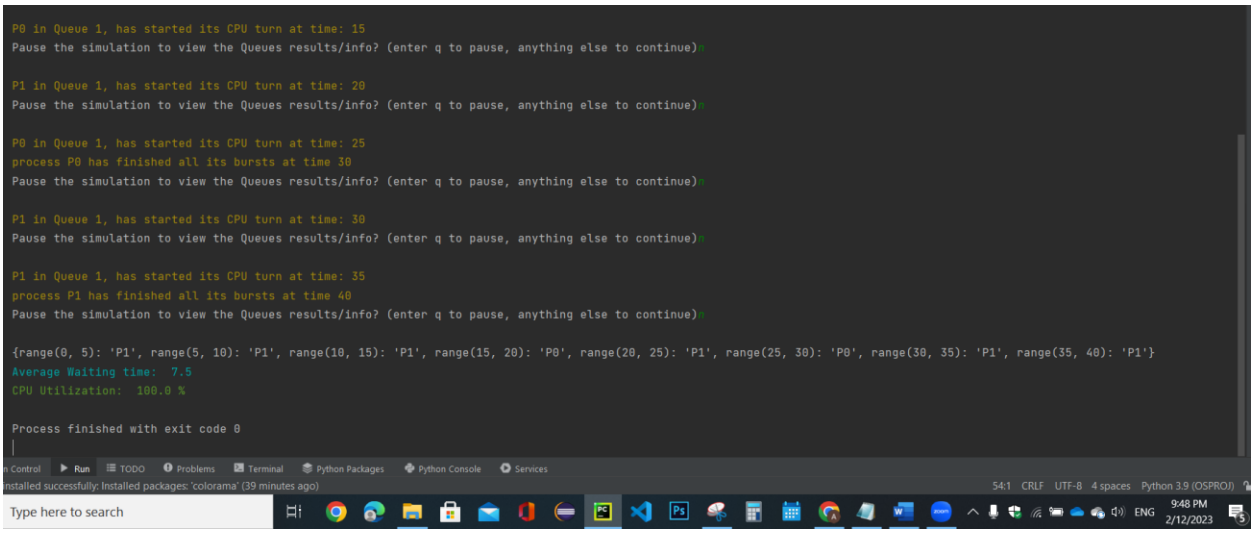

Figure 3: The Execution for the First Test Case-Part Two

Test Case Two

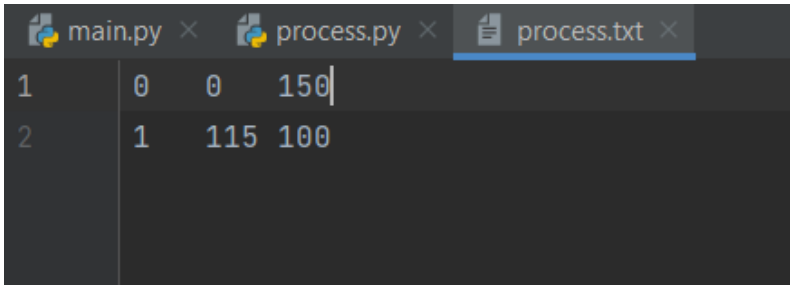

Figure 4: The Second Test Case

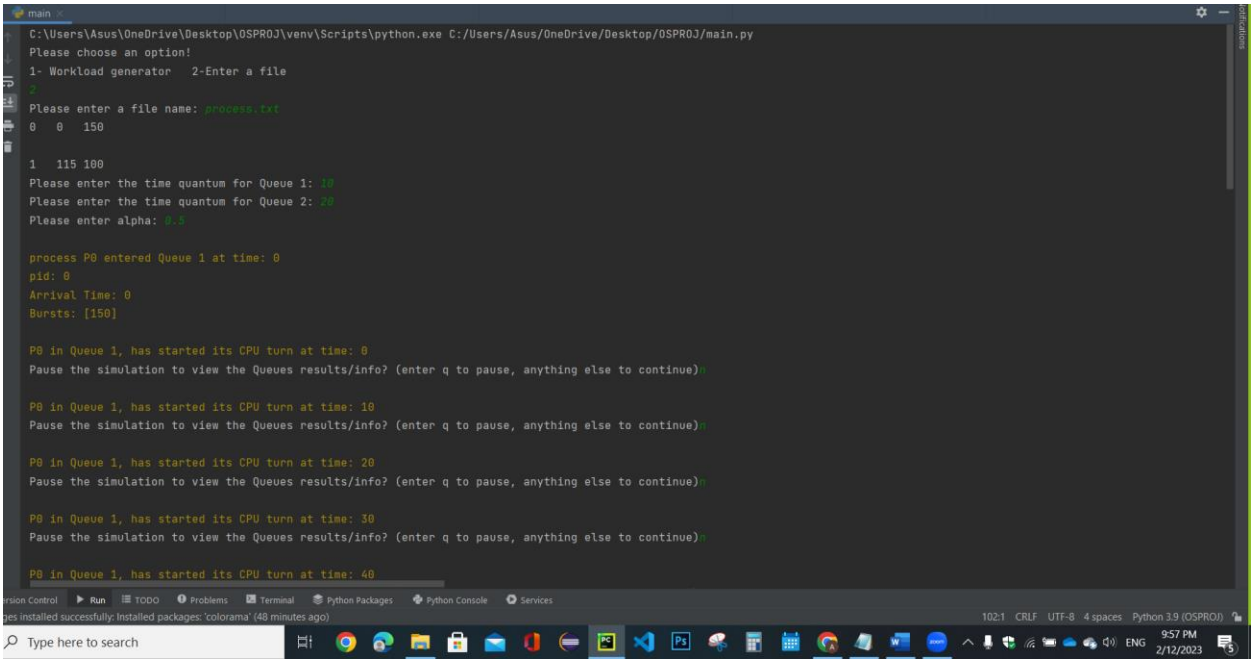

Figure 5: The Execution for the Second Test Case-Part One

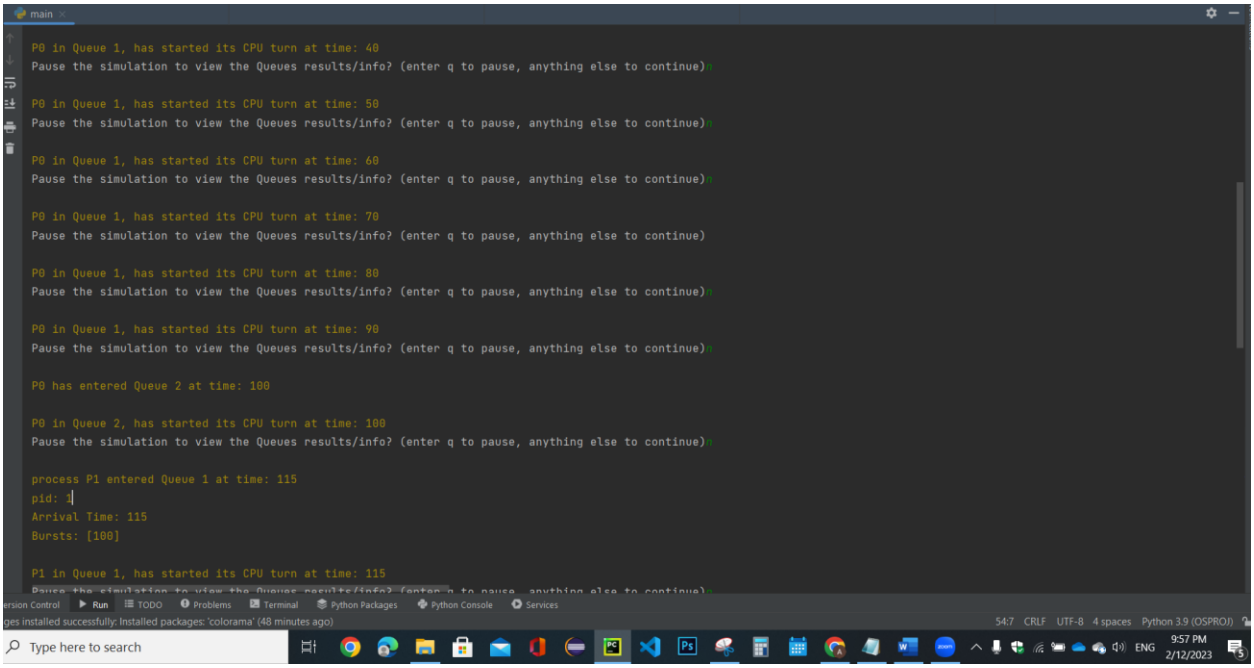

Figure 6: The Execution for the Second Test Case-Part Two

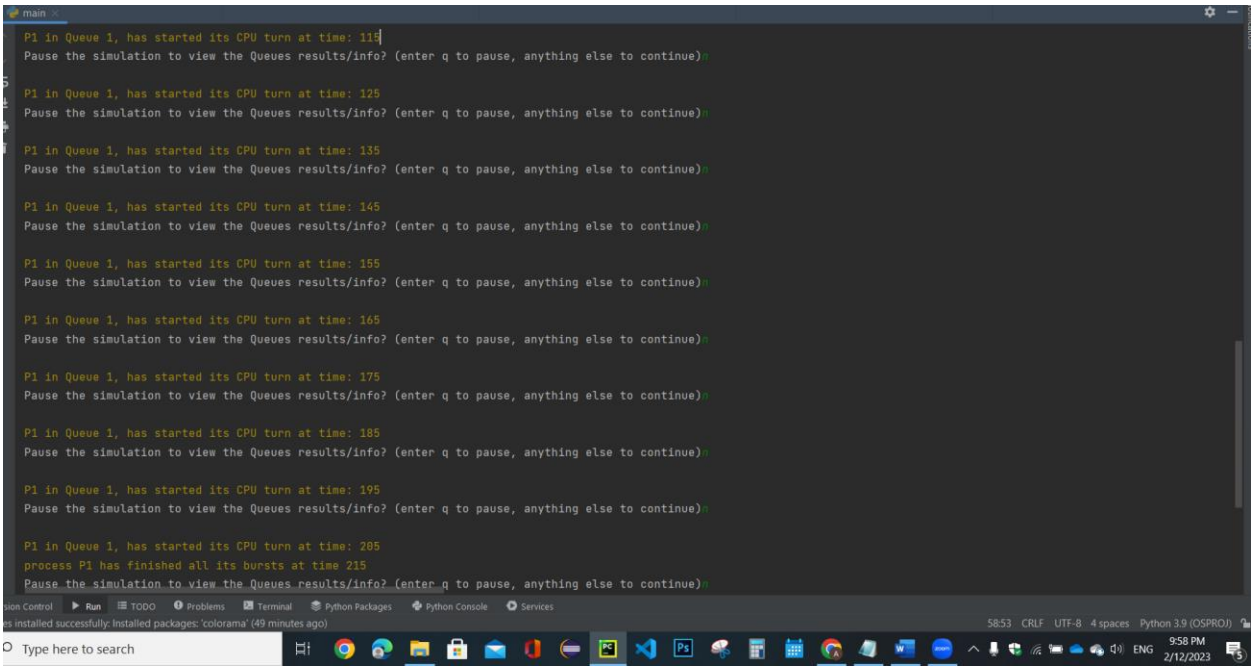

Figure 7: The Execution for the Second Test Case-Part Three

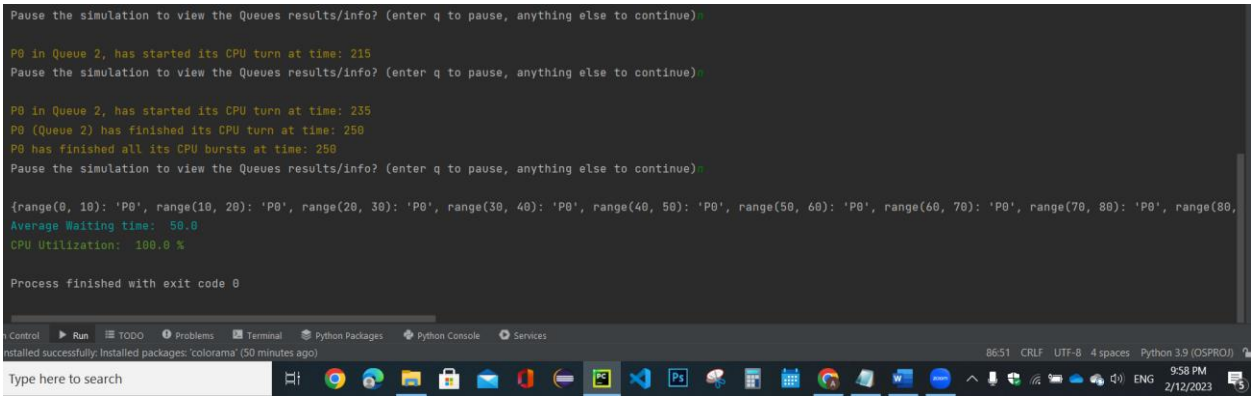

Figure 8: The Execution for the Second Test Case-Part Four

Test Case Three

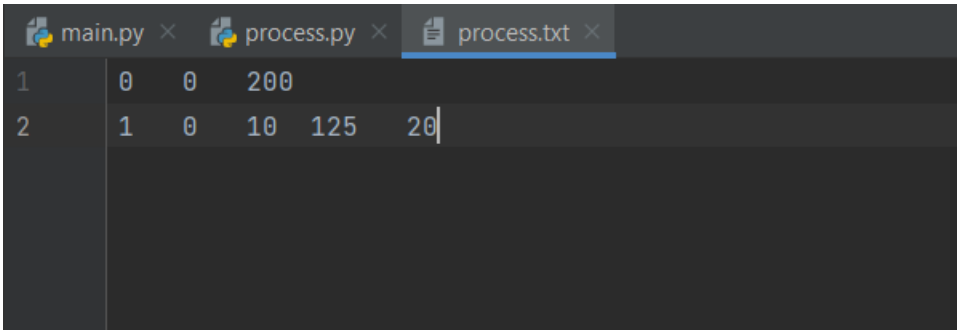

Figure 9: The Third Test Case

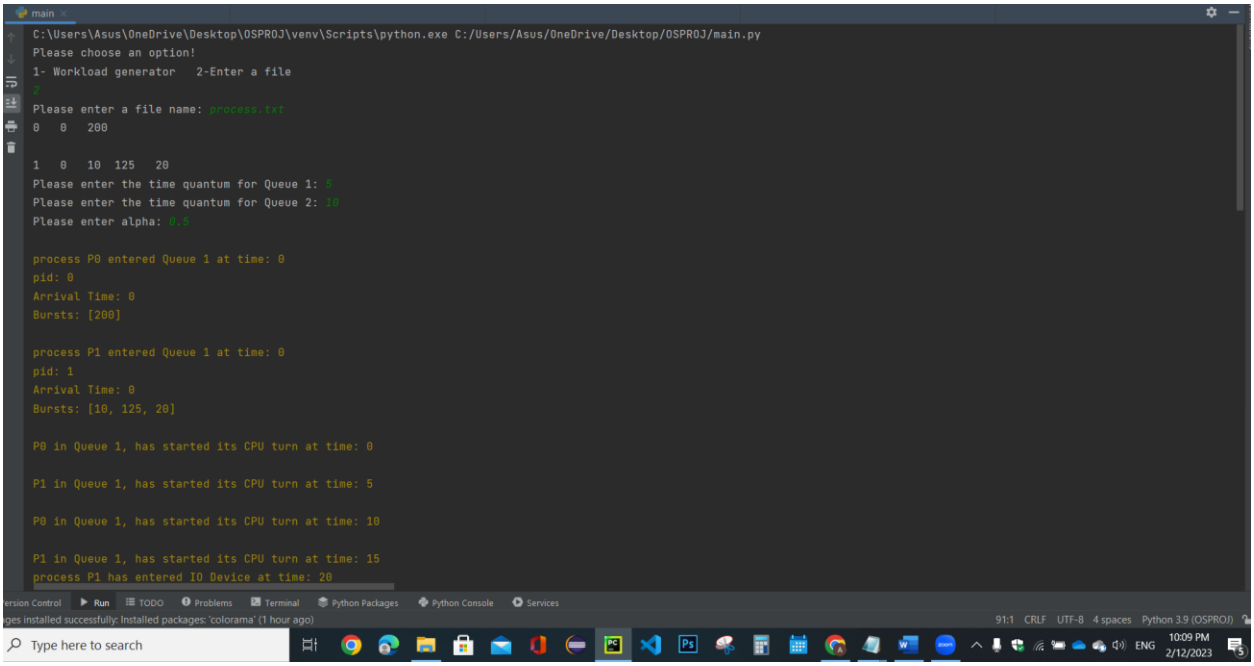

Figure 10: The Execution for the Third Test Case-Part One

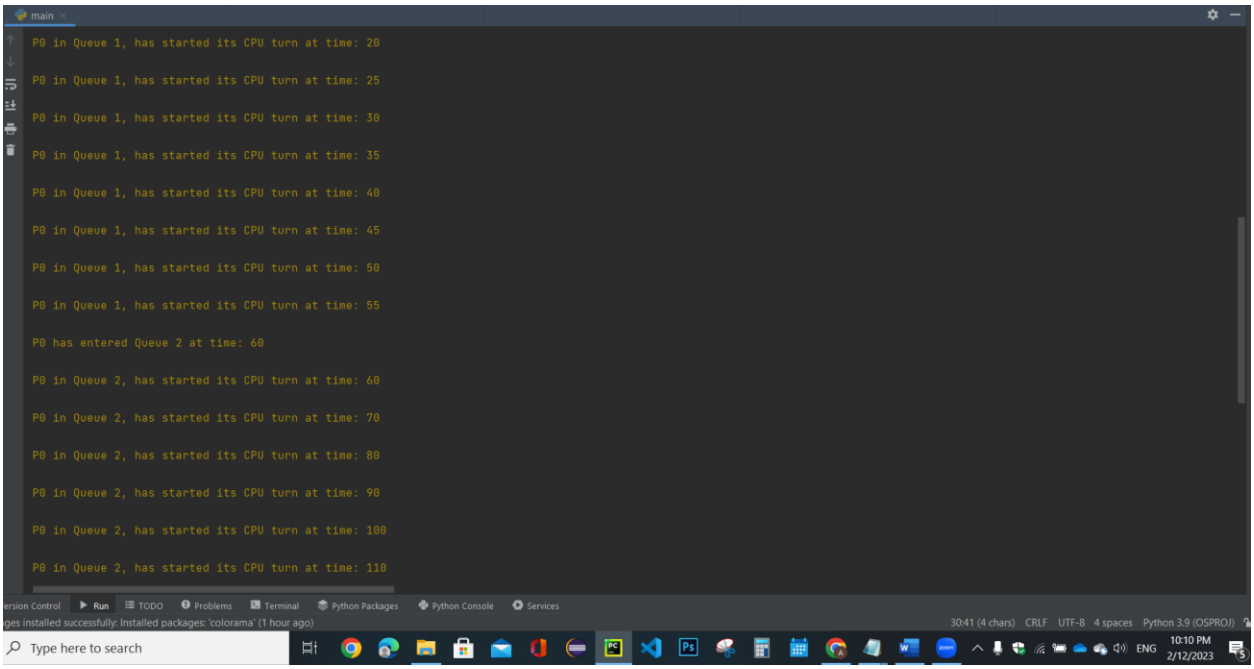

Figure 11: The Execution for the Third Test Case-Part Two

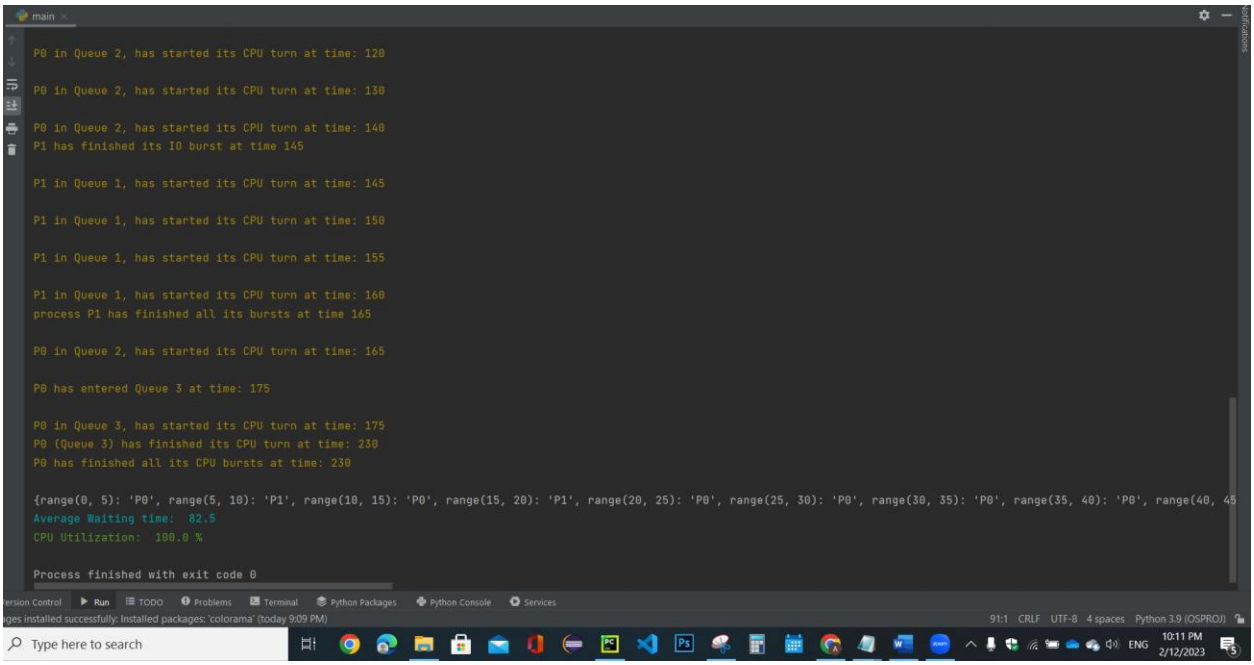

Figure 12: The Execution for the Third Test Case-Part Three

*The last results of this case (Gantt Chart, average waiting time and CPU utilization)

```
{range(0, 5): 'P0', range(5, 10): 'P1', range(10, 15): 'P0', range(15, 20): 'P1', range(20, 25):
'P0', range(25, 30): 'P0', range(30, 35): 'P0', range(35, 40): 'P0', range(40, 45): 'P0', range(45,
50): 'P0', range(50, 55): 'P0', range(55, 60): 'P0', range(60, 70): 'P0', range(70, 80): 'P0',
range(80, 90): 'P0', range(90, 100): 'P0', range(100, 110): 'P0', range(110, 120): 'P0', range(120,
130): 'P0', range(130, 140): 'P0', range(140, 145): 'P0', range(145, 150): 'P1', range(150, 155):
'P1', range(155, 160): 'P1', range(160, 165): 'P1', range(165, 175): 'P0', range(175, 230): 'P0'}

Average Waiting time:  82.5

CPU Utilization:  100.0 %



Process finished with exit code 0
```

Test Case Four

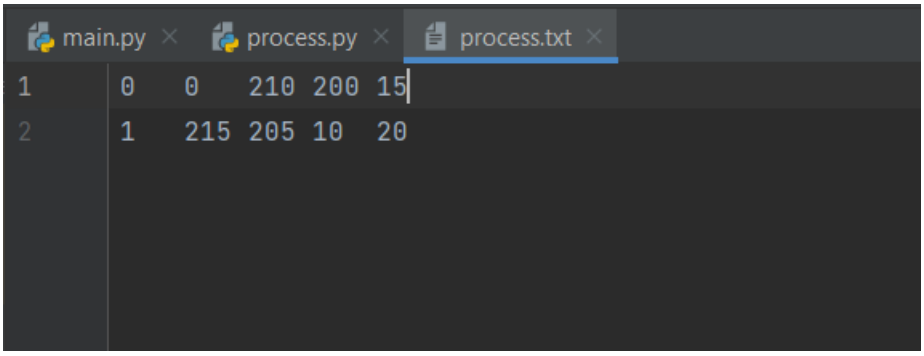

Figure 13: The Fourth Test Case

```
C:\Users\Asus\OneDrive\Desktop\OSPROJ\venv\Scripts\python.exe
C:/Users/Asus/OneDrive/Desktop/OSPROJ/main.py

Please choose an option!

1- Workload generator        2-Enter a file

2

Please enter a file name: process.txt

0         0         210       200       15



1         215       205       10        20

Please enter the time quantum for Queue 1: 5

Please enter the time quantum for Queue 2: 10

Please enter alpha: 0.5



process P0 entered Queue 1 at time: 0

pid: 0
```

```
Arrival Time: 0

Bursts: [210, 200, 15]


P0 in Queue 1, has started its CPU turn at time: 0


P0 in Queue 1, has started its CPU turn at time: 5


P0 in Queue 1, has started its CPU turn at time: 10


P0 in Queue 1, has started its CPU turn at time: 15


P0 in Queue 1, has started its CPU turn at time: 20


P0 in Queue 1, has started its CPU turn at time: 25


P0 in Queue 1, has started its CPU turn at time: 30


P0 in Queue 1, has started its CPU turn at time: 35


P0 in Queue 1, has started its CPU turn at time: 40


P0 in Queue 1, has started its CPU turn at time: 45


P0 has entered Queue 2 at time: 50


P0 in Queue 2, has started its CPU turn at time: 50
```

```
P0 in Queue 2, has started its CPU turn at time: 60

P0 in Queue 2, has started its CPU turn at time: 70

P0 in Queue 2, has started its CPU turn at time: 80

P0 in Queue 2, has started its CPU turn at time: 90

P0 in Queue 2, has started its CPU turn at time: 100

P0 in Queue 2, has started its CPU turn at time: 110

P0 in Queue 2, has started its CPU turn at time: 120

P0 in Queue 2, has started its CPU turn at time: 130

P0 in Queue 2, has started its CPU turn at time: 140

P0 has entered Queue 3 at time: 150

P0 in Queue 3, has started its CPU turn at time: 150
P0 (Queue 3) has finished its CPU turn at time: 210
P0 has entered IO Device at time: 210



process P1 entered Queue 1 at time: 215
pid: 1
```

```
Arrival Time: 215

Bursts: [205, 10, 20]


P1 in Queue 1, has started its CPU turn at time: 215


P1 in Queue 1, has started its CPU turn at time: 220


P1 in Queue 1, has started its CPU turn at time: 225


P1 in Queue 1, has started its CPU turn at time: 230


P1 in Queue 1, has started its CPU turn at time: 235


P1 in Queue 1, has started its CPU turn at time: 240


P1 in Queue 1, has started its CPU turn at time: 245


P1 in Queue 1, has started its CPU turn at time: 250


P1 in Queue 1, has started its CPU turn at time: 255


P1 in Queue 1, has started its CPU turn at time: 260


P1 has entered Queue 2 at time: 265


P1 in Queue 2, has started its CPU turn at time: 265
```

```
P1 in Queue 2, has started its CPU turn at time: 275

P1 in Queue 2, has started its CPU turn at time: 285

P1 in Queue 2, has started its CPU turn at time: 295

P1 in Queue 2, has started its CPU turn at time: 305

P1 in Queue 2, has started its CPU turn at time: 315

P1 in Queue 2, has started its CPU turn at time: 325

P1 in Queue 2, has started its CPU turn at time: 335

P1 in Queue 2, has started its CPU turn at time: 345

P1 in Queue 2, has started its CPU turn at time: 355

P1 has entered Queue 3 at time: 365

P1 in Queue 3, has started its CPU turn at time: 365
P0 has finished its IO burst at time 410

P1 in Queue 3, has started its CPU turn at time: 410
P1 (Queue 3) has finished its CPU turn at time: 420
P1 has entered IO Device at time: 420
```

```
P0 in Queue 3, has started its CPU turn at time: 420

P1 has finished its IO burst at time 430



P1 in Queue 3, has started its CPU turn at time: 430

P1 (Queue 3) has finished its CPU turn at time: 450

P1 has finished all its CPU bursts at time: 450



P0 in Queue 3, has started its CPU turn at time: 450

P0 (Queue 3) has finished its CPU turn at time: 455

P0 has finished all its CPU bursts at time: 455



{range(0, 5): 'P0', range(5, 10): 'P0', range(10, 15): 'P0', range(15, 20): 'P0', range(20, 25):
'P0', range(25, 30): 'P0', range(30, 35): 'P0', range(35, 40): 'P0', range(40, 45): 'P0', range(45,
50): 'P0', range(50, 60): 'P0', range(60, 70): 'P0', range(70, 80): 'P0', range(80, 90): 'P0',
range(90, 100): 'P0', range(100, 110): 'P0', range(110, 120): 'P0', range(120, 130): 'P0',
range(130, 140): 'P0', range(140, 150): 'P0', range(150, 210): 'P0', range(215, 220): 'P1',
range(220, 225): 'P1', range(225, 230): 'P1', range(230, 235): 'P1', range(235, 240): 'P1',
range(240, 245): 'P1', range(245, 250): 'P1', range(250, 255): 'P1', range(255, 260): 'P1',
range(260, 265): 'P1', range(265, 275): 'P1', range(275, 285): 'P1', range(285, 295): 'P1',
range(295, 305): 'P1', range(305, 315): 'P1', range(315, 325): 'P1', range(325, 335): 'P1',
range(335, 345): 'P1', range(345, 355): 'P1', range(355, 365): 'P1', range(365, 410): 'P1',
range(410, 420): 'P1', range(420, 430): 'P0', range(430, 450): 'P1', range(450, 455): 'P0'}

Average Waiting time:  120.0

CPU Utilization:  98.9010989010989 %



Process finished with exit code 0
```