Florida Atlantic University

Department of Electrical Engineering & Computer Science

Computer Operating Systems – COP4610

CPU SCHEDULING PROJECT

Solan Degefa

Sdegefa2020@fau.edu

Friday, October 21, 2022

Table of Contents

Introduction

When it comes to computer performance, the true measure of speed is set in the CPU. That speed is set by how many processes it can perform as quickly as possible. To that extent, a CPU, and by proxy a computer, is performs as well as its scheduling algorithm. The purpose of this project is to simulate three different scheduling algorithms to find the best.

Sample Dynamic Execution Program Output

```
Execution Time:1 - P1 is Running
Execution Time:2 - P1 is Running
Execution Time:3 - P1 is Running
Execution Time:4 - P1 is Running
Execution Time:5 - P1 is Running
Process 1 has run. Execution time: 5
IO Queue: [1]
I/O Queue: Process 1 will be out of the IO in 27
Ready Queue: [3, 4, 5, 6, 7, 8]
CPU Queue: [2]
///////////////////////////////////////////////////////////
//////////////    Context Switch    ///////////////////////
///////////////////////////////////////////////////////////
Ready Queue: ['P3:8', 'P4:3', 'P5:16', 'P6:11', 'P7:14', 'P8:4']
Execution Time:6 - P2 is Running
Execution Time:7 - P2 is Running
Execution Time:8 - P2 is Running
Execution Time:9 - P2 is Running
Process 2 has run. Execution time: 9
IO Queue: [1, 2]
I/O Queue: Process 1 will be out of the IO in 23
I/O Queue: Process 2 will be out of the IO in 48
Ready Queue: [4, 5, 6, 7, 8]
CPU Queue: [3]
```

Logic

FCFS:

In a first come first serve sorting algorithm, the processes will execute in the order they arrive in the ready queue. The first process in this queue is taken and placed into the CPU. As this process completes its CPU burst, it is sent into the I/O queue where it awaits completion of its I/O event. Upon completion of this event, it is placed back into the ready queue.

SJF:

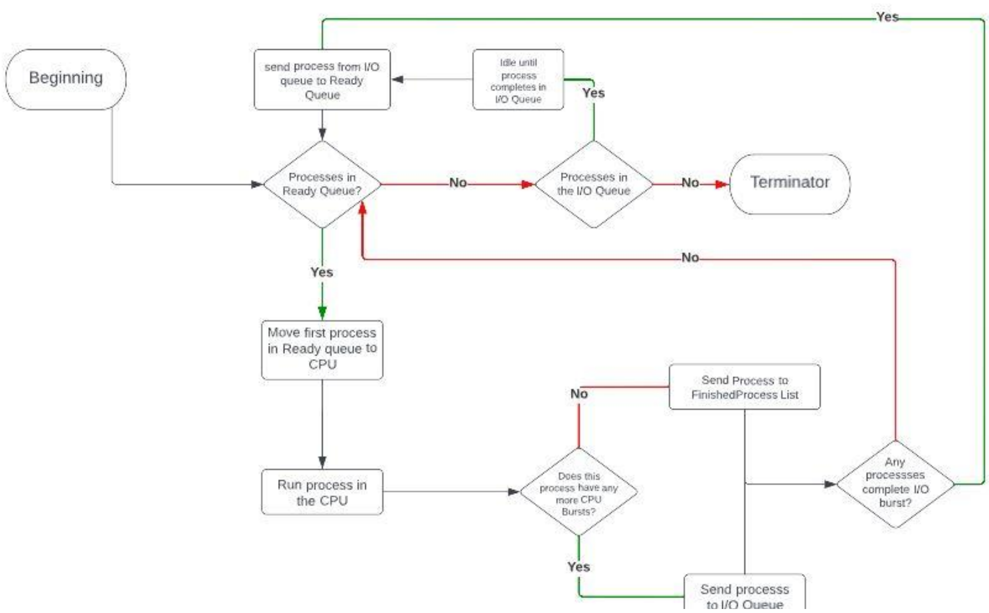
Shortest job first is theoretically the best scheduling algorithm. In this algorithm, the job with the shortest CPU burst time is selected from the ready queue and executed. This is done by sorting the ready queue as processes arrive. After the selection of a process, it follows the same stream of action as first come first serve, CPU to I/O and from I/O to ready queue. The issue with this, and the reason it is only theoretically the best, is because the computer does not always know the length of the burst ahead of time, thus, to produce this algorithm without explicitly inputting the burst times, a prediction algorithm using previous burst times is also needed.

Multilevel Feedback Queue:

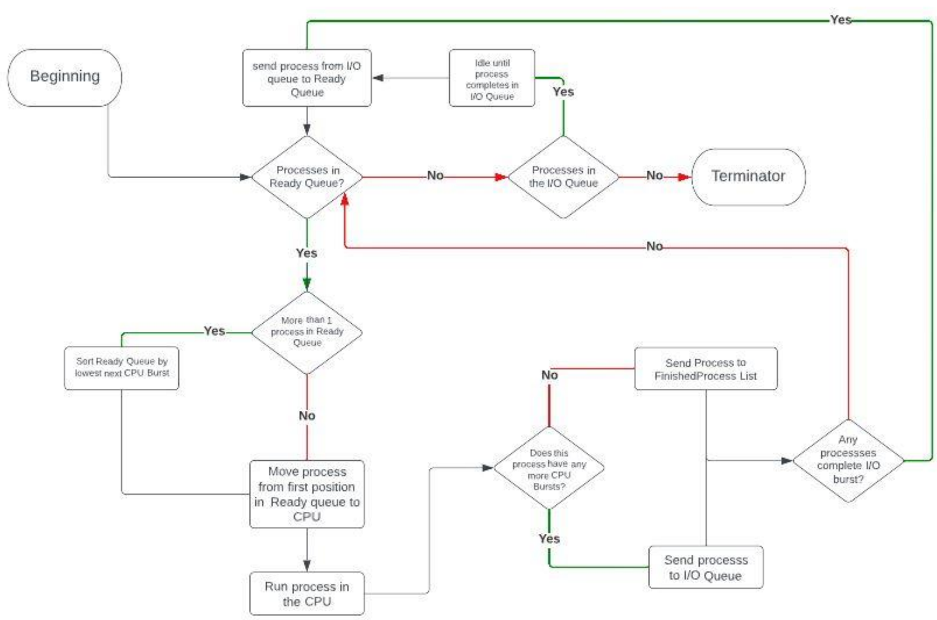
The multilevel feedback queue scheduling algorithm uses the behavior of the processes within it to assign each a priority. In the context of this project, the first and highest level of priority began with all processes. Each process was allowed to run for five time units, where it was either sent to the I/O, if it completed the burst, or sent to the second priority queue if it had not. Processes in the lower priority queues are not able to run if there is an available process in the queue above it. If the first queue is empty, then a process in the second queue is allowed to run. Here it is given a limit of ten time units. If it fails to complete again, it is sent to the third and final queue. If it completes its burst however, it is placed in the I/O queue and upon completion of its I/O event, placed back into the first queue. Programs in the lowest priority queue are only able to run if no processes are available in the two high priority queues and follow a first come first serve scheduling algorithm.

Flow Chart

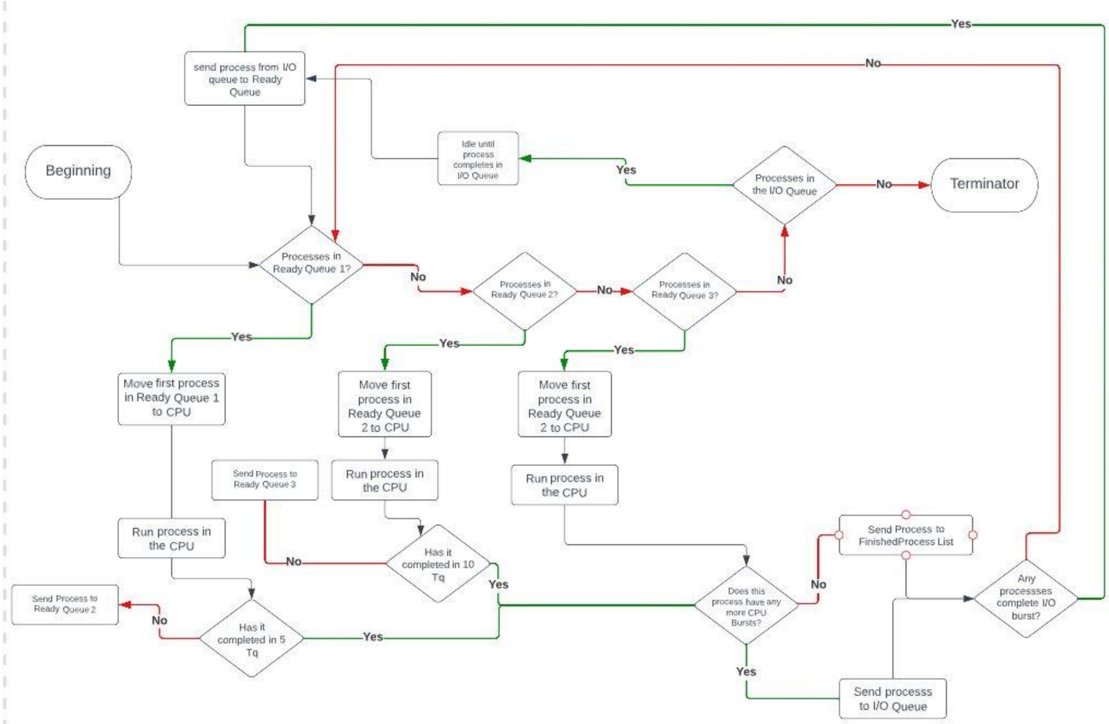
FCFS:



SJF:



MLFQ:



Simulation Results

Final Analytics

Scheduling Type	SJF	FCFS	MLFQ
AVG Wait Time	133.25	186.0	187.625
AVG Turnaround	469.375	522.25	524.375
AVG Response	27.125	24.375	15.75
CPU Utilization	82.537	84.299	87.224
Total Time	670	656	634

First Come First Serve

Process	Waiting Time	Turnaround Time	Response Time
P1	174	395	0
P2	163	597	5
P3	169	562	9
P4	170	656	17
P5	213	519	20
P6	229	441	36
P7	189	515	47
P8	181	493	61

Shortest Job First

Process	Waiting Time	Turnaround Time	Response Time
P1	45	268	11
P2	68	500	3
P3	277	670	16
P4	48	534	0
P5	239	546	109
P6	123	332	24
P7	148	477	47
P8	118	428	7

Multilevel Feedback Queue

Process	Waiting Time	Turnaround Time	Response Time
P1	81	306	0
P2	107	534	5
P3	222	620	9
P4	91	574	14
P5	326	634	17
P6	194	403	22
P7	256	589	27
P8	224	535	32

Analysis

The first come first serve scheduling algorithm in nature is simple. As such, it was middle of the park in all categories. In average wait time and turnaround time, the multilevel feedback queue scheduling algorithm was slightly larger, in total execution time and average response time, the shortest job first scheduling algorithm was larger. Lacking in complexity and efficiency, it is simple to implement but comes at the cost of total execution time and CPU utilization.

The shortest job first scheduling algorithm specializes in minimizing waiting time, allowing shorter processes to be executed instead of being stuck behind large processes. However, the tradeoff is the larger projects being forced to wait long periods of time, which leads to large total execution times and starvation. The long waits combined with long I/O periods lead to low CPU utilization as shown in this project.

The multilevel feedback queue scheduling algorithm attempts to combat the shortcomings of the shortest job first scheduling algorithm, allowing all processes the ability to run, then demoting the longer processes to run again while the CPU would otherwise be idle. This however did result in the similar issue of starvation. As it was specified not to include an aging mechanic into this algorithm, it was possible for processes to get stuck in the lowest queue and significantly extend the total execution time and lower CPU utilization. With aging, this algorithm would erase the issue of starvation and perhaps have higher CPU utilization and lower waiting and turnaround times.

Sample Output

CPU Queue: [1]

//

//////////////   Context Switch   ///////////////////////

//

Ready Queue: ['P2:4', 'P3:8', 'P4:3', 'P5:16', 'P6:11', 'P7:14', 'P8:4']

Execution Time:1 - P1 is Running

Execution Time:2 - P1 is Running

Execution Time:3 - P1 is Running

Execution Time:4 - P1 is Running

Execution Time:5 - P1 is Running

Process 1 has run. Execution time: 5

IO Queue: [1]

I/O Queue: Process 1 will be out of the IO in 27

Ready Queue: [3, 4, 5, 6, 7, 8]

CPU Queue: [2]

//

//////////////   Context Switch   ///////////////////////

//

Ready Queue: ['P3:8', 'P4:3', 'P5:16', 'P6:11', 'P7:14', 'P8:4']

Execution Time:6 - P2 is Running

Execution Time:7 - P2 is Running

Execution Time:8 - P2 is Running

Execution Time:9 - P2 is Running

Process 2 has run. Execution time: 9

IO Queue: [1, 2]

I/O Queue: Process 1 will be out of the IO in 23

I/O Queue: Process 2 will be out of the IO in 48

Ready Queue: [4, 5, 6, 7, 8]

CPU Queue: [3]

//

//////////////   Context Switch   //////////////////////

//

Ready Queue: ['P4:3', 'P5:16', 'P6:11', 'P7:14', 'P8:4']

Execution Time:10 - P3 is Running

Execution Time:11 - P3 is Running

Execution Time:12 - P3 is Running

Execution Time:13 - P3 is Running

Execution Time:14 - P3 is Running

Execution Time:15 - P3 is Running

Execution Time:16 - P3 is Running

Execution Time:17 - P3 is Running

Process 3 has run. Execution time: 17

IO Queue: [1, 2, 3]

I/O Queue: Process 1 will be out of the IO in 15

I/O Queue: Process 2 will be out of the IO in 40

I/O Queue: Process 3 will be out of the IO in 33

Ready Queue: [5, 6, 7, 8]

CPU Queue: [4]

//

//////////////   Context Switch   //////////////////////

//

Ready Queue: ['P5:16', 'P6:11', 'P7:14', 'P8:4']

Execution Time:18 - P4 is Running

Execution Time:19 - P4 is Running

Execution Time:20 - P4 is Running

Process 4 has run. Execution time: 20

IO Queue: [1, 2, 3, 4]

I/O Queue: Process 1 will be out of the IO in 12

I/O Queue: Process 2 will be out of the IO in 37

I/O Queue: Process 3 will be out of the IO in 30

I/O Queue: Process 4 will be out of the IO in 35

Ready Queue: [6, 7, 8]

Table of Results for Comparison

Data Types	SJF	FCFS	MLFQ
CPU Utilization	82.537%	84.299%	87.224%
AVG Waiting Time (Tw)	133.25	186.0	187.625
AVG Turnaround Time (Ttr)	469.375	522.250	524.375
AVG Response Time (Tr)	27.125	24.37	15.75
Total Time	670	656	634

	SJF CPU utilization:		**82.537%**	**FCFS** CPU utilization:		**84.299%**	**MLFQ** CPU utilization:		**87.224%**
	Tw	***Ttr***	***Tr***	***Tw***	***Ttr***	***Tr***	***Tw***	***Ttr***	***Tr***
P1	45	268	11	174	395	0	81	306	0
P2	68	500	3	163	597	5	107	534	5
P3	277	670	16	169	562	9	222	620	9
P4	48	534	0	170	656	17	91	574	14
P5	239	546	109	213	519	20	326	634	17
P6	123	332	24	229	441	36	194	403	22
P7	148	477	47	189	515	47	256	589	27
P8	118	428	7	181	493	61	224	535	32
Avg	***133.25***	***469.375***	***27.125***	***186***	***522.25***	***24.37***	***187.625***	***524.375***	**15.75**

Source Code:

Process.py

```
class Process:

    def __init__(self, pID, burst):
        self.pID = pID
        self.burst = burst

        self.cpu_burst = []
        self.io_burst = []
        self.ioWaitTime = 0

    def parse(self):
        for i in range(len(self.burst)):
            if i % 2 == 0:
                self.cpu_burst.append(self.burst[i])
            else:
                self.io_burst.append(self.burst[i])

    def getCPUBurst(self):
        return self.cpu_burst[0]

    def __repr__(self):
        return "Process('p{}', 'CPU Burst:{}', 'IO Burst:{}')".format(self.pID, self.cpu_burst, self.io_burst)
```

Scheduler.py

```
processQueue2 = []
processQueue3 = []


readyQueue = []
cpuQueue = []
finishedProcesses = []
waitingInIO =[]


#                        fcfs  sjf  mlfq
cpuUtilization =      [  0,    0,   0  ]
finalExecutionTimes = [  0,    0,   0  ]



#              Tr   Tw   Ttr
#             [0]  [0]  [0]
avgTimes =  [
              [0,   0,   0],# fcfs
              [0,   0,   0],# sjf
              [0,   0,   0],# mlfq
            ]


# #   Has Run  pID     Tr     Tw     Ttr
# #     [0]     [0]    [0]    [0]     [0]
dataBank = [
       [0, 0, 0, 0, 0],    # p1
       [0, 0, 0, 0, 0],    # p2
       [0, 0, 0, 0, 0],    # p3
       [0, 0, 0, 0, 0],    # p4
       [0, 0, 0, 0, 0],    # p5
       [0, 0, 0, 0, 0],    # p6
       [0, 0, 0, 0, 0],    # p7
       [0, 0, 0, 0, 0]     # p8
       ]

dataBankFCFS = [
       [0, 0, 0, 0, 0],    # p1
       [0, 0, 0, 0, 0],    # p2
       [0, 0, 0, 0, 0],    # p3
       [0, 0, 0, 0, 0],    # p4
       [0, 0, 0, 0, 0],    # p5
       [0, 0, 0, 0, 0],    # p6
       [0, 0, 0, 0, 0],    # p7
       [0, 0, 0, 0, 0]     # p8
       ]
```

```
dataBankSJF = [
    [0, 0, 0, 0, 0],   # p1
    [0, 0, 0, 0, 0],   # p2
    [0, 0, 0, 0, 0],   # p3
    [0, 0, 0, 0, 0],   # p4
    [0, 0, 0, 0, 0],   # p5
    [0, 0, 0, 0, 0],   # p6
    [0, 0, 0, 0, 0],   # p7
    [0, 0, 0, 0, 0]    # p8
    ]

dataBankMLFQ = [
    [0, 0, 0, 0, 0],   # p1
    [0, 0, 0, 0, 0],   # p2
    [0, 0, 0, 0, 0],   # p3
    [0, 0, 0, 0, 0],   # p4
    [0, 0, 0, 0, 0],   # p5
    [0, 0, 0, 0, 0],   # p6
    [0, 0, 0, 0, 0],   # p7
    [0, 0, 0, 0, 0]    # p8
    ]

dataBanks = [dataBankFCFS, dataBankSJF, dataBankMLFQ]

def add(p):
  readyQueue.append(p)


def fromReadyToCPU():


    cpuQueue.append(readyQueue[0])
    readyQueue.pop(0)
    print(f"Ready Queue:", readyQueue)
    print(f"CPU Queue:", cpuQueue)

def fromCPUToIO():
    waitingInIO.append(cpuQueue[0])
    cpuQueue.pop(0)
```

Main.py

```
# -*- coding: utf-8 -*-


from Process import *
from Scheduler import *


executionTime = 0

p1 = Process(1, [5, 27, 3, 31, 5, 43, 4, 18, 6, 22, 4, 26, 3, 24, 4])
p2 = Process(2, [4, 48, 5, 44, 7, 42, 12, 37, 9, 76, 4, 41, 9, 31, 7, 43, 8], )
p3 = Process(3, [8, 33, 12, 41, 18, 65, 14, 21, 4, 61, 15, 18, 14, 26, 5, 31, 6])
p4 = Process(4, [3, 35, 4, 41, 5, 45, 3, 51, 4, 61, 5, 54, 6, 82, 5, 77, 3])
p5 = Process(5, [16, 24, 17, 21, 5, 36, 16, 26, 7, 31, 13, 28, 11, 21, 6, 13, 3, 11, 4])
p6 = Process(6, [11, 22, 4, 8, 5, 10, 6, 12, 7, 14, 9, 18, 12, 24, 15, 30, 8])
p7 = Process(7, [14, 46, 17, 41, 11, 42, 15, 21, 4, 32, 7, 19, 16, 33, 10])
p8 = Process(8, [4, 14, 5, 33, 6, 51, 14, 73, 16, 87, 6])
processList = [p1, p2, p3, p4, p5, p6, p7, p8]


def loadReadyQueue():
    for j in processList:
        j.parse()
    for processes in range(len(processList)):
        add(processList[processes].pID)


def FCFS():
    loadReadyQueue()
    resetDataBank()
    print(f'Ready Queue:{readyQueue}')
    global wastedTime
    wastedTime = 0
    global executionTime
    executionTime = 0

    while len(readyQueue) > 0:
        fromReadyToCPU()
        printOnContextSwitch()
        currentProcess = processList[cpuQueue[0] - 1]

        if dataBank[currentProcess.pID - 1][0] == 0:  # first time running marker
            dataBank[currentProcess.pID - 1][2] = executionTime  # saving the response time
            dataBank[currentProcess.pID - 1][0] = 1

        while currentProcess.cpu_burst[0] > 0:
            executionTime += 1
            currentProcess.cpu_burst[0] -= 1
            print(f'Execution Time:{executionTime} - P{currentProcess.pID} is Running')
            decIO()
            countWait()

        if currentProcess.cpu_burst[0] == 0:
            print("Process {} has run. Execution time: {}".format(currentProcess.pID, executionTime))
```

```
        currentProcess.cpu_burst.pop(0)

        try:
            # currentProcess.ioWaitTime = executionTime + currentProcess.io_burst[0]
            # currentProcess.io_burst.pop(0)
            if not currentProcess.io_burst:
                finishedProcesses.append(currentProcess.pID)
                cpuQueue.pop(0)
                print(f'P{currentProcess.pID} had finished')
                dataBank[currentProcess.pID - 1][4] = executionTime  # sets turnaround time
            else:
                fromCPUToIO()

            print(f'IO Queue: {waitingInIO}')
            for c in waitingInIO:
                print(f'I/O Queue: Process', processList[c - 1].pID, "will be out of the IO in",
                      processList[c - 1].io_burst[0])
        except IndexError:
            finishedProcesses.append(currentProcess.pID)
            cpuQueue.pop(0)

    while readyQueue == [] and waitingInIO:
        print(f'CPU IDLING\t Execution Time:{executionTime}')
        decIO()
        wastedTime += 1
        executionTime +=1
        print(f'Finished Processes: {finishedProcesses}')

    if not readyQueue and not waitingInIO and not cpuQueue:
        saveData(0)

        print('FCFS Complete')


def SJF():
    loadReadyQueue()
    finishedProcesses = []

    print(f'Ready Queue:{readyQueue}')
    global wastedTime
    wastedTime = 0
    global executionTime
    executionTime = 0
    while len(readyQueue) > 0:
        sortReadyQueue()
        fromReadyToCPU()
        printOnContextSwitch()
        currentProcess = processList[cpuQueue[0] - 1]
        if dataBank[currentProcess.pID - 1][0] == 0:  # first time running marker
            dataBank[currentProcess.pID - 1][2] = executionTime  # saving the response time
            dataBank[currentProcess.pID - 1][0] = 1
        while currentProcess.cpu_burst[0] > 0:
            executionTime += 1
            currentProcess.cpu_burst[0] -= 1
            print(f'Execution Time:{executionTime} - P{currentProcess.pID} is Running')
            decIO()
```

```
            countWait()
        if currentProcess.cpu_burst[0] == 0:
            print("Process {} has run. Execution time: {}".format(currentProcess.pID, executionTime))
            currentProcess.cpu_burst.pop(0)
            try:
                # currentProcess.ioWaitTime = executionTime + currentProcess.io_burst[0]
                # currentProcess.io_burst.pop(0)
                if not currentProcess.io_burst:
                    finishedProcesses.append(currentProcess.pID)
                    cpuQueue.pop(0)
                    print(f'P{currentProcess.pID} had finished')
                    dataBank[currentProcess.pID - 1][4] = executionTime  # sets turnaround time
                else:
                    fromCPUToIO()
                print(f'IO Queue: {waitingInIO}')
                for c in waitingInIO:
                    print(f'I/O Queue: Process', processList[c - 1].pID, "will be out of the IO in",
                          processList[c - 1].io_burst[0])
            except IndexError:
                finishedProcesses.append(currentProcess.pID)
                cpuQueue.pop(0)
        while readyQueue == [] and waitingInIO:
            print(f'CPU IDLING\t Execution Time:{executionTime}')
            decIO()
            wastedTime += 1
            executionTime += 1
            print(f'Finished Processes: {finishedProcesses}')
        if not readyQueue and not waitingInIO and not cpuQueue:
            saveData(1)  # data bank index key: FCFS = 0; SJF = 1; MLFQ = 2
            print('SJF Complete')

def MLFQ():
    loadReadyQueue()
    finishedProcesses = []

    print(f'Ready Queue:{readyQueue}')
    global wastedTime
    wastedTime = 0
    global executionTime
    executionTime = 0
    global timeQuantum
    timeQuantum = 0

    while readyQueue or processQueue2 or processQueue3 or waitingInIO:
        while len(readyQueue) > 0:  # RR for Priority Queue 1
            printOnContextSwitch()
            fromReadyToCPU()

            currentProcess = processList[cpuQueue[0] - 1]

            if dataBank[currentProcess.pID - 1][0] == 0:  # first time running marker
                dataBank[currentProcess.pID - 1][2] = executionTime  # saving the response time
                dataBank[currentProcess.pID - 1][0] = 1
            timeQuantum = 0
            while currentProcess.cpu_burst[0] > 0 and timeQuantum < 5:
                executionTime += 1
```

```
        currentProcess.cpu_burst[0] -= 1
        timeQuantum += 1
        print(f'Execution Time:{executionTime} - P{currentProcess.pID} is Running - Tq: {timeQuantum}')
        decIO()
        countWait()
        if timeQuantum == 5 and currentProcess.cpu_burst[0] > 0:
            demoteProcessToP2(currentProcess)
            timeQuantum = 0
            break

        if currentProcess.cpu_burst[0] == 0:
            print("Process {} has run. Execution time: {}".format(currentProcess.pID, executionTime))
            currentProcess.cpu_burst.pop(0)

            try:
                # currentProcess.ioWaitTime = executionTime + currentProcess.io_burst[0]
                # currentProcess.io_burst.pop(0)
                if not currentProcess.io_burst:
                    finishedProcesses.append(currentProcess.pID)
                    cpuQueue.pop(0)
                    print(f'P{currentProcess.pID} had finished')
                    dataBank[currentProcess.pID - 1][4] = executionTime  # sets turnaround time
                    break
                else:
                    fromCPUToIO()
                    break

                print(f'IO Queue: {waitingInIO}')
                for c in waitingInIO:
                    print(f'I/O Queue: Process', processList[c - 1].pID, "will be out of the IO in",
                          processList[c - 1].io_burst[0])
                break
            except IndexError:  # if no more cpu bursts
                finishedProcesses.append(currentProcess.pID)
                cpuQueue.pop(0)

    # while readyQueue == [] and waitingInIO:
    #     print(f'CPU IDLING\t Execution Time:{executionTime}')
    #     decIO()
    #     wastedTime += 1
    #     print(f'Finished Processes: {finishedProcesses}')

    # if not readyQueue and not waitingInIO and not cpuQueue:
    #    printData()

while readyQueue == [] and len(processQueue2) > 0:  # RR for Priority Queue 2

    print('////////////////////////////////////////////////////////////\n'
          '//////////////    Context Switch    ///////////////////////\n'
          '////////////////////////////////////////////////////////////')

    processQueue2WithTimes = []

    for i in processQueue2:
        processQueue2WithTimes.append(f'P{i}:{processList[i - 1].cpu_burst[0]}')
    print(f'Ready Queue 2: {processQueue2WithTimes}')
```

```
    cpuQueue.append(processQueue2[0])
    processQueue2.pop(0)

    currentProcess = processList[cpuQueue[0] - 1]

    timeQuantum = 0
    while currentProcess.cpu_burst[0] > 0 and timeQuantum < 10:
        executionTime += 1
        currentProcess.cpu_burst[0] -= 1
        timeQuantum += 1
        print(f'Execution Time:{executionTime} - P{currentProcess.pID} is Running - Tq: {timeQuantum}')
        decIO()
        countWait()

        if timeQuantum == 10 and currentProcess.cpu_burst[0] > 0:
            demoteProcessToP3(currentProcess)
            timeQuantum = 0
            break

        if currentProcess.cpu_burst[0] == 0:
            print("Process {} has run. Execution time: {}".format(currentProcess.pID, executionTime))
            currentProcess.cpu_burst.pop(0)
            print(currentProcess.cpu_burst)
            print(f'Process Queue 3 :{processQueue3}')

            try:
                # currentProcess.ioWaitTime = executionTime + currentProcess.io_burst[0]
                # currentProcess.io_burst.pop(0)
                if not currentProcess.io_burst:
                    finishedProcesses.append(currentProcess.pID)
                    cpuQueue.pop(0)
                    print(f'P{currentProcess.pID} had finished')
                    dataBank[currentProcess.pID - 1][4] = executionTime  # sets turnaround time
                    break
                else:
                    fromCPUToIO()
                    break

                print(f'IO Queue: {waitingInIO}')
                for c in waitingInIO:
                    print(f'I/O Queue: Process', processList[c - 1].pID, "will be out of the IO in",
                          processList[c - 1].io_burst[0])
                break
            except IndexError:
                finishedProcesses.append(currentProcess.pID)
                cpuQueue.pop(0)
                break
    # while processQueue2 == [] and waitingInIO and not processQueue3 and not readyQueue:
    #     print(f'CPU IDLING\t Execution Time:{executionTime}')
    #     decIO()
    #     wastedTime += 1
    #     print(f'Finished Processes: {finishedProcesses}')

while readyQueue == [] and processQueue2 == [] and len(processQueue3) > 0:
```

```
    print('//////////////////////////////////////////////////\n'
          '///////////////    Context Switch   ///////////////\n'
          '//////////////////////////////////////////////////')
    processQueue3WithTimes = []
    print(f'Execution Time: {executionTime}')

    for i in processQueue3:
        processQueue3WithTimes.append(f'P{i}:{processList[i - 1].cpu_burst[0]}')
    print(f'Ready Queue 3: {processQueue3WithTimes}')

    cpuQueue.append(processQueue3[0])
    processQueue3.pop(0)

    currentProcess = processList[cpuQueue[0] - 1]
    while currentProcess.cpu_burst[0] > 0:
        executionTime += 1
        currentProcess.cpu_burst[0] -= 1
        print(f'Execution Time:{executionTime} - P{currentProcess.pID} is Running')
        decIO()
        countWait()

    if currentProcess.cpu_burst[0] == 0:
        print("Process {} has run. Execution time: {}".format(currentProcess.pID, executionTime))
        currentProcess.cpu_burst.pop(0)

        try:
            # currentProcess.ioWaitTime = executionTime + currentProcess.io_burst[0]
            # currentProcess.io_burst.pop(0)
            if not currentProcess.io_burst:
                finishedProcesses.append(currentProcess.pID)
                cpuQueue.pop(0)
                print(f'P{currentProcess.pID} had finished')
                dataBank[currentProcess.pID - 1][4] = executionTime  # sets turnaround time
            else:
                fromCPUToIO()

            print(f'IO Queue: {waitingInIO}')
            for c in waitingInIO:
                print(f'I/O Queue: Process', processList[c - 1].pID, "will be out of the IO in",
                      processList[c - 1].io_burst[0])
        except IndexError:
            finishedProcesses.append(currentProcess.pID)
            cpuQueue.pop(0)

while processQueue3 == [] and waitingInIO and not processQueue2 and not readyQueue:
    print(f'CPU IDLING\t Execution Time:{executionTime}')
    decIO()
    wastedTime += 1
    executionTime += 1
    print(f'Finished Processes: {finishedProcesses}')

    for i in waitingInIO:
        print(f'I/O Queue: Process', processList[i - 1].pID, "will be out of the IO in",
              processList[i - 1].io_burst[0])
if not readyQueue and not waitingInIO and not cpuQueue and not processQueue2 and not processQueue3:
    saveData(2)  # data bank index key: FCFS = 0; SJF = 1; MLFQ = 2
```

```
        print('MLFQ Complete')

def demoteProcessToP2(process):
    cpuQueue.pop(0)
    processQueue2.append(process.pID)
    print(processQueue2)


def demoteProcessToP3(process):
    cpuQueue.pop(0)
    processQueue3.append(process.pID)


def countWait():
    for process in readyQueue:  # processes start use process ID so -1 to properly index
        dataBank[process - 1][3] += 1
    for process in processQueue2:  # processes start use process ID so -1 to properly index
        dataBank[process - 1][3] += 1
    for process in processQueue3:  # processes start use process ID so -1 to properly index
        dataBank[process - 1][3] += 1


def decIO():
    try:
        # for process in range(len(waitingInIO)):
        processs = 0
        while processs in range(len(waitingInIO)):

            p = processList[waitingInIO[processs] - 1]
            p.io_burst[0] -= 1
            # for i in range(len(p1.io_burst)):
            #    print(p1.io_burst[i])
            if p.io_burst[0] == 0:
                p.io_burst.pop(0)
                readyQueue.append(p.pID)
                waitingInIO.remove(p.pID)
                print('P{} has left the IO'.format(p.pID))
            # for c in waitingInIO:
            #    print(f'I/O Queue: Process', processList[c - 1].pID, "will be out of the IO in", processList[c -
1].io_burst[0])

            processs += 1
    except:  # debugging
        print(f'IO Queue: {waitingInIO}')
        print(p, 'broke in decIO')
        print(f'Ready Queue: {readyQueue}')
        for c in waitingInIO:
            print(f'I/O Queue: Process', processList[c - 1].pID, "will be out of the IO at",
                  processList[c - 1].ioWaitTime)
        exit()


def saveData(dataBankIndex):
    avgTat = 0
    avgTr = 0
    avgTwt = 0
```

```
    cpuUtilTemp = 0

    finishedProcesses.sort()

    finalExecutionTimes[dataBankIndex] = executionTime
    cpuUtilTemp= ((executionTime - wastedTime) / executionTime) * 100
    cpuUtilization[dataBankIndex] = round(cpuUtilTemp, 3)

    for i in range(8):
        for j in range(5):
            dataBanks[dataBankIndex][i][j]  = dataBank[i][j] # copies data from the data bank, which was gathered
during the algorithm, into data bank specifically made for that algorithm
            dataBank[i][j] = 0


    for q in finishedProcesses:
        avgTat += dataBanks[dataBankIndex][q - 1][4]  # Turnaround Time
        avgTr += dataBanks[dataBankIndex][q - 1][2]  # Response Time
        avgTwt += dataBanks[dataBankIndex][q - 1][3]  # Waiting Time

    avgTimes[dataBankIndex][2] = avgTat / len(finishedProcesses)  # Turnaround Time
    avgTimes[dataBankIndex][0] = avgTr  / len(finishedProcesses)  # Response Time
    avgTimes[dataBankIndex][1] = avgTwt / len(finishedProcesses)  # Waiting Time



def printData():


    # finishedProcesses.sort()

    print(
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t
╔══════════════════════════════════════════════════╗ \n'
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t   ║                  Final Analytics                 ║\n'
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t
╚══════════════════════════════════════════════════╝ \n'
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t   \n'
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t
╔════════════════╦════════════╦════════════╦════════════╗ \n'
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t   ║ Scheduling Type ║   SJF    ║    FCFS    ║    MLFQ    ║\n'
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t
╠════════════════╬════════════╬════════════╬════════════╣ \n'
       f'\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t   ║  AVG Wait Time  ║  {avgTimes[1][1]}  ║   {avgTimes[0][1]}   ║
{avgTimes[2][1]}   ║\n'
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t
╠════════════════╬════════════╬════════════╬════════════╣ \n'
       f'\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t   ║  AVG Turnaround ║  {avgTimes[1][2]}  ║   {avgTimes[0][2]}   ║
{avgTimes[2][2]}   ║\n'
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t
╠════════════════╬════════════╬════════════╬════════════╣ \n'
       f'\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t   ║  AVG Response   ║  {avgTimes[1][0]}  ║   {avgTimes[0][0]}   ║
{avgTimes[2][0]}     ║\n'
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t
╠════════════════╬════════════╬════════════╬════════════╣ \n'
       f'\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t   ║ CPU Utilization ║  {cpuUtilization[1]}  ║   {cpuUtilization[0]}   ║
```

```
{cpuUtilization[2]}     ║\n'
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t
╠══════════════════════╬══════════════╬════════════════╬═══════════════╣\n'
       f'\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t   ║   Total  Time   ║    {finalExecutionTimes[1]}   ║    {finalExecutionTimes[0]}
║     {finalExecutionTimes[2]}     ║\n'
        '\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t
╚══════════════════════╩══════════════╩════════════════╩═══════════════╝\n\n\n'







' ╔═══════════════════════════════════════════════════════════════════════╗ \t\t
╔═══════════════════════════════════════════════════════════════════════╗ \t\t
╔═══════════════════════════════════════════════════════════════════════╗ \n'
                 '║              First Come  First Serve              ║\t\t ║              Shortest Job First              ║\t\t ║
Multilevel Feedback Queue              ║\n'

'╠═══════════════════════════════════════════════════════════════════════╣ \t\t
╠═══════════════════════════════════════════════════════════════════════╣ \t\t
╠═══════════════════════════════════════════════════════════════════════╣ \n'
                 '  Process     Waiting Time  Turnaround Time  Response Time  \t\t    Process     Waiting Time
Turnaround Time  Response Time   \t\t    Process     Waiting Time  Turnaround Time  Response Time  '
    )
    for i in range(8):
        print(f' \t  P{i+1}           {dataBankFCFS[i][3]}            {dataBankFCFS[i][4]} \t\t\t    {dataBankFCFS[i][2]}
\t\t '
              f' \t  P{i+1}           {dataBankSJF [i][3]}            {dataBankSJF [i][4]} \t\t\t    {dataBankSJF [i][2]}   \t\t '
              f' \t  P{i+1}           {dataBankMLFQ[i][3]}            {dataBankMLFQ[i][4]} \t\t\t
{dataBankMLFQ[i][2]}   ')


print('═══════════════════════════════════════════════════════════════════════════════\t\t
══════════════════════════════════════════════════════════════════════════\t\t
══════════════════════════════════════════════════════════════════════════\n')


def prime():
    while readyQueue:
        fromReadyToCPU()
        fromCPUToIO()


def sortReadyQueue():
    if len(readyQueue) > 1:
        n = len(readyQueue)
        for i in range(n):
            already_sorted = True
            for j in range(n - i - 1):
                if processList[readyQueue[j] - 1].cpu_burst[0] > processList[readyQueue[j + 1] - 1].cpu_burst[0]:
```

```
                readyQueue[j], readyQueue[j + 1] = readyQueue[j + 1], readyQueue[j]
                already_sorted = False
            if already_sorted:
                break
    else:
        pass


def printOnContextSwitch():
    print('///////////////////////////////////////////////////////////\n'
          '//////////////    Context Switch    //////////////////////\n'
          '///////////////////////////////////////////////////////////')
    readyQueueWithTimes = []

    for i in readyQueue:
        readyQueueWithTimes.append(f'P{i}:{processList[i - 1].cpu_burst[0]}')
    print(f'Ready Queue: {readyQueueWithTimes}')


def resetDataBank():
    for i in range(8):
        for j in range(5):
            dataBank[i][j] = 0


FCFS()
SJF()
MLFQ()

printData()
```