CPU Scheduling Simulation Report

Hsin Ting Ho

Z-23500922

2019 FALL Operating System
PROJECT 1

Table of Content

INTRODUCTION

This report simulates 3 CPU scheduling algorithms, FCFS (non-preemptive), SJF(non preemptive), and MLFQ. CPU utilization, turnaround time, waiting time, and response time are calculated at the end of simulation.

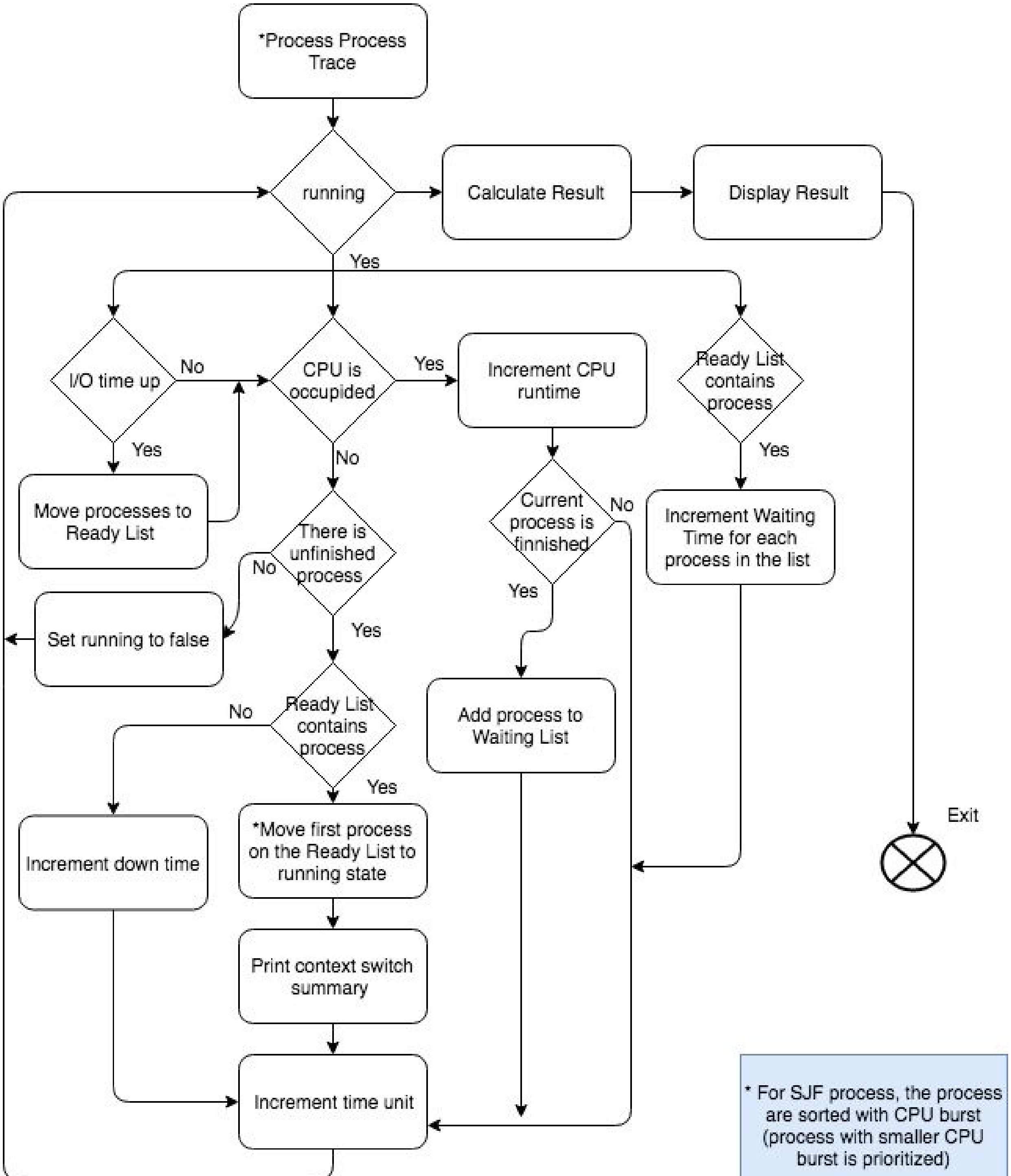
The logic of the simulation is presented with flowcharts, and the results of the simulation are shown with GANTT charts and tables. A brief discussion is then made based on the CPU utilization, turnaround time, waiting time, and response time.

For each algorithm, a snip of program output during context switch is included. Processes state change can be observed from the program output. A python source code is attached at the end of the report.

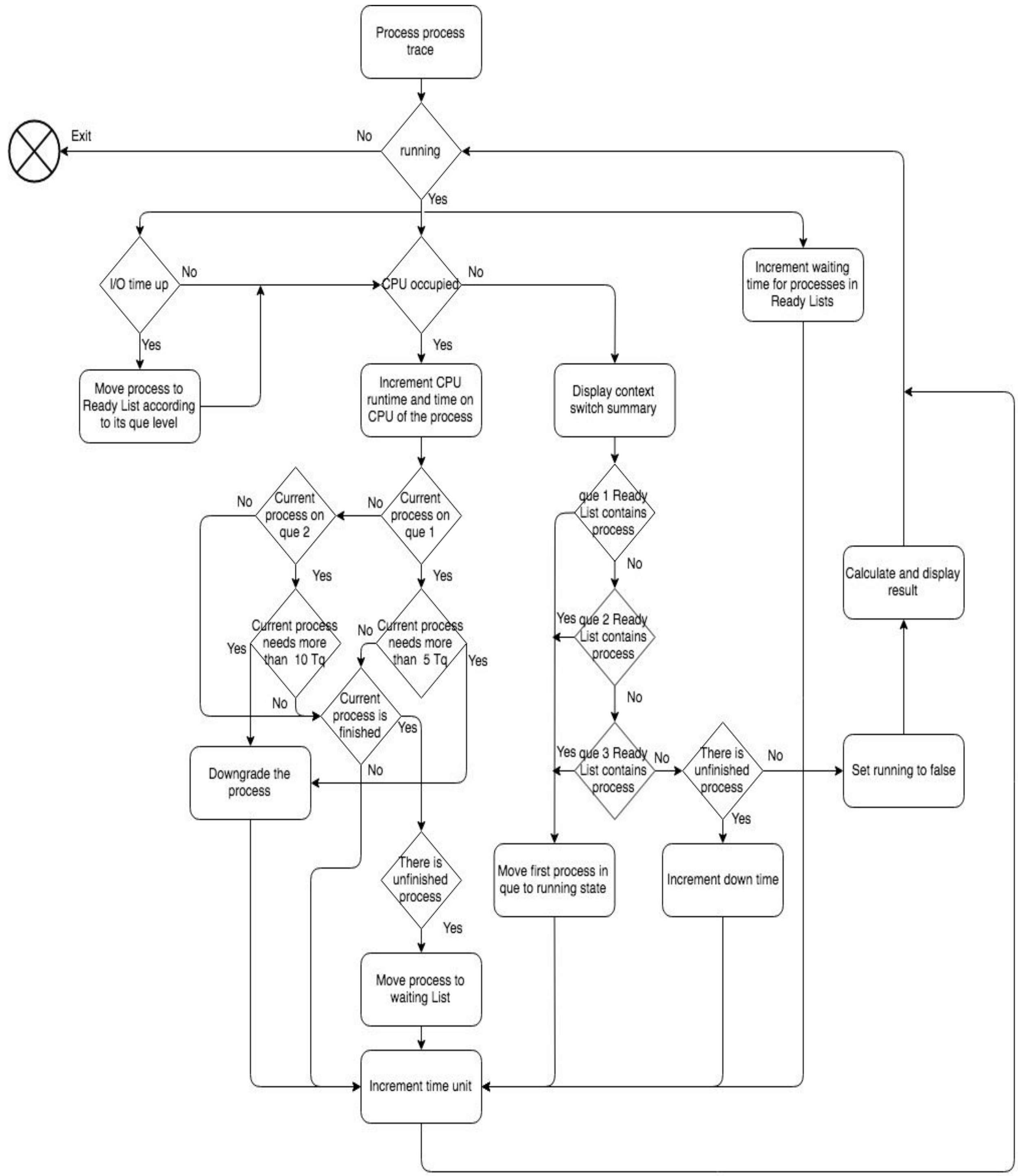
The simulation is made based on the following general rules and assumptions. For MLFQ, queue 1 uses Round Robin scheduling with 5 time quantum; queue 2 uses Round Robin scheduling with 10 time quantum; queue 3 uses First Come First Serve scheduling. If time quantum expires before CPU burst is completed, the process is downgraded to next lower level queue. Processes are not downgraded when preempted by a higher queue level process. For all three scheduling algorithms, all the processes are activated at time 0 and there is no waiting for I/O. There are total 8 processes with trace provided.

General Flowcharts

FCFS and SJF flowchart



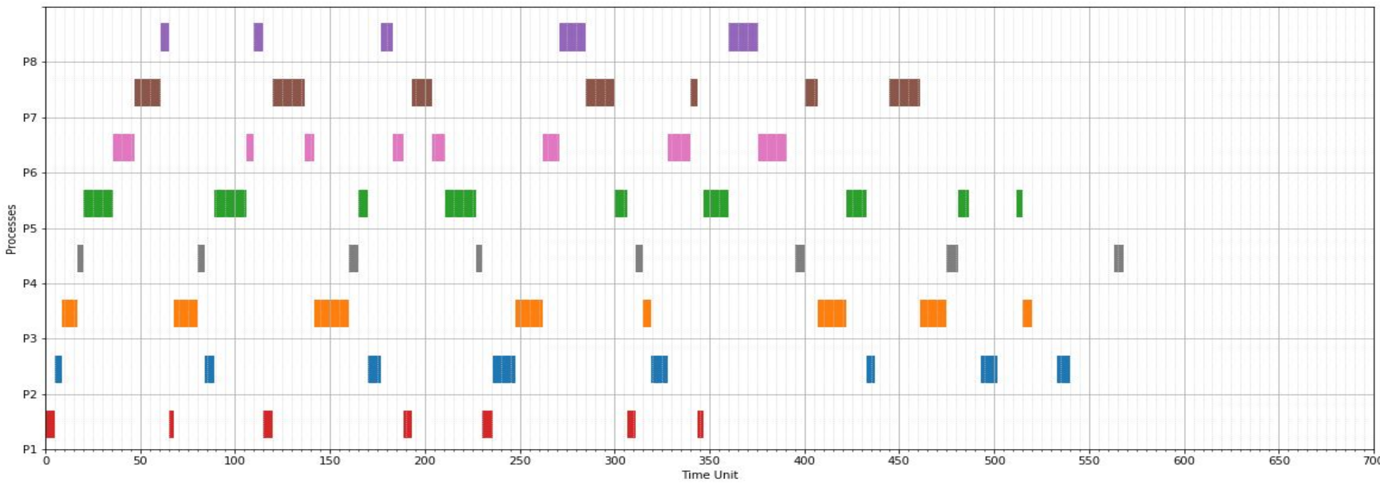
MLFQ flowchart



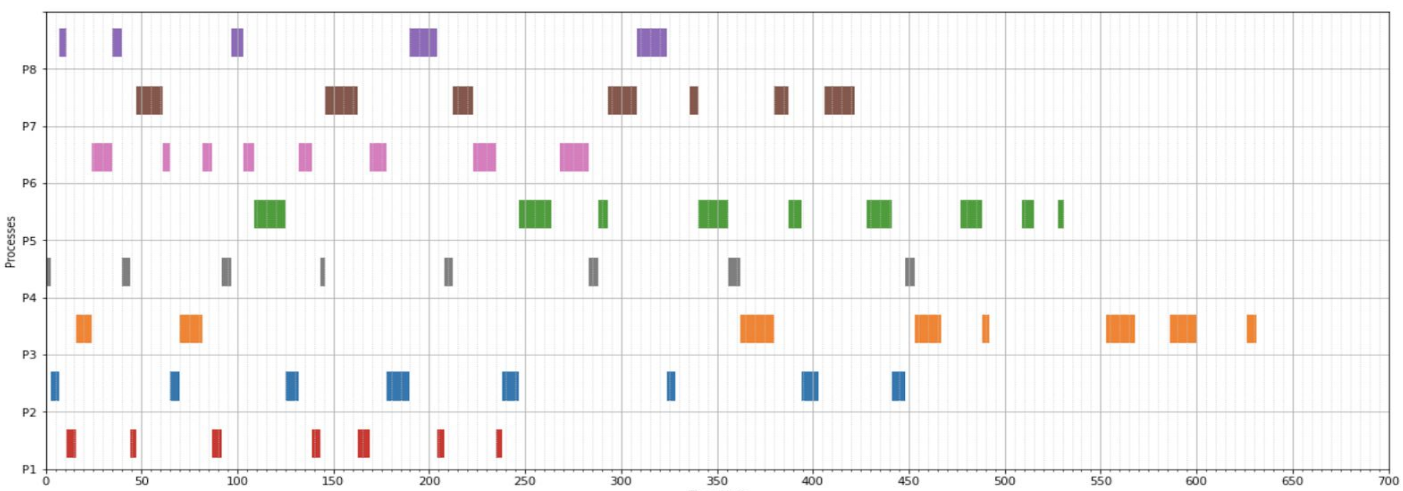
Process process trace
running
Exit
No
Yes
I/O time up
No
Yes
Move process to Ready List according to its que level
CPU occupied
No
Yes
Increment waiting time for processes in Ready Lists
Increment CPU runtime and time on CPU of the process
Display context switch summary
Current process on que 1
Current process on que 2
Current process needs more than 5 Tq
Current process needs more than 10 Tq
Current process is finished
Downgrade the process
There is unfinished process
Move process to waiting List
Increment time unit
que 1 Ready List contains process
que 2 Ready List contains process
que 3 Ready List contains process
There is unfinished process
Move first process in que to running state
Increment down time
Set running to false
Calculate and display result

GANTT chart

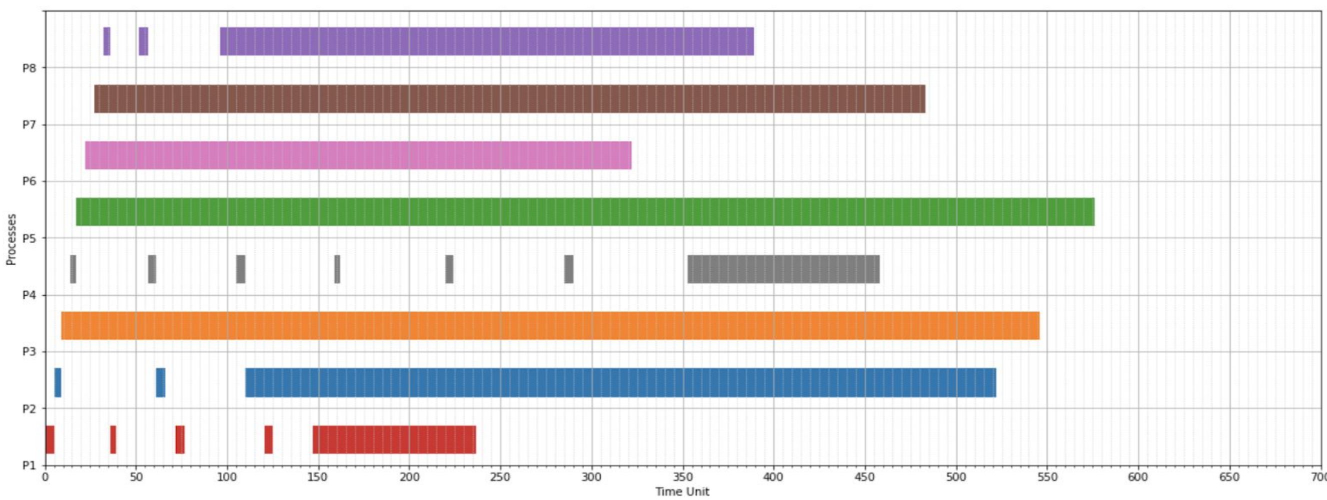
FCFS GANTT



SJF GANTT



MLFQ GANTT



Final Results

Table

	SJF	FCFS	MLFQ
CPU utilization	82.78%	85.34%	92.55%
Avg Waiting Time (Tw)	133.1	185.25	76.62
Avg Turnaround Time (Ttr)	469.6	521.37	497.1
Avg Response Time (Tr)	27.12	24.37	15.75

Processes Comparison

	SJF CPU utilization: 82.78%				FCFS CPU utilization: 85.34%				MLFQ CPU utilization: 92.55%			
	Tw	Ttr	Tr		Tw	Ttr	Tr		Tw	Ttr	Tr	
P1	43	268	11		170	395	0		69	265	0	
P2	73	500	3		164	591	5		51	573	5	
P3	276	668	16		165	557	9		56	581	9	
P4	50	534	0		164	648	17		75	549	14	
P5	237	546	109		220	530	20		37	591	17	
P6	119	336	24		229	445	36		149	397	22	
P7	148	477	47		184	512	47		102	532	27	
P8	119	428	7		184	493	61		74	489	32	
Avg	133.1	469.6	27.12		185.0	521.3	24.37		76.62	497.1	15.75	

Discussion

According to the simulation result, MLFQ has the best performance among all the algorithms used in this simulation. MLFQ yields the highest CPU utilization(92.55%), the lowest average response time (15.75 time units), and the lowest average waiting time (76.62 time units).

SJF and FCFS have similar performance on CPU utilization (82.78% and 85.34%). Although FCFS has shorter average response time(24.37 time units), SJF has shorter average turnaround time(469.6 time units) and average waiting time(133.1 time units).

Program Output

FCFS Sample Output

Current time: 0

Next Process on the CPU: P1

****

List of processes in the ready queue:

Processes	Burst	Queue
P2	4	Q1
P3	8	Q1
P4	3	Q1
P5	16	Q1
P6	11	Q1
P7	14	Q1
P8	4	Q1

****

List of processes in the I/O:

Processes	Remaining I/O time
[empty]	

****

****

Current time: 5

Next Process on the CPU: P2

****

List of processes in the ready queue:

Processes	Burst	Queue
P3	8	Q1
P4	3	Q1
P5	16	Q1
P6	11	Q1
P7	14	Q1
P8	4	Q1

****

List of processes in the I/O:

Processes	Remaining I/O time
P1	27

Current time: 583
Next Process on the CPU: P2

****

List of processes in the ready queue:

Processes	Burst	Queue

****

List of processes in the I/O:

Processes	Remaining I/O time
P4	62

****
****

Current time: 645
Next Process on the CPU: P4

****

List of processes in the ready queue:

Processes   Burst   Queue
****

List of processes in the I/O:

Processes   Remaining I/O time
[empty]

****
****

****
The Simulation is Ended.
Total time used: 648 time units
****

FCFS   Result Output

****

****

cpuUtil: 85.3395061728395

P1: Tw:170 / Ttr: 395 /Tr: 0

P2: Tw:164 / Ttr: 591 /Tr: 5

P3: Tw:165 / Ttr: 557 /Tr: 9

P4: Tw:164 / Ttr: 648 /Tr: 17

P5: Tw:220 / Ttr: 530 /Tr: 20

P6: Tw:229 / Ttr: 445 /Tr: 36

P7: Tw:184 / Ttr: 512 /Tr: 47

P8: Tw:184 / Ttr: 493 /Tr: 61

avgTtr: 521.375

avgTw: 185.0

avgTr: 24.375

Total Time: 648

SJF Sample Output

Current time: 0
Next Process on the CPU: P4

****

List of processes in the ready queue:

Processes	Burst	Queue
P2	4	Q1
P8	4	Q1
P1	5	Q1
P3	8	Q1
P6	11	Q1
P7	14	Q1

Processes	Burst	Queue
P5	16	Q1

****

List of processes in the I/O:

Processes	Remaining I/O time
[empty]	

****
****

Current time: 3
Next Process on the CPU: P2

****

List of processes in the ready queue:

Processes	Burst	Queue
P8	4	Q1
P1	5	Q1
P3	8	Q1
P6	11	Q1
P7	14	Q1
P5	16	Q1

****

List of processes in the I/O:

Processes	Remaining I/O time
P4	35

****
****

Current time: 7

Next Process on the CPU: P8

****

List of processes in the ready queue:

Processes	Burst	Queue
P1	5	Q1
P3	8	Q1
P6	11	Q1
P7	14	Q1
P5	16	Q1

****

List of processes in the I/O:

Processes	Remaining I/O time
P4	31
P2	48

Current time: 11
Next Process on the CPU: P1

****

List of processes in the ready queue:

Processes	Burst	Queue
P3	8	Q1
P6	11	Q1
P7	14	Q1
P5	16	Q1

****

List of processes in the I/O:

Processes	Remaining I/O time

P8 14
P4 27
P2 44

****
****

Current time: 626
Next Process on the CPU: P3

****

List of processes in the ready queue:

Processes Burst Queue
****

List of processes in the I/O:

Processes Remaining I/O time
[empty]

****
****

Current time: 662
Next Process on the CPU: P3

****

List of processes in the ready queue:

Processes Burst Queue
****

List of processes in the I/O:

Processes   Remaining I/O time
[empty]

****
****

****
The Simulation is Ended.
Total time used: 668 time units
****

SJF Result Output

cpuUtil: 82.78443113772454
P1: Tw:43 / Ttr: 268 /Tr: 11
P2: Tw:73 / Ttr: 500 /Tr: 3
P3: Tw:276 / Ttr: 668 /Tr: 16
P4: Tw:50 / Ttr: 534 /Tr: 0
P5: Tw:237 / Ttr: 546 /Tr: 109
P6: Tw:119 / Ttr: 336 /Tr: 24
P7: Tw:148 / Ttr: 477 /Tr: 47
P8: Tw:119 / Ttr: 428 /Tr: 7
avgTtr: 469.625
avgTw: 133.125
avgTr: 27.125
Total Time: 668

MLFQ Sample Output

Current time: 0
Next Process on the CPU: P1

****

List of processes in the ready queue:

Processes	Burst	Queue
P2	4	Q1
P3	8	Q1
P4	3	Q1
P5	16	Q1
P6	11	Q1
P7	14	Q1
P8	4	Q1

****

List of processes in the I/O:

Processes   Remaining I/O time
[empty]

****
****

Current time: 5
Next Process on the CPU: P2

****

List of processes in the ready queue:

Processes   Burst   Queue

P3	8	Q1
P4	3	Q1
P5	16	Q1
P6	11	Q1
P7	14	Q1
P8	4	Q1

****

List of processes in the I/O:

Processes	Remaining I/O time
P1	27

****
****

Current time: 9
Next Process on the CPU: P3

****

List of processes in the ready queue:

Processes	Burst	Queue
P4	3	Q1
P5	16	Q1
P6	11	Q1
P7	14	Q1
P8	4	Q1

****

List of processes in the I/O:

Processes	Remaining I/O time
P1	23
P2	48

Current time: 587
Next Process on the CPU: P5

****

List of processes in the ready queue:

Processes   Burst   Queue
****

List of processes in the I/O:

Processes   Remaining I/O time
P3   25

****
****

Current time: 591
Next Process on the CPU: None

****

List of processes in the ready queue:

Processes   Burst   Queue
[empty]

****

List of processes in the I/O:

Processes   Remaining I/O time
P3   21

****

****

****
The Simulation is Ended.
Total time used: 591 time units
****

MLFQ Result Output

cpuUtil: 92.55499153976311
P1: Tw:69 / Ttr: 265 /Tr: 0
P2: Tw:51 / Ttr: 573 /Tr: 5
P3: Tw:56 / Ttr: 581 /Tr: 9
P4: Tw:75 / Ttr: 549 /Tr: 14
P5: Tw:37 / Ttr: 591 /Tr: 17
P6: Tw:149 / Ttr: 397 /Tr: 22
P7: Tw:102 / Ttr: 532 /Tr: 27
P8: Tw:74 / Ttr: 489 /Tr: 32
avgTtr: 497.125
avgTw: 76.625
avgTr: 15.75
Total Time: 591

Source Code

```
class Task: #A process is divided into tasks by CPU bust time in the trace
    ProcessName = ''
    StartTime = 0 #The time I/O is finished
    ProcessTime = 0 #CPU burst
    IOtime = 0
    TimeOnCPU = 0 #used in MLFQ to record the amount of time has been spend on CPU
    QueLevel = 0 #All process starts at Q1
    DownGraded =False #Marked to true when downgraded to a lower queue
    CPUburst = 0 #Used as a copy of Process Time in MLFQ
    def __init__(self, start, process, io, pName): #initialize a task
        self.StartTime = start
        self.ProcessTime = process
        self.IOtime = io
        self.ProcessName = pName
        self.QueLevel = 1
        self.CPUburst = process
        print('ProcessName: '+pName)


def addtoReadySorted(readyList, P):#add to readyList with ascending CPU burst
    index = 0

    if len(readyList) == 0:
```

```
        print('adding to empty list')
        readyList.append(P)
    else:

        for w in readyList:
            #print('w '+str(w.StartTime))
            if P.ProcessTime < w.ProcessTime:    #if process start time is smaller, insert
                index = readyList.index(w)
                readyList.insert(index, P)
                break
            elif w == readyList[-1]:      #if process start time is the largest, append
                readyList.append(P)
                break
            else:
                continue
    return readyList

def addtoListSorted(anyList, P):#add to readyList with ascending StartTime
    index = 0
    print('adding '+P.ProcessName+' to waitingList')
    if len(anyList) == 0:
        print('adding to empty list')
        anyList.append(P)
    else:
```

```
        for w in anyList:
            #print('w '+str(w.StartTime))
            if P.StartTime < w.StartTime:    #if process start time is smaller, insert
                index = anyList.index(w)
                anyList.insert(index, P)
                break
            elif w == anyList[-1]:      #if process start time is the largest, append
                anyList.append(P)
                break
            else:
                continue
    return anyList
```

```
def makeProcessStack(traceDic):#Divide process trace by its CPU burst. Store each CPU burst, I/O in a Task object.
    processStackList = []      #Store all the tasks in order in a dictionary. use process name as key.
    cpu = io = 0
    for P in traceDic:
        processStack = []
        trace= traceDic[P]
        last = len(trace)-1
        count = 0
```

```
        for time in trace:
            if count%2 == 0:
                cpu = time
                if count == last:
                    task =  Task(0, cpu, 0, P)
                    processStack.append(task)
            else:
                io = time
                task =  Task(0, cpu, io, P)
                processStack.append(task)
            count += 1
        processStackList.append(processStack)
    return processStackList#processStackList = [[{P1,cpu, io}...],[{P2,cpu, io}...],...,[{P8, cpu, io}]]


def display(CPUruntime, timeUnit, CurrentProcess, waitingList, readyList):#Display info for each context switch
    print('Current time: '+str(timeUnit))
    if len(readyList)>0:
        print('Next Process on the CPU: '+ readyList[0].ProcessName)
    else:
        print('Next Process on the CPU: None')
    print('\n******************************************************************\n')
    print('List of processes in the ready queue:\n')
```

```
    print(' Processes    Burst    Queue')
    if len(readyList) == 0:
        print('    '+'[ empty ]\n')
    else:
        for r in readyList[1:]:
            print('    '+r.ProcessName+'        '+str(r.ProcessTime)+'        Q'+str(r.QueLevel))
    print('******************************************************************\n')
    print('List of processes in the I/O:\n')
    print(' Processes    Remaining I/O time')
    if len(waitingList) == 0:
        print('    '+'[ empty ]\n')
    else:
        for w in waitingList:
          ioRemain = w.StartTime-timeUnit
          print('    '+w.ProcessName+'        '+str(ioRemain))
    print('\n******************************************************************')
    print('******************************************************************\n')

def calcResult(timeUnit, downTime, cpuTime, startTimeDic, finishTimeDic, WaitTimeDic):#Calculat average turnaround time, average waitting time, and average response time and store result in a dictionary
    resultDic = {}
    Ttr =0
    totalTtr = 0
```

```
    waitTime = 0

    resTime = 0

    resultDic['cpuUtil'] = (float(cpuTime)/float(timeUnit))*100

    for key in startTimeDic: #add part to handle a list of start times and a list of end times

        if len(startTimeDic[key]) > 1:

            Ttr = finishTimeDic[key][-1]#-startTimeDic[key][1]

            resultDic[key] = 'Tw:'+ str(WaitTimeDic[key]) + ' / Ttr: '+str(Ttr) + ' /Tr: '+
str(startTimeDic[key][1])

            totalTtr += Ttr

            resTime += startTimeDic[key][1]

        waitTime += WaitTimeDic[key]


    resultDic['avgTtr'] = float(totalTtr)/8.0

    resultDic['avgTw']=float(waitTime)/8.0

    resultDic['avgTr']=float(resTime)/8.0

    resultDic['Total Time'] = timeUnit

    return resultDic #keys = [cpuUtil, PnTtr, avgTtr, avgTw]


def FCFS(traceDic):

        #store the start time of each task

        StartTimeDic = {'P1':[0],'P2':[0],'P3':[0],'P4':[0],'P5':[0],'P6':[0],'P7':[0],'P8':[0]}

        #store the end time of each task

        EndTimeDic = {'P1':[0],'P2':[0],'P3':[0],'P4':[0],'P5':[0],'P6':[0],'P7':[0],'P8':[0]}

        #store the wait time of each task
```

```
WaitTimeDic = {'P1':0,'P2':0,'P3':0,'P4':0,'P5':0,'P6':0,'P7':0,'P8':0}

timeUnit = CPUruntime = downTime = processIndex = tempStartTime = waitTime = 0

processStackList= makeProcessStack(traceDic)

readyList = []

waitingList = []

finishList = []

resultDic = {}

running = True

#add first task of every process to readyList and set current task to none

for PS in processStackList:

    PS[0].StartTime = tempStartTime

    tempStartTime += PS[0].ProcessTime

    readyList.append(PS[0])

    PS.pop(0)

current = None

while running:

    #Move process to readyList

    if len(waitingList)>0 and waitingList[0].StartTime <= timeUnit:

        for w in waitingList:

            if w.StartTime <= timeUnit:

                readyList.append(w)

                waitingList.remove(w)


    if not current:#When CPU is not occupied
```

```
            if processStackList or len(readyList) > 0 or len(waitingList)>0:
                if len(readyList) > 0:
                    current = readyList[0] #Move the first process in readyList to running state
                    display(CPUruntime, timeUnit, current, waitingList, readyList)
                    current.StartTime = timeUnit
                    readyList.pop(0)
                    StartTimeDic.get(current.ProcessName).append(timeUnit)
                else:#If the readyList is empty, increament down time
                    downTime += 1
            else:
                running = False #stop the loop
                resultDic = calcResult(timeUnit, downTime, CPUruntime, StartTimeDic,
EndTimeDic, WaitTimeDic) #calculate results
                for key in resultDic:
                    print(key+': '+str(resultDic[key])) #display results
        else:#When CPU is occupied
            CPUruntime += 1
            if timeUnit == (current.StartTime + current.ProcessTime):#if the running process
is finished
                EndTimeDic[current.ProcessName].append(timeUnit)#record the end time
                index = int(current.ProcessName[-1])-1
                try:#handle error thrown when the current process is the last task
                    nextTaskInProcess = processStackList[index][0]
                    nextTaskInProcess.StartTime = timeUnit + current.IOtime
```

```
                waitingList = addtoListSorted(waitingList, nextTaskInProcess)
                processStackList[index].pop(0)
            except:
                count = 0
                for stack in processStackList:#if all the tasks are finished, wipe out the
processStackList
                    if len(stack) == 0:
                        count += 1
                        if count == len(processStackList):
                            processStackList.clear()
            current = None
            continue
    timeUnit += 1
    #acumulating waiting time for each process in WaitTimeDic
    if len(readyList)>0:
        for r in readyList:
            WaitTimeDic[r.ProcessName] += 1


def SJF(traceDic):
    StartTimeDic = {'P1':[0],'P2':[0],'P3':[0],'P4':[0],'P5':[0],'P6':[0],'P7':[0],'P8':[0]}
    EndTimeDic = {'P1':[0],'P2':[0],'P3':[0],'P4':[0],'P5':[0],'P6':[0],'P7':[0],'P8':[0]}
    WaitTimeDic = {'P1':0,'P2':0,'P3':0,'P4':0,'P5':0,'P6':0,'P7':0,'P8':0}
    timeUnit = CPUruntime = downTime = processIndex = tempStartTime = waitTime = 0
    processStackList= makeProcessStack(traceDic)
```

```
    readyList = []
    waitingList = []
    finishList = []
    resultDic = {}
    running = True
    #add first task of every process to readyList and set current task to none
    for PS in processStackList:
        PS[0].StartTime = tempStartTime
        tempStartTime += PS[0].ProcessTime
        readyList = addtoReadySorted(readyList, PS[0])
        PS.pop(0)
    current = None
    while running:
        if len(waitingList)>0 and waitingList[0].StartTime <= timeUnit:
            readyList = addtoReadySorted(readyList,waitingList[0]) #Move the process to readyList and sort the list by CPU burst time
            waitingList.pop(0)
        if not current:#CPU not occupied
            if processStackList or len(readyList) > 0 or len(waitingList)>0:
                if len(readyList) > 0:
                    current = readyList[0]#Move the first process on readyList to running state
                    display(CPUruntime, timeUnit, current, waitingList, readyList)#display context switch info
                    current.StartTime = timeUnit
```

```
                readyList.pop(0)
                StartTimeDic.get(current.ProcessName).append(timeUnit)#store the start time
            else:
                downTime += 1
        else:
            running = False
            resultDic = calcResult(timeUnit, downTime, CPUruntime, StartTimeDic, EndTimeDic, WaitTimeDic)
            for key in resultDic:
                print(key+': '+str(resultDic[key]))
    else:#CPU is occupied
        CPUruntime += 1
        if timeUnit == (current.StartTime + current.ProcessTime):#the process on running state is finished
            EndTimeDic[current.ProcessName].append(timeUnit)
            index = int(current.ProcessName[-1])-1
            try:#handle the error thrown at the end of process
                nextTaskInProcess = processStackList[index][0]
                nextTaskInProcess.StartTime = timeUnit + current.IOtime
                waitingList = addtoListSorted(waitingList, nextTaskInProcess)
                processStackList[index].pop(0)
            except:
                count = 0
                for stack in processStackList:#when all the process are finished, wipe out the list
```

```
                if len(stack) == 0:
                    count += 1
                    if count == len(traceDic):
                        processStackList.clear()
            current = None
            continue
        timeUnit += 1
        #acumulating waiting time for each process in WaitTimeDic
        if len(readyList)>0:
            for r in readyList:
                WaitTimeDic[r.ProcessName] += 1


def MLFQ(traceDic):
    StartTimeDic = {'P1':[0],'P2':[0],'P3':[0],'P4':[0],'P5':[0],'P6':[0],'P7':[0],'P8':[0]}
    EndTimeDic = {'P1':[0],'P2':[0],'P3':[0],'P4':[0],'P5':[0],'P6':[0],'P7':[0],'P8':[0]}
    WaitTimeDic = {'P1':0,'P2':0,'P3':0,'P4':0,'P5':0,'P6':0,'P7':0,'P8':0}
    timeUnit = CPUruntime = downTime = processIndex = tempStartTime = waitTime = 0
    processStackList= makeProcessStack(traceDic)
    Q1readyList = []
    Q2readyList = []
    Q3readyList = []
    readyListDisplay = []
    waitingList = []
    resultDic = {}
```

```
#put all the first task to Q1readyList
for PS in processStackList:
    PS[0].StartTime = tempStartTime
    tempStartTime += PS[0].ProcessTime
    Q1readyList.append(PS[0])
    PS.pop(0)
current = None
running = True

while running:
    if len(waitingList) > 0:
        for w in waitingList:
            if w.StartTime > timeUnit:#Move process to its ready queue
                break
            if w.StartTime <= timeUnit:
                if w.QueLevel == 1:
                    Q1readyList.append(w)
                elif w.QueLevel == 2:
                    Q2readyList.append(w)
                else:
                    Q3readyList.append(w)
            waitingList.remove(w)
    if current:#cpu is occupied
        CPUruntime +=1
```

```
        current.TimeOnCPU += 1
        if current.QueLevel == 1:
            if current.TimeOnCPU == 5 and current.ProcessTime >5:#downgrade the process when 5 quantam are used and the process is not finished
                print('*** DownGrading '+current.ProcessName + ' to Q2')
                current.ProcessTime -= 5
                current.QueLevel = 2
                current.DownGraded = True
                Q2readyList.append(current)
                current = None
                continue
        elif current.QueLevel == 2:
            #downgrade the process when 10 quantam are used and the process is not finished
            if (current.DownGraded and current.TimeOnCPU == 15) or (not current.DownGraded and current.TimeOnCPU ==10) and current.ProcessTime > 10:
                current.ProcessTime -= 10
                current.QueLevel = 3
                current.DownGraded = True
                Q3readyList.append(current)
                current = None
                continue

        if current.CPUburst == current.TimeOnCPU:#when the CPU burst is finished
            current.DownGraded = False
```

```
            index = int(current.ProcessName[-1]) -1
            try:#handle the error thrown at the end of the process
                nextTask = processStackList[index][0]
                nextTask.StartTime = timeUnit + current.IOtime
                nextTask.QueLevel = current.QueLevel
                waitingList = addtoListSorted(waitingList, nextTask)
                processStackList[index].pop(0)
            except:
                count = 0
                for stack in processStackList:#wipe out the list when all the processes are finished
                    if len(stack) == 0:
                        count += 1
                        if count == len(traceDic):
                            processStackList.clear()
            EndTimeDic[current.ProcessName].append(timeUnit)#record the end time of the current process
            current = None
            continue

        else:#CPU is not occupied
            readyListDisplay = Q1readyList + Q2readyList + Q3readyList #join all the readyLists for display
            display(CPUruntime, timeUnit, current, waitingList, readyListDisplay)
```

```
#Move process from readyLists to running state according to its priority
if len(Q1readyList)>0:
    current = Q1readyList[0]
    current.StartTime = timeUnit
    StartTimeDic[current.ProcessName].append(timeUnit)
    Q1readyList.pop(0)
elif len(Q2readyList)>0:
    current = Q2readyList[0]
    current.StartTime = timeUnit
    StartTimeDic[current.ProcessName].append(timeUnit)
    Q2readyList.pop(0)
elif len(Q3readyList)>0:
    current = Q3readyList[0]
    current.StartTime = timeUnit
    StartTimeDic[current.ProcessName].append(timeUnit)
    Q3readyList.pop(0)
else:
    if processStackList:
        downTime += 1
    else:
        running = False
        resultDic = calcResult(timeUnit, downTime, CPUruntime, StartTimeDic,
EndTimeDic, WaitTimeDic)
        for key in resultDic:
```

```
                print(key+': '+str(resultDic[key]))
            continue
    #increment waiting time for all the processes in the readyLists
    if len(Q1readyList) > 0:
        for r in Q1readyList:
            WaitTimeDic[r.ProcessName] += 1
    if len(Q2readyList) > 0:
        for r in Q2readyList:
            WaitTimeDic[r.ProcessName] += 1

    if len(Q3readyList) > 0:
        for r in Q2readyList:
            WaitTimeDic[r.ProcessName] += 1

    timeUnit += 1


def main():
    traceDic = {'P1':[5, 27, 3, 31, 5, 43, 4, 18, 6, 22, 4, 26, 3, 24, 4],
            'P2':[4, 48, 5, 44, 7, 42, 12, 37, 9, 76, 4, 41, 9, 31, 7, 43, 8],
            'P3':[8, 33, 12, 41, 18, 65, 14, 21, 4, 61, 15, 18, 14, 26, 5, 31, 6],
            'P4':[3, 35, 4, 41, 5, 45, 3, 51, 4, 61, 5, 54, 6, 82, 5, 77, 3],
            'P5':[16, 24, 17, 21, 5, 36, 16, 26, 7, 31, 13, 28, 11, 21, 6, 13, 3, 11, 4],
            'P6':[11, 22, 4, 8, 5, 10, 6, 12, 7, 14, 9, 18, 12, 24, 15, 30, 8],
```

```
        'P7':[14, 46, 17, 41, 11, 42, 15, 21, 4, 32, 7, 19, 16, 33, 10],
        'P8':[4, 14, 5, 33, 6, 51, 14, 73, 16, 87, 6]}

  FCFS(traceDic)
  SJF(traceDic)
  MLFQ(traceDic)
main()#run the simulation
```