5913262 SIWAPORN C.

OPERATING SYSTEM

CPU SCHEDULING SIMULATION

CPU SCHEDULING

FCFS

SJF

PRIORITY

ROUND ROBIN

Average waiting time

$$= \frac{3 + 16 + 9 + 0}{4} = 7 \text{ ms}$$

F C F S

```
calculateWaitingTime() {
var current = 0
var pCount : Int = processNo.count()
for (i : Int in 0..pCount - 1) {
    val wait: Int = current - arrivalTime[i]
    waitingTime.add(wait)
    current+=cpuBT[i]
}


averageWaitingTime(): Double{
val sum : Int = waitingTime.sum()
val average: Double = (sum / waitingTime.count()).toDoubl
return average


chart() {
var result = ""
var scale = ""
var current = 0
var used: ArrayList<Int> = ArrayList()
var pCount : Int = processNo.count()
for (i : Int in 0..pCount - 1) {
    val half : Int = cpuBT[i] / 2
    for (j : Int in 0..half - 1) {
        result = result + "-"
    }
    result.plus(i)
    for (j : Int in 0..half - 1) {
        result = result + "-"
    }
    current += cpuBT[i]
    used.add(current)
}
```

FCFS

CALCULATING

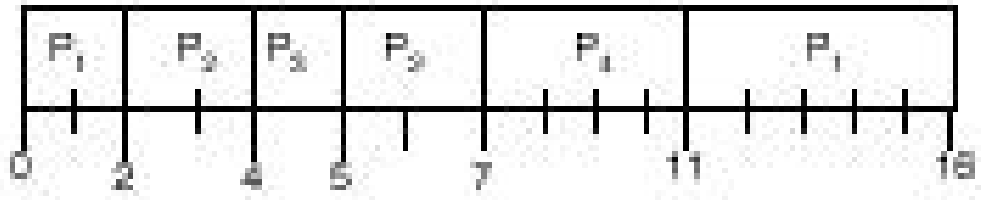
```
Arrival Time
0 1 2 3
Process Number
1 3 2 4
CPU Burst Time
10 5 8 3
Waiting Time: [0, 9, 13, 20]
Average Waiting Time: 10.0
------------------------
0          10      15         23    26

Process finished with exit code 0
```

FCFS

INPUT & OUTPUT

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



average waiting time = (9 + 1 + 0 +2)/4 = 3

S J F

```
        return 0
    }
    val lowKey : Int? = list.min()
    val position : Int = list.indexOf(lowKey)
    return position


for (i : Int in 0..totalTime) {
    if (i < processNo) {
        list.add(cpuBT[i])
    }
    val lowIndex : Int = lowest()

    if (cpuBT[lowIndex] != 0) {
        used.add(lowIndex)
        cpuBT[lowIndex]= lowIndex - 1
        finishedTime[lowIndex]= (i + 1)
    }

    if (cpuBT[lowIndex] == 0) {
        list[lowIndex] = 999
    }


fun calculateWaitingTime(): Double {
    for (i : Int in 0..(processNo.toDouble() - 1).toInt()) {
        waitingTime[i] = finishedTime[i] - cpuBackUp[i] - i
    }
    var sum = 0
    for (j : Int in 0..waitingTime.size) {
        sum += j
    }
    return (sum / processNo).toDouble()
```

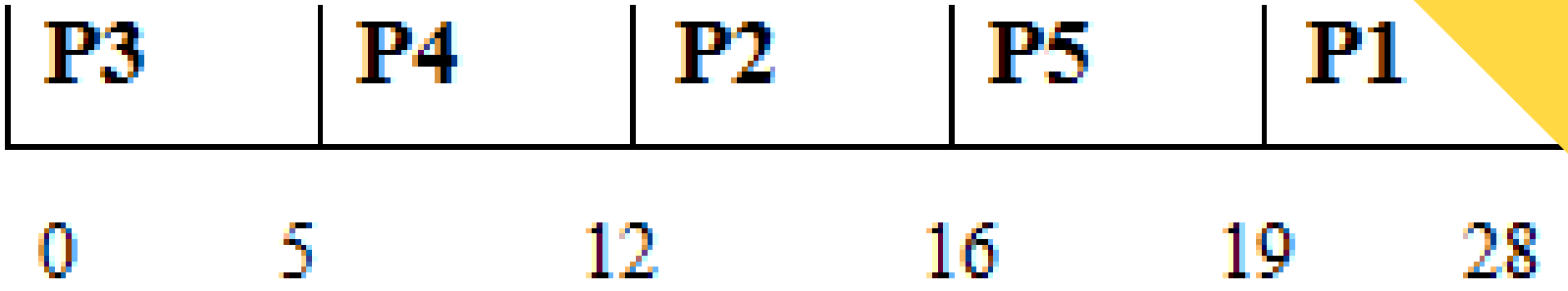
SJF

CALCULATING

```
Process Number: 1 2 3 4 5
Burst time: 2 4 7 6 8
Waiting time: 0 1 10 3 15
Average waiting time = 5.8

Process finished with exit code 0
```

SJF

INPUT & OUTPUT



T.W.T= 19 + 12 + 0 + 5 + 16 =52 mills
A.W.T = 52 / 5 = 10.4 mills
T.T.T = 28 + 16 + 5 + 12 + 19 = 80 mills.
A.T.T =80 / 5 = 16 mills.

PRIORITY SCHEDULING

```
println("CPU Priority")
var prioList :List<Int>  = readLine()!!.split( ...delimite

var cpuBackUp :List<Int>  = cpuBT
var finishedTime :MutableList<Int!> = Collections.nCopi
var waitingTime :MutableList<Int!> = Collections.nCopie

for (i :Int  in cpuBT) {
    totalTime += 1
}

fun lowest(): Int{
    if (list.count() == 1) {
        return 0
    }
    val lowKey :Int?  = list.min()
    val position :Int  = list.indexOf(lowKey)
    return position
}

for (i :Int  in 0..totalTime) {
    if (i < processNo) {
        list.add(prioList[i])
    }
    val lowIndex :Int  = lowest()

    if (cpuBT[lowIndex] != 0) {
        used.add(lowIndex)
        cpuBT[lowIndex].plus( other: lowIndex -1)
        finishedTime[lowIndex].plus( other: i + 1)
    }
```

PRIORITY SCHEDULING

CALCULATING

```
Process No:  Waiting Time:
1                0
2                1
3                3
4                12
5                24

Average Wating Time : 8.0

Process finished with exit code 0
```

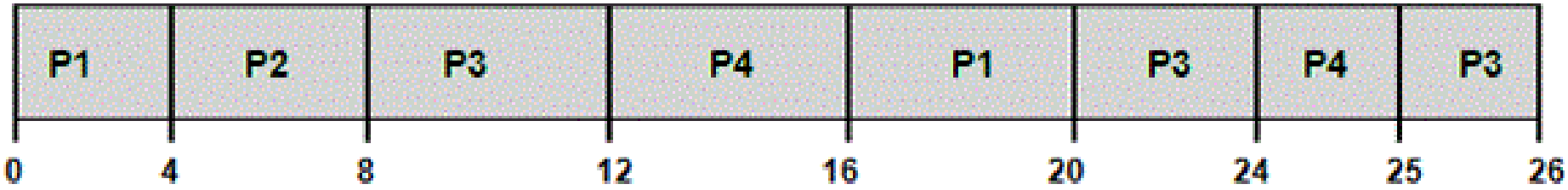
PRIORITY SCHEDULING

INPUT & OUTPUT

Process	Arrival Time	Service Time
1	0	8
2	1	4
3	2	9
4	3	5

Note.
Example violates
quantum size since
processes don't finish in
quantum.

Round Robin, quantum = 4, no priority-based preemption



Average wait = ((20-0) + (8-1) + (26-2) + (25-3))/4 = 74/4 = 18.5

ROUND ROBIN

```
un last() {
    var temp2 = -1
    for (i :Int in 0..used.size - 1){
        val temp :Int = used[i]
        if (lastNo.contains(temp) && (temp2 != used[i])) {
            lastNo[temp] = i
        } else if (temp2 == used[i]) {

        } else {
            lastNo[temp] = i
            temp2 = used[i]
        }
    }


un calculateWaitingTime(): Double {
    var sum = 0.0
    for (i :Int in 0..finished.size - 1) {
        val temp:Double = (finishedTime[i] - cpuHistory[i].
        waitingTime.add(temp)
    }
    for (i :Int in 0..waitingTime.size - 1) {
        sum += waitingTime[i]
    }
    val temp: Double = (waitingTime.size).toDouble()
    return sum / temp


un realTime(endTime: Int, number: Int): Int {
    var temp = 0
    for (i :Int in 0..endTime) {
        if (number == used[i]) {
            temp = i
        }
```

ROUND ROBIN

CALCULATING

```
Process: 1 2 3 4 5
CPU Burst time: 5 10 7 6 2
Waiting time: 14 20 19 22 16
Average waiting time = 54.6

Process finished with exit code 0
```


ROUND ROBIN

INPUT & OUTPUT

THANK YOU!