
# O's – Scheduling

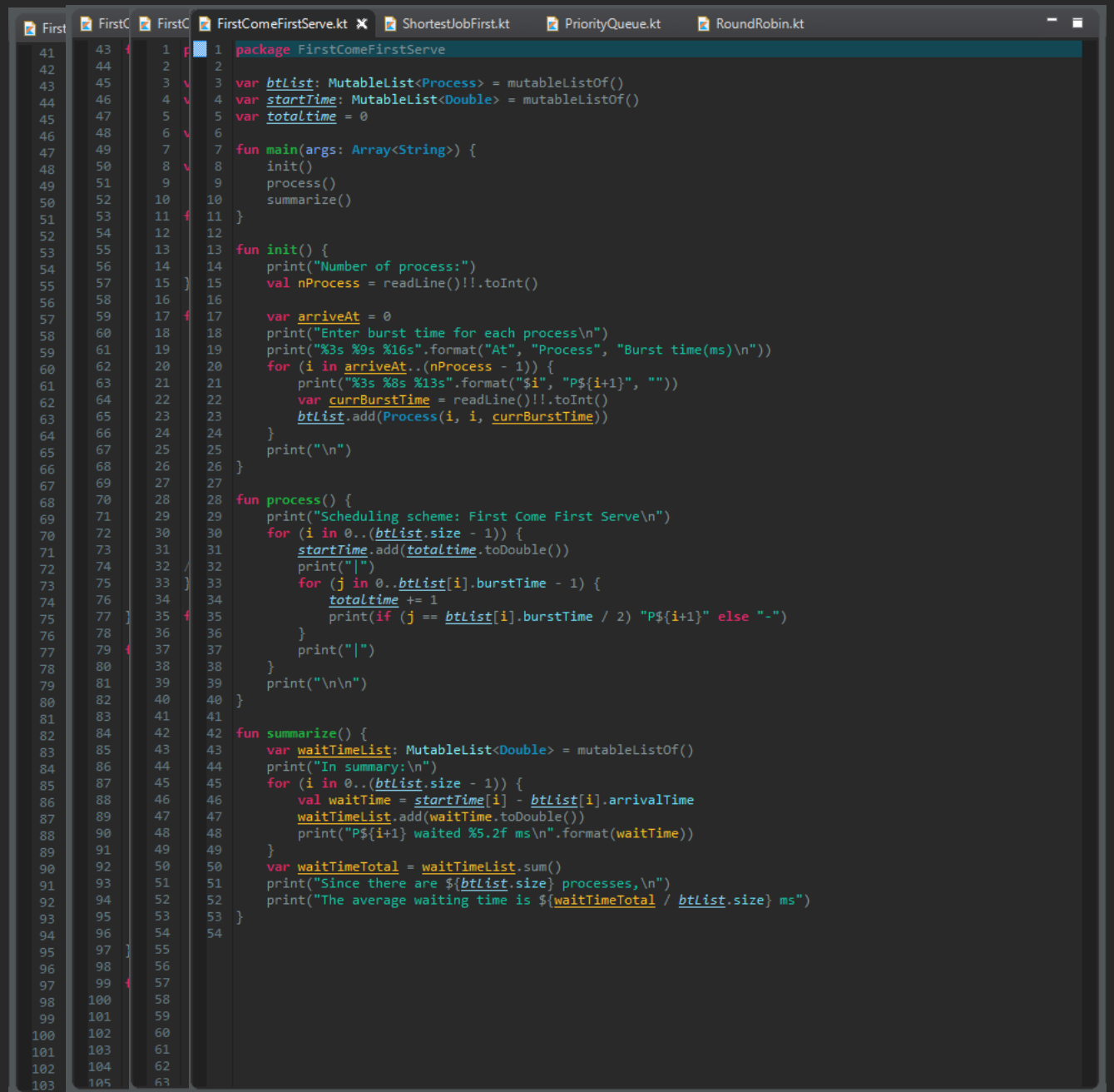
Using Kotlin

via Eclipse Java Neon

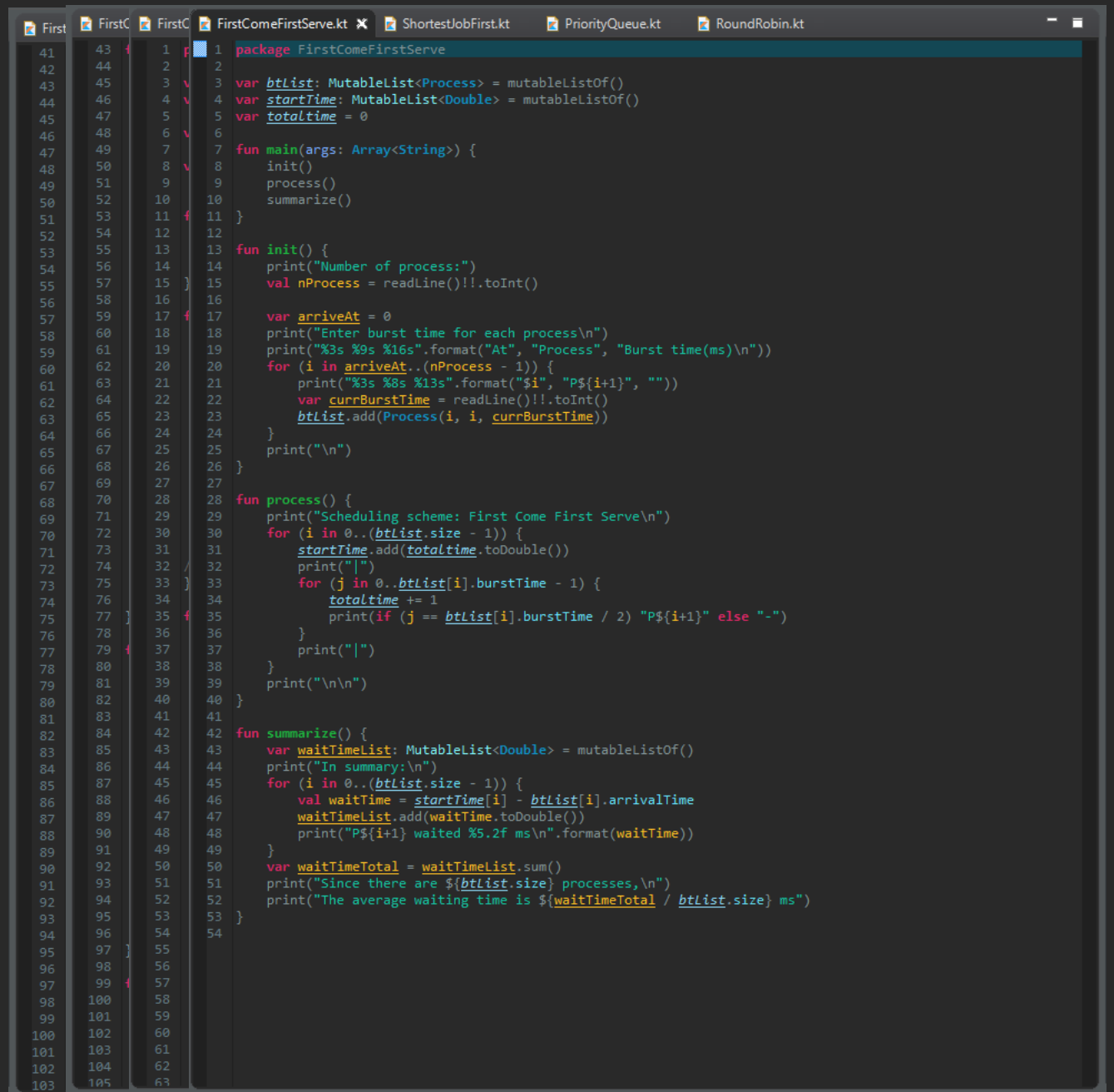
To Simulate How

‘Os’

Schedules processes




1.First Come First Serve




The key

is to execute the process

in order of their arrival

FirstComeFirstServe.kt | ShortestJobFirst.kt | PriorityQueue.kt | RoundRobin.kt

```kotlin
package FirstComeFirstServe

var btList: MutableList<Process> = mutableListOf()
var startTime: MutableList<Double> = mutableListOf()
var totaltime = 0

fun main(args: Array<String>) {
    init()
    process()
    summarize()
}

fun init() {
    print("Number of process:")
    val nProcess = readLine()!!.toInt()

    var arriveAt = 0
    print("Enter burst time for each process\n")
    print("%3s %9s %16s".format("At", "Process", "Burst time(ms)\n"))
    for (i in arriveAt..(nProcess - 1)) {
        print("%3s %8s %13s".format("$i", "P${i+1}", ""))
        var currBurstTime = readLine()!!.toInt()
        btList.add(Process(i, i, currBurstTime))
    }
    print("\n")
}

fun process() {
    print("Scheduling scheme: First Come First Serve\n")
    for (i in 0..(btList.size - 1)) {
        startTime.add(totaltime.toDouble())
        print("|")
        for (j in 0..btList[i].burstTime - 1) {
            totaltime += 1
            print(if (j == btList[i].burstTime / 2) "P${i+1}" else "-")
        }
        print("|")
    }
    print("\n\n")
}
```

As soon as the process arrives..

...

We directly follows the order its come in

The waiting time could be calculate as follows

```
Number of process:4
Enter burst time for each process
 At   Process  Burst time(ms)
  0       P1              5
  1       P2              8
  2       P3              2
  3       P4              4

Scheduling scheme: First Come First Serve
|--P1--||----P2---||-P3||--P4-|

In summary:
P1 waited  0.00 ms
P2 waited  4.00 ms
P3 waited 11.00 ms
P4 waited 12.00 ms
Since there are 4 processes,
The average waiting time is 6.75 ms
```

The waiting could be calculate as follows

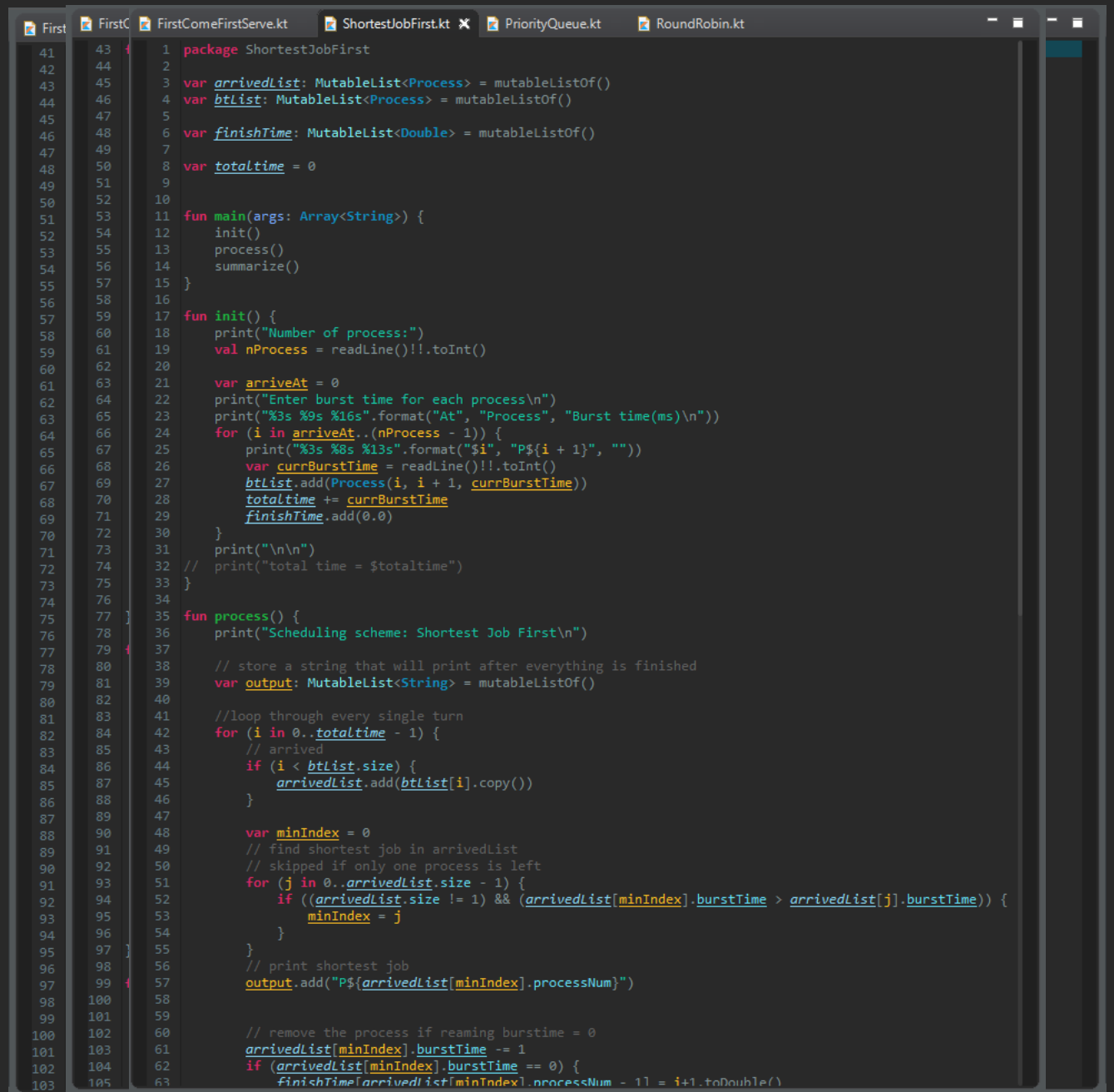
Finished time

–

(Arrival time

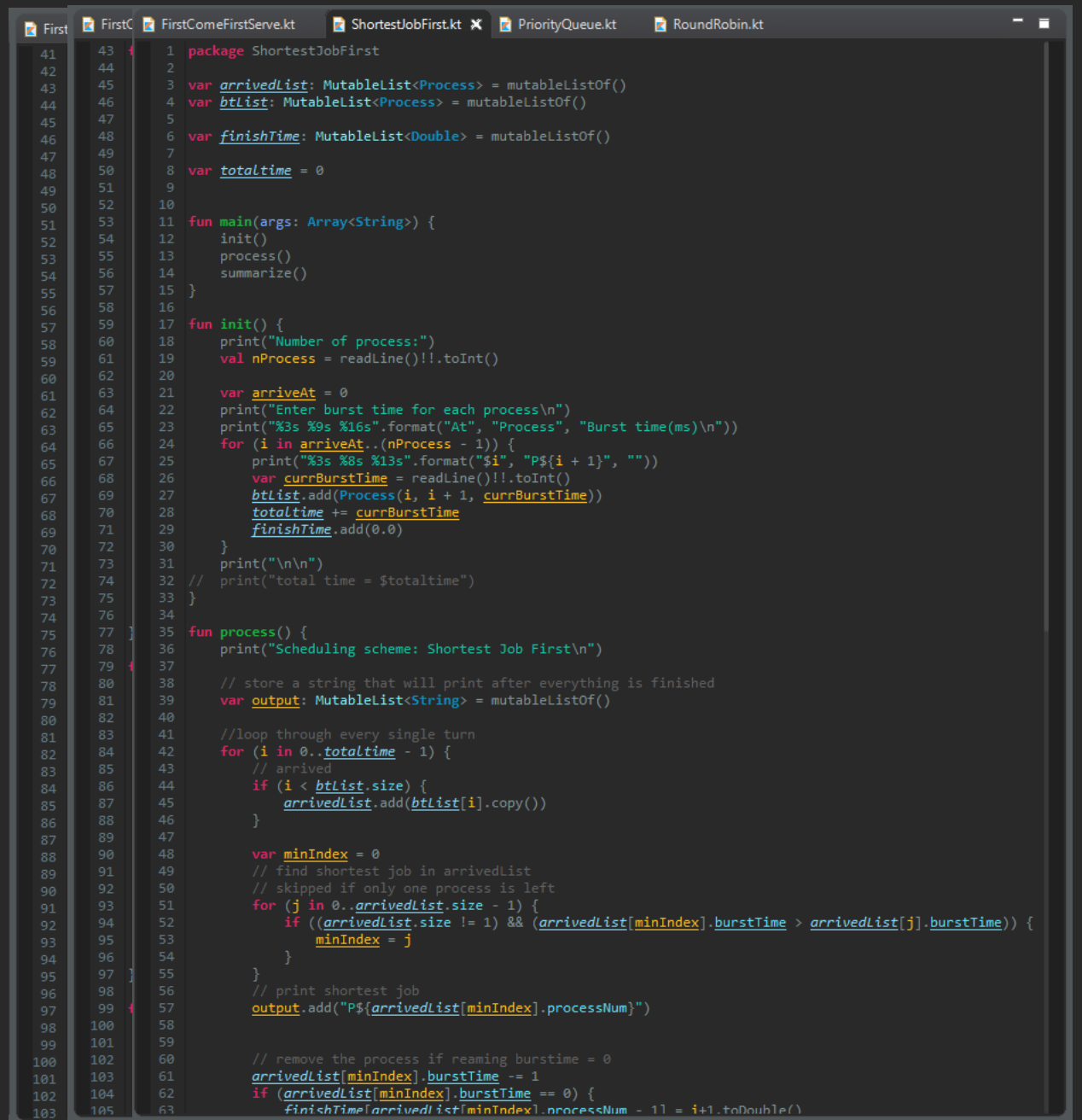
+ Burst time)

```
Number of process:4
Enter burst time for each process
 At   Process  Burst time(ms)
  0        P1              5
  1        P2              8
  2        P3              2
  3        P4              4

Scheduling scheme: First Come First Serve
|--P1--||----P2---||-P3||--P4-|

In summary:
P1 waited  0.00 ms
P2 waited  4.00 ms
P3 waited 11.00 ms
P4 waited 12.00 ms
Since there are 4 processes,
The average waiting time is 6.75 ms
```




```kotlin
package ShortestJobFirst

var arrivedList: MutableList<Process> = mutableListOf()
var btList: MutableList<Process> = mutableListOf()

var finishTime: MutableList<Double> = mutableListOf()

var totaltime = 0


fun main(args: Array<String>) {
    init()
    process()
    summarize()
}

fun init() {
    print("Number of process:")
    val nProcess = readLine()!!.toInt()

    var arriveAt = 0
    print("Enter burst time for each process\n")
    print("%3s %9s %16s".format("At", "Process", "Burst time(ms)\n"))
    for (i in arriveAt..(nProcess - 1)) {
        print("%3s %8s %13s".format("$i", "P${i + 1}", ""))
        var currBurstTime = readLine()!!.toInt()
        btList.add(Process(i, i + 1, currBurstTime))
        totaltime += currBurstTime
        finishTime.add(0.0)
    }
    print("\n\n")
//  print("total time = $totaltime")
}

fun process() {
    print("Scheduling scheme: Shortest Job First\n")

    // store a string that will print after everything is finished
    var output: MutableList<String> = mutableListOf()

    //loop through every single turn
    for (i in 0..totaltime - 1) {
        // arrived
        if (i < btList.size) {
            arrivedList.add(btList[i].copy())
        }

        var minIndex = 0
        // find shortest job in arrivedList
        // skipped if only one process is left
        for (j in 0..arrivedList.size - 1) {
            if ((arrivedList.size != 1) && (arrivedList[minIndex].burstTime > arrivedList[j].burstTime)) {
                minIndex = j
            }
        }
        // print shortest job
        output.add("P${arrivedList[minIndex].processNum}")


        // remove the process if reaming burstime = 0
        arrivedList[minIndex].burstTime -= 1
        if (arrivedList[minIndex].burstTime == 0) {
            finishTime[arrivedList[minIndex].processNum - 1] = i+1.toDouble()
```

2.Shortest Job First




```kotlin
package ShortestJobFirst

var arrivedList: MutableList<Process> = mutableListOf()
var btList: MutableList<Process> = mutableListOf()

var finishTime: MutableList<Double> = mutableListOf()

var totaltime = 0


fun main(args: Array<String>) {
    init()
    process()
    summarize()
}

fun init() {
    print("Number of process:")
    val nProcess = readLine()!!.toInt()

    var arriveAt = 0
    print("Enter burst time for each process\n")
    print("%3s %9s %16s".format("At", "Process", "Burst time(ms)\n"))
    for (i in arriveAt..(nProcess - 1)) {
        print("%3s %8s %13s".format("$i", "P${i + 1}", ""))
        var currBurstTime = readLine()!!.toInt()
        btList.add(Process(i, i + 1, currBurstTime))
        totaltime += currBurstTime
        finishTime.add(0.0)
    }
    print("\n\n")
//  print("total time = $totaltime")
}

fun process() {
    print("Scheduling scheme: Shortest Job First\n")

    // store a string that will print after everything is finished
    var output: MutableList<String> = mutableListOf()

    //loop through every single turn
    for (i in 0..totaltime - 1) {
        // arrived
        if (i < btList.size) {
            arrivedList.add(btList[i].copy())
        }

        var minIndex = 0
        // find shortest job in arrivedList
        // skipped if only one process is left
        for (j in 0..arrivedList.size - 1) {
            if ((arrivedList.size != 1) && (arrivedList[minIndex].burstTime > arrivedList[j].burstTime)) {
                minIndex = j
            }
        }
        // print shortest job
        output.add("P${arrivedList[minIndex].processNum}")


        // remove the process if reaming burstime = 0
        arrivedList[minIndex].burstTime -= 1
        if (arrivedList[minIndex].burstTime == 0) {
            finishTime[arrivedList[minIndex].processNum - 1] = i+1.toDouble()
```

Focuses on
shortest process first,
before actually touching
time-consuming ones

```
13      process()
14      summarize()
15  }
16
17  fun init() {
18      print("Number of process:")
19      val nProcess = readLine()!!.toInt()
20
21      var arriveAt = 0
22      print("Enter burst time for each process\n")
23      print("%3s %9s %16s".format("At", "Process", "Burst time(ms)\n"))
24      for (i in arriveAt..(nProcess - 1)) {
25          print("%3s %8s %13s".format("$i", "P${i + 1}", ""))
26          var currBurstTime = readLine()!!.toInt()
27          btList.add(Process(i, i + 1, currBurstTime))
28          totaltime += currBurstTime
29          finishTime.add(0.0)
30      }
31      print("\n\n")
32  //  print("total time = $totaltime")
33  }
34
35  fun process() {
36      print("Scheduling scheme: Shortest Job First\n")
37
38      // store a string that will print after everything is finished
39      var output: MutableList<String> = mutableListOf()
40
41      //loop through every single turn
42      for (i in 0..totaltime - 1) {
43          // arrived
44          if (i < btList.size) {
45              arrivedList.add(btList[i].copy())
46          }
47
48          var minIndex = 0
49          // find shortest job in arrivedList
50          // skipped if only one process is left
51          for (j in 0..arrivedList.size - 1) {
52              if ((arrivedList.size != 1) && (arrivedList[minIndex].burstTime > arrivedList[j].burstTime)) {
53                  minIndex = j
54              }
55          }
56          // print shortest job
57          output.add("P${arrivedList[minIndex].processNum}")
58
59
```

Loops through the process pool

to pick the shortest one

Since P1 burst time is 6,

as P2 arrives

with less burst time.

The algorithm instantly shift to work on P2

```
Number of process:5
Enter burst time for each process
 At   Process  Burst time(ms)
  0        P1              6
  1        P2              4
  2        P3              1
  3        P4              3
  4        P5              5


Scheduling scheme: Shortest Job First
|P1||P2||P3||-P2-||-P4-||--P1--||--P5--|

In summary:
P1 waited  8.00 ms
P2 waited  1.00 ms
P3 waited  0.00 ms
P4 waited  3.00 ms
P5 waited 10.00 ms
Since there are 5 processes,
The average waiting time is 4.4 ms
```

Same goes for when P3 arrives.

After the P3 is done,

Automatically looks for the next shortest

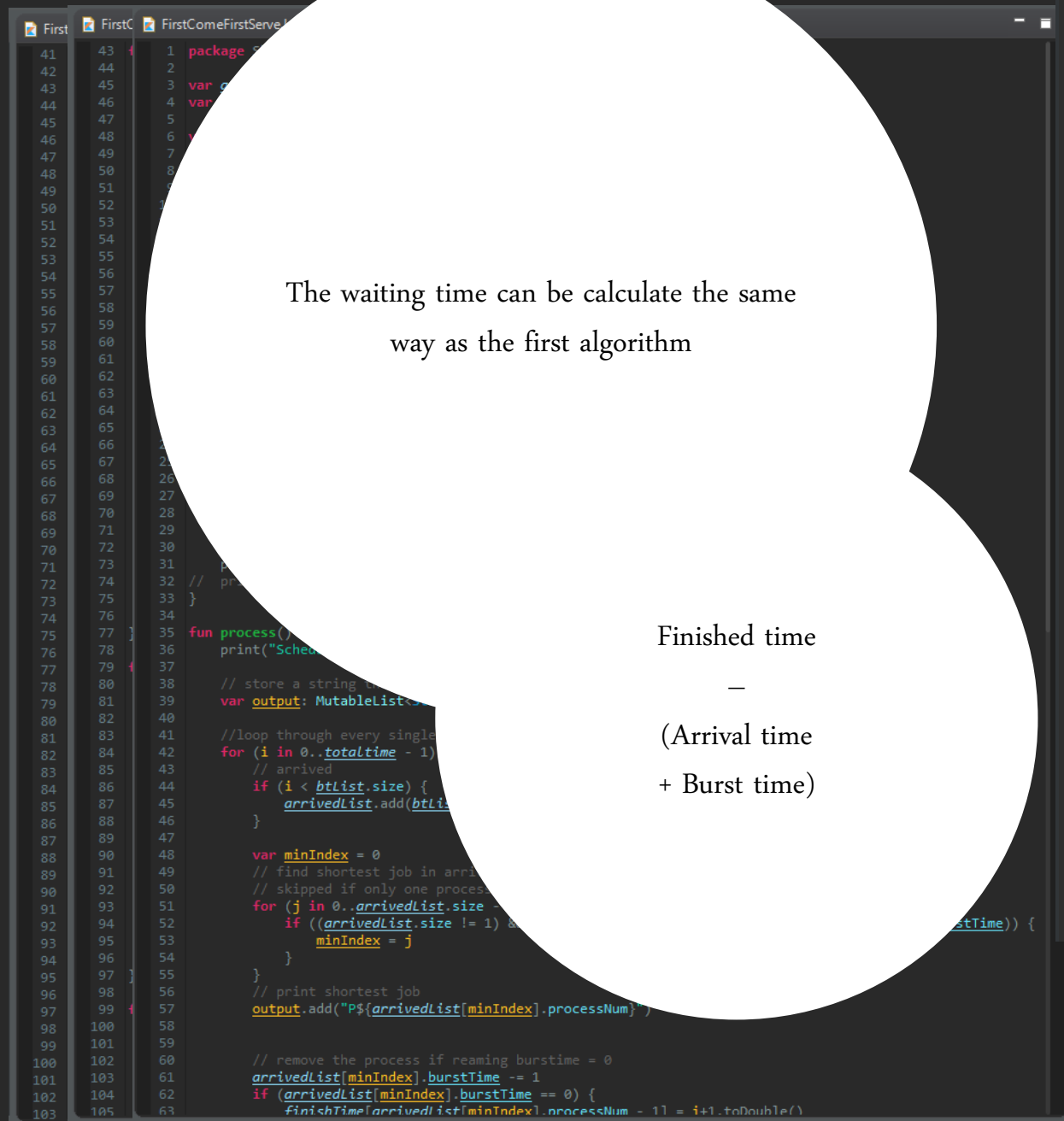
job

that is left in the pool

```
Number of process:5
Enter burst time for each process
 At   Process  Burst time(ms)
  0        P1              6
  1        P2              4
  2        P3              1
  3        P4              3
  4        P5              5


Scheduling scheme: Shortest Job First
|P1||P2||P3||-P2-||-P4-||--P1--||--P5--|

In summary:
P1 waited  8.00 ms
P2 waited  1.00 ms
P3 waited  0.00 ms
P4 waited  3.00 ms
P5 waited 10.00 ms
Since there are 5 processes,
The average waiting time is 4.4 ms
```

The waiting time can be calculate the same way as the first algorithm

```
Number of process:5
Enter burst time for each process
 At   Process  Burst time(ms)
  0        P1              6
  1        P2              4
  2        P3              1
  3        P4              3
  4        P5              5


Scheduling scheme: Shortest Job First
|P1||P2||P3||-P2-||-P4-||--P1--||--P5--|

In summary:
P1 waited  8.00 ms
P2 waited  1.00 ms
P3 waited  0.00 ms
P4 waited  3.00 ms
P5 waited 10.00 ms
Since there are 5 processes,
The average waiting time is 4.4 ms
```



The waiting time can be calculate the same way as the first algorithm

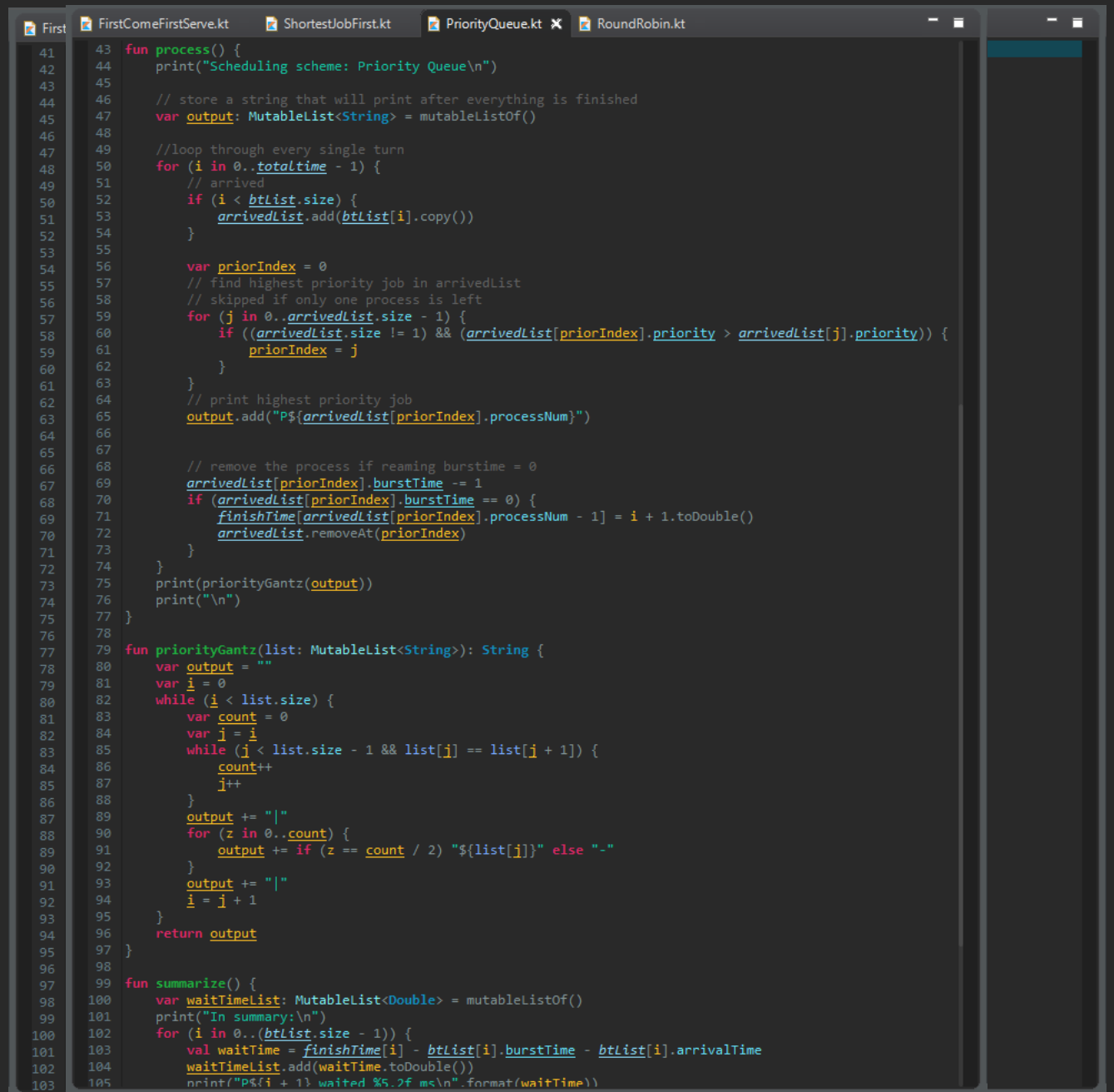
Finished time

–

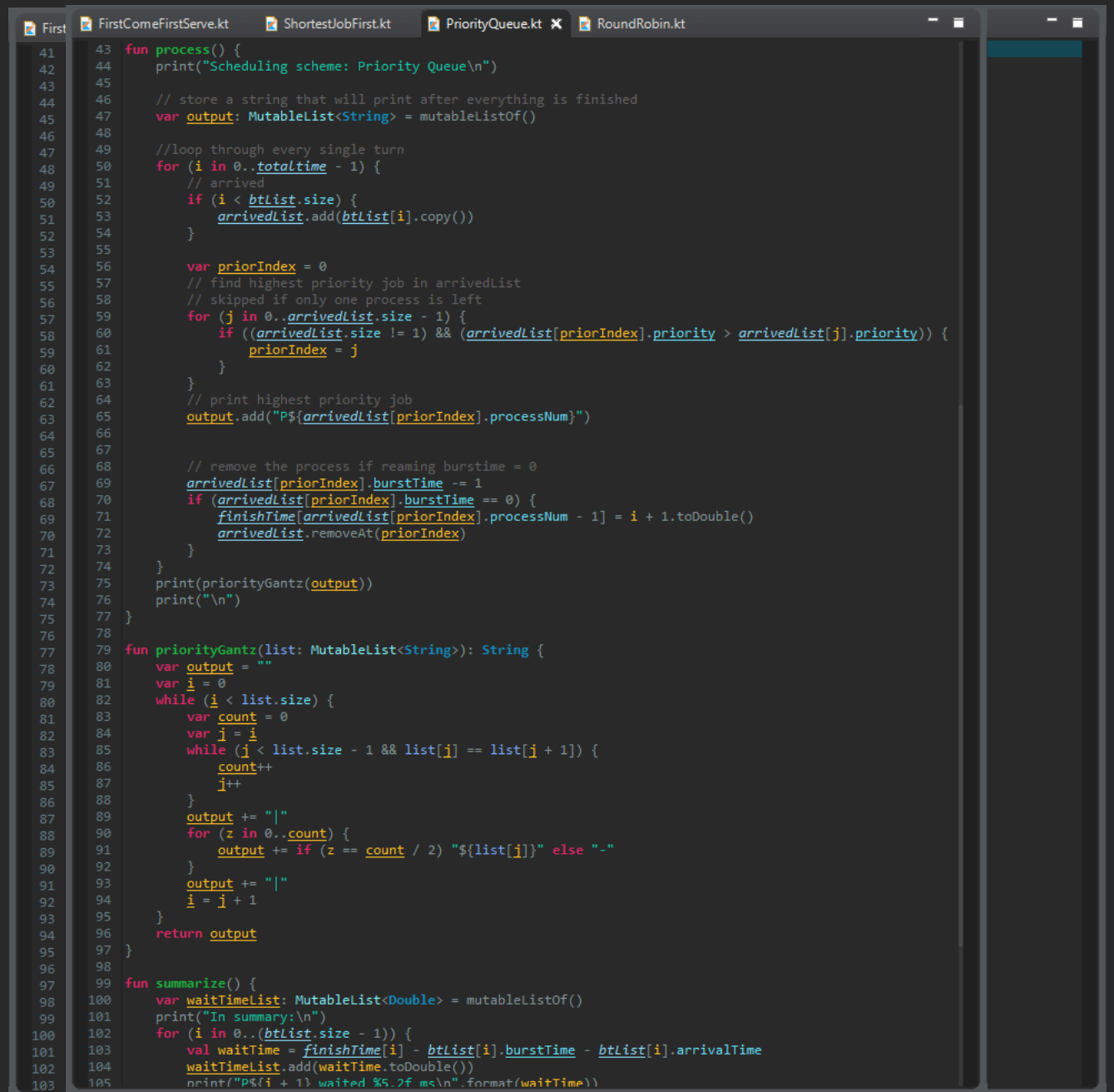
(Arrival time

+ Burst time)

```
Number of process:5
Enter burst time for each process
 At   Process  Burst time(ms)
  0        P1              6
  1        P2              4
  2        P3              1
  3        P4              3
  4        P5              5


Scheduling scheme: Shortest Job First
|P1||P2||P3||-P2-||-P4-||--P1--||--P5--|

In summary:
P1 waited  8.00 ms
P2 waited  1.00 ms
P3 waited  0.00 ms
P4 waited  3.00 ms
P5 waited 10.00 ms
Since there are 5 processes,
The average waiting time is 4.4 ms
```




3.Priority Queue




Add priority variable;

Determines which jobs to be executed fist

```
43  fun process() {
44      print("Scheduling scheme: Priority Queue\n")
45
46      // store a string that will print after everything is finished
47      var output: MutableList<String> = mutableListOf()
48
49      //loop through every single turn
50      for (i in 0..totaltime - 1) {
51          // arrived
52          if (i < btList.size) {
53              arrivedList.add(btList[i].copy())
54          }
55
56          var priorIndex = 0
57          // find highest priority job in arrivedList
58          // skipped if only one process is left
59          for (j in 0..arrivedList.size - 1) {
60              if ((arrivedList.size != 1) && (arrivedList[priorIndex].priority > arrivedList[j].pri
61                  priorIndex = j
62              }
63          }
64          // print highest priority job
65          output.add("P${arrivedList[priorIndex].processNum}")
66
67
68          // remove the process if reaming burstime = 0
69          arrivedList[priorIndex].burstTime -= 1
70          if (arrivedList[priorIndex].burstTime == 0) {
71              finishTime[arrivedList[priorIndex].processNum - 1] = i + 1.toDouble()
72              arrivedList.removeAt(priorIndex)
73          }
74      }
75      print(priorityGantz(output))
76      print("\n")
77  }
78
79  fun priorityGantz(list: MutableList<String>): String {
80      var output = ""
81      var i = 0
82      while (i < list.size) {
83          var count = 0
84          var j = i
85          while (j < list.size - 1 && list[j] == list[j + 1]) {
86              count++
87              j++
88          }
89          output += "|"
90          for (z in 0..count) {
91              output += if (z == count / 2) "${list[j]}" else "-"
```

Loops through the process pool

to pick the shortest one

```
43  fun process() {
44      print("Scheduling scheme: Priority Queue\n")
45
46      // store a string that will print after everything is finished
47      var output: MutableList<String> = mutableListOf()
48
49      //loop through every single turn
50      for (i in 0..totaltime - 1) {
51          // arrived
52          if (i < btList.size) {
53              arrivedList.add(btList[i].copy())
54          }
55
56          var priorIndex = 0
57          // find highest priority job in arrivedList
58          // skipped if only one process is left
59          for (j in 0..arrivedList.size - 1) {
60              if ((arrivedList.size != 1) && (arrivedList[priorIndex].priority > arrivedList[j].pri
61                  priorIndex = j
62              }
63          }
64          // print highest priority job
65          output.add("P${arrivedList[priorIndex].processNum}")
66
67
68          // remove the process if reaming burstime = 0
69          arrivedList[priorIndex].burstTime -= 1
70          if (arrivedList[priorIndex].burstTime == 0) {
71              finishTime[arrivedList[priorIndex].processNum - 1] = i + 1.toDouble()
72              arrivedList.removeAt(priorIndex)
73          }
74      }
75      print(priorityGantz(output))
76      print("\n")
77  }
78
79  fun priorityGantz(list: MutableList<String>): String {
80      var output = ""
81      var i = 0
82      while (i < list.size) {
83          var count = 0
84          var j = i
85          while (j < list.size - 1 && list[j] == list[j + 1]) {
86              count++
87              j++
88          }
89          output += "|"
90          for (z in 0..count) {
91              output += if (z == count / 2) "${list[j]}" else "-"
```

If a higher priority task arrives,
pause the current one to prioritize it first

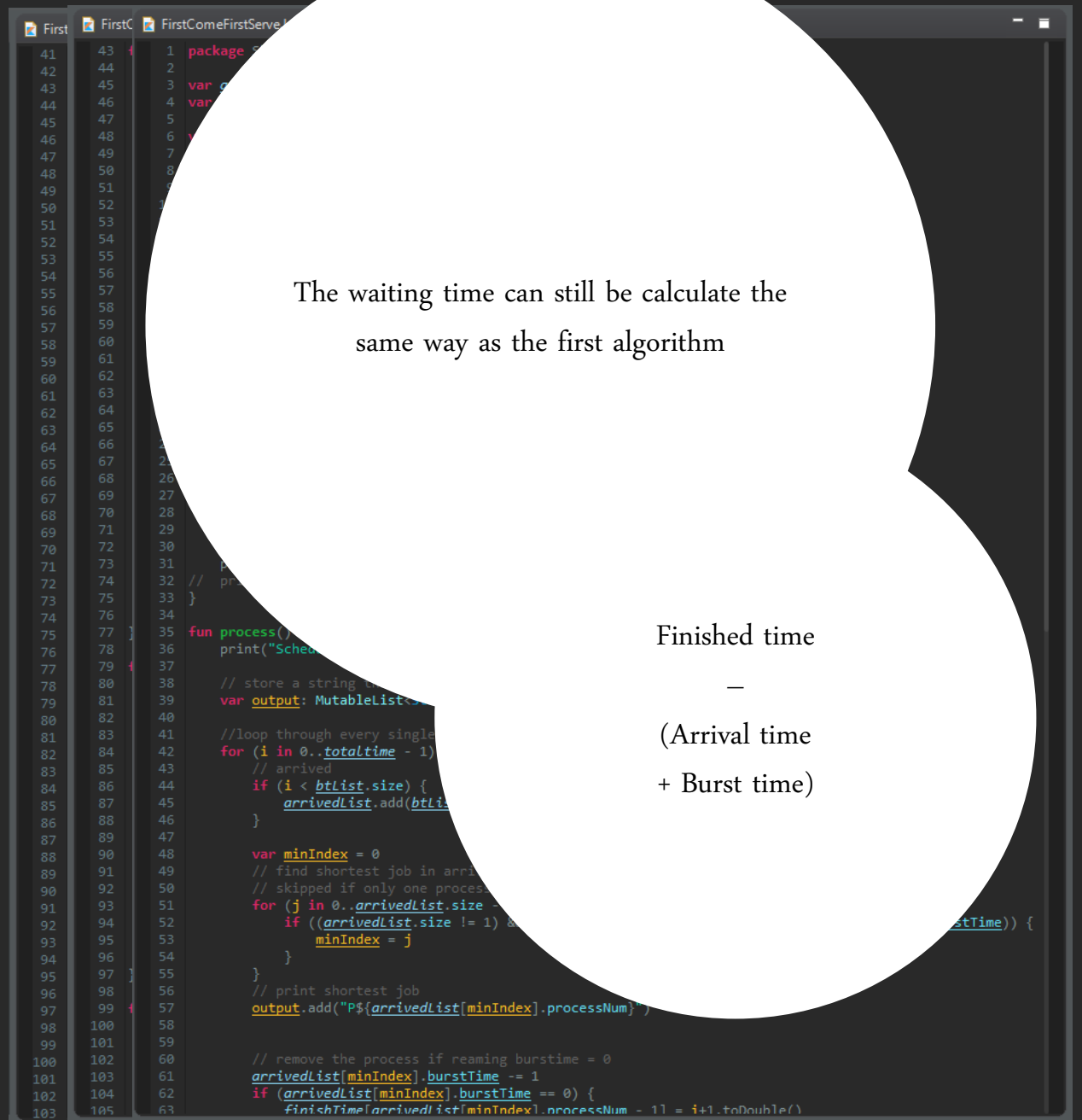

```
Number of process:4
Enter burst time for each process
 At   Process  Burst time(ms)
  0        P1              4
  1        P2              5
  2        P3              4
  3        P4              6
Enter priority for each process
 At   Process        Priority
  0        P1              4
  1        P2              2
  2        P3              3
  3        P4              1


Scheduling scheme: Priority Queue
|P1||P2-||--P4---||-P2-||-P3--||-P1-|
In summary:
P1 waited 15.00 ms
P2 waited  6.00 ms
P3 waited 10.00 ms
P4 waited  0.00 ms
Since there are 4 processes,
The average waiting time is 7.75 ms
```

The same goes for when P4 which has highest priority arrives

```
Number of process:4
Enter burst time for each process
 At   Process  Burst time(ms)
  0        P1              4
  1        P2              5
  2        P3              4
  3        P4              6
Enter priority for each process
 At   Process        Priority
  0        P1              4
  1        P2              2
  2        P3              3
  3        P4              1


Scheduling scheme: Priority Queue
|P1||P2-||--P4---||-P2-||-P3--||-P1-|
In summary:
P1 waited 15.00 ms
P2 waited  6.00 ms
P3 waited 10.00 ms
P4 waited  0.00 ms
Since there are 4 processes,
The average waiting time is 7.75 ms
```

The waiting time can still be calculate the same way as the first algorithm

```
Number of process:4
Enter burst time for each process
 At   Process  Burst time(ms)
  0        P1              4
  1        P2              5
  2        P3              4
  3        P4              6
Enter priority for each process
 At   Process        Priority
  0        P1              4
  1        P2              2
  2        P3              3
  3        P4              1


Scheduling scheme: Priority Queue
|P1||P2-||--P4---||-P2-||-P3--||-P1-|
In summary:
P1 waited 15.00 ms
P2 waited  6.00 ms
P3 waited 10.00 ms
P4 waited  0.00 ms
Since there are 4 processes,
The average waiting time is 7.75 ms
```



The waiting time can still be calculate the same way as the first algorithm

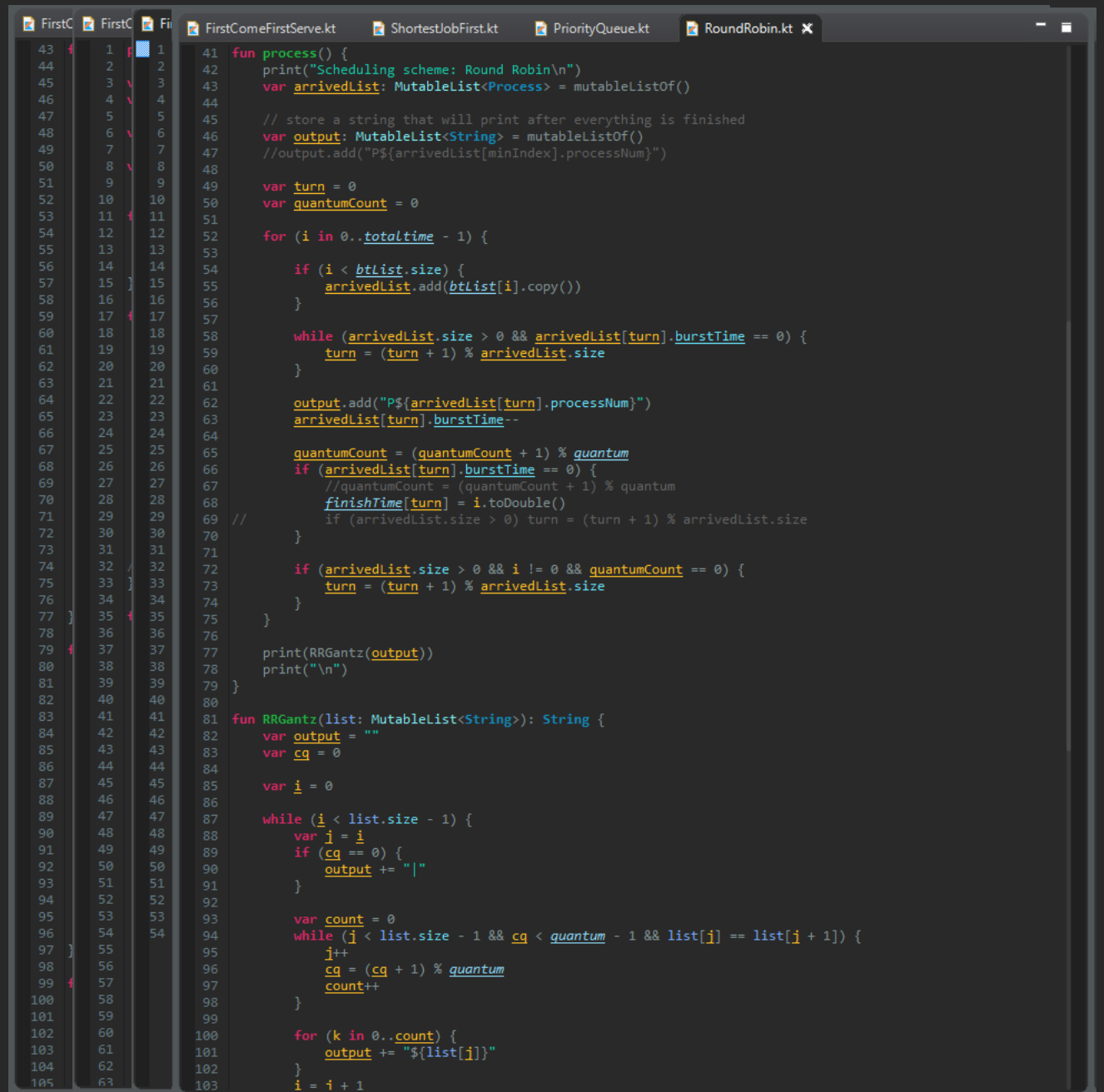
Finished time

–

(Arrival time

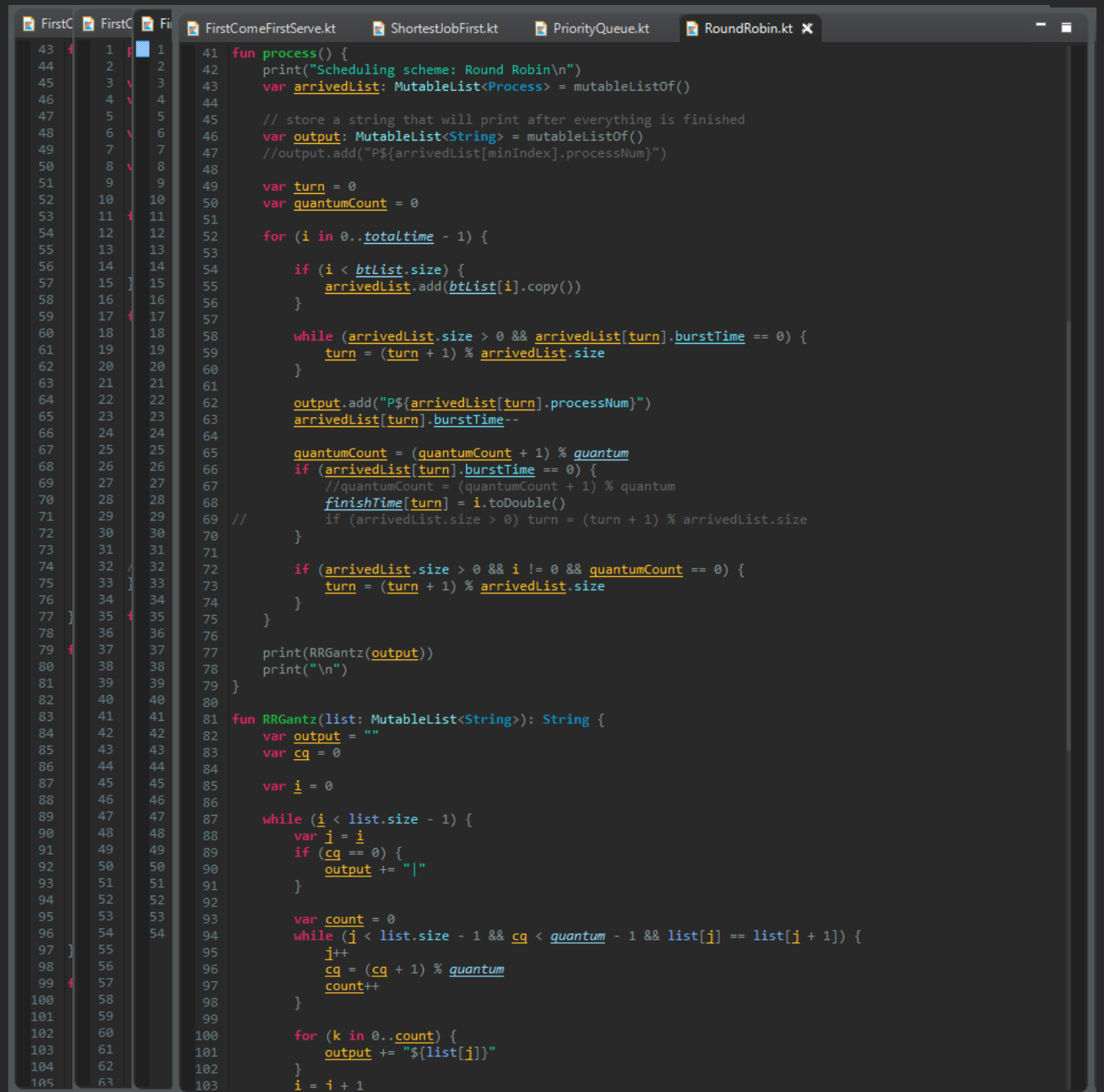
+ Burst time)

```
Number of process:4
Enter burst time for each process
 At   Process  Burst time(ms)
  0        P1              4
  1        P2              5
  2        P3              4
  3        P4              6
Enter priority for each process
 At   Process        Priority
  0        P1              4
  1        P2              2
  2        P3              3
  3        P4              1


Scheduling scheme: Priority Queue
|P1||P2-||--P4---||-P2-||-P3--||-P1-|
In summary:
P1 waited 15.00 ms
P2 waited  6.00 ms
P3 waited 10.00 ms
P4 waited  0.00 ms
Since there are 4 processes,
The average waiting time is 7.75 ms
```




```kotlin
fun process() {
    print("Scheduling scheme: Round Robin\n")
    var arrivedList: MutableList<Process> = mutableListOf()

    // store a string that will print after everything is finished
    var output: MutableList<String> = mutableListOf()
    //output.add("P${arrivedList[minIndex].processNum}")

    var turn = 0
    var quantumCount = 0

    for (i in 0..totaltime - 1) {

        if (i < btList.size) {
            arrivedList.add(btList[i].copy())
        }

        while (arrivedList.size > 0 && arrivedList[turn].burstTime == 0) {
            turn = (turn + 1) % arrivedList.size
        }

        output.add("P${arrivedList[turn].processNum}")
        arrivedList[turn].burstTime--

        quantumCount = (quantumCount + 1) % quantum
        if (arrivedList[turn].burstTime == 0) {
            //quantumCount = (quantumCount + 1) % quantum
            finishTime[turn] = i.toDouble()
//          if (arrivedList.size > 0) turn = (turn + 1) % arrivedList.size
        }

        if (arrivedList.size > 0 && i != 0 && quantumCount == 0) {
            turn = (turn + 1) % arrivedList.size
        }
    }

    print(RRGantz(output))
    print("\n")
}

fun RRGantz(list: MutableList<String>): String {
    var output = ""
    var cq = 0

    var i = 0

    while (i < list.size - 1) {
        var j = i
        if (cq == 0) {
            output += "|"
        }

        var count = 0
        while (j < list.size - 1 && cq < quantum - 1 && list[j] == list[j + 1]) {
            j++
            cq = (cq + 1) % quantum
            count++
        }

        for (k in 0..count) {
            output += "${list[j]}"
        }
        i = i + 1
```

4.RoundRobin




Introduce quantum;

Fairly divides tasks amongst processes in the pool

```
40
41  fun process() {
42      print("Scheduling scheme: Round Robin\n")
43      var arrivedList: MutableList<Process> = mutableListOf()
44
45      // store a string that will print after everything is finished
46      var output: MutableList<String> = mutableListOf()
47      //output.add("P${arrivedList[minIndex].processNum}")
48
49      var turn = 0
50      var quantumCount = 0
51
52      for (i in 0..totaltime - 1) {
53
54          if (i < btList.size) {
55              arrivedList.add(btList[i].copy())
56          }
57
58          while (arrivedList.size > 0 && arrivedList[turn].burstTime == 0) {
59              turn = (turn + 1) % arrivedList.size
60          }
61
62          output.add("P${arrivedList[turn].processNum}")
63          arrivedList[turn].burstTime--
64
65          quantumCount = (quantumCount + 1) % quantum
66          if (arrivedList[turn].burstTime == 0) {
67              //quantumCount = (quantumCount + 1) % quantum
68              finishTime[turn] = i.toDouble()
69  //          if (arrivedList.size > 0) turn = (turn + 1) % arrivedList.size
70          }
71
72          if (arrivedList.size > 0 && i != 0 && quantumCount == 0) {
73              turn = (turn + 1) % arrivedList.size
74          }
75      }
76
77      print(RRGantz(output))
78      print("\n")
79  }
80
81  fun RRGantz(list: MutableList<String>): String {
82      var output = ""
83      var cq = 0
84
85      var i = 0
86
87      while (i < list.size - 1) {
88          var j = i
```

Each tasks will be execute concurrently and equally

```
40
41  fun process() {
42      print("Scheduling scheme: Round Robin\n")
43      var arrivedList: MutableList<Process> = mutableListOf()
44
45      // store a string that will print after everything is finished
46      var output: MutableList<String> = mutableListOf()
47      //output.add("P${arrivedList[minIndex].processNum}")
48
49      var turn = 0
50      var quantumCount = 0
51
52      for (i in 0..totaltime - 1) {
53
54          if (i < btList.size) {
55              arrivedList.add(btList[i].copy())
56          }
57
58          while (arrivedList.size > 0 && arrivedList[turn].burstTime == 0) {
59              turn = (turn + 1) % arrivedList.size
60          }
61
62          output.add("P${arrivedList[turn].processNum}")
63          arrivedList[turn].burstTime--
64
65          quantumCount = (quantumCount + 1) % quantum
66          if (arrivedList[turn].burstTime == 0) {
67              //quantumCount = (quantumCount + 1) % quantum
68              finishTime[turn] = i.toDouble()
69  //          if (arrivedList.size > 0) turn = (turn + 1) % arrivedList.size
70          }
71
72          if (arrivedList.size > 0 && i != 0 && quantumCount == 0) {
73              turn = (turn + 1) % arrivedList.size
74          }
75      }
76
77      print(RRGantz(output))
78      print("\n")
79  }
80
81  fun RRGantz(list: MutableList<String>): String {
82      var output = ""
83      var cq = 0
84
85      var i = 0
86
87      while (i < list.size - 1) {
88          var j = i
```

Ensures all tasks progress

Executes each task for a turn of 3

(quantum)

```
Number of process:5
Quantum is:3
Enter burst time for each process
 At   Process  Burst time(ms)
  0       P1              4
  1       P2              5
  2       P3              4
  3       P4              3
  4       P5              5


Scheduling scheme: Round Robin
|P1P1P1|P2P2P2|P3P3P3|P4P4P4|P5P5P5|P1P2P2|P3P5P5

In summary:
P1 waited 12.00 ms
P2 waited 13.00 ms
P3 waited 15.00 ms
P4 waited  9.00 ms
P5 waited 16.00 ms
Since there are 5 processes,
The average waiting time is 13.0 ms
```

Notice when one task is done

and the quantum is yet complete, it utilize

the remaining turn on the next task

In P1 to P2 to P2

($6^{th}$ quantum)

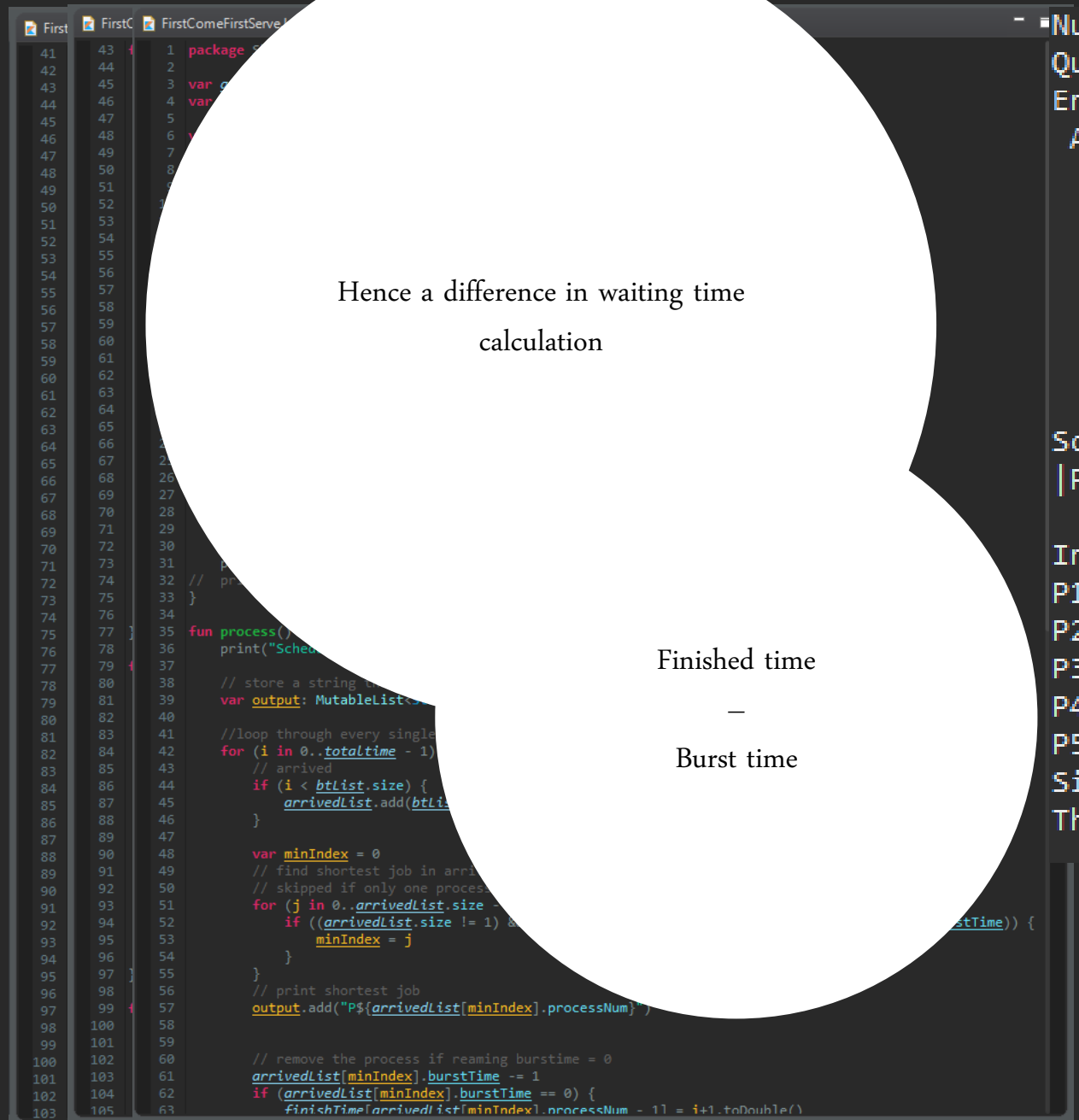
```
Number of process:5
Quantum is:3
Enter burst time for each process
 At   Process  Burst time(ms)
  0       P1              4
  1       P2              5
  2       P3              4
  3       P4              3
  4       P5              5


Scheduling scheme: Round Robin
|P1P1P1|P2P2P2|P3P3P3|P4P4P4|P5P5P5|P1P2P2|P3P5P5

In summary:
P1 waited 12.00 ms
P2 waited 13.00 ms
P3 waited 15.00 ms
P4 waited  9.00 ms
P5 waited 16.00 ms
Since there are 5 processes,
The average waiting time is 13.0 ms
```

Since in round robin, we neglect arrival time.

```
Number of process:5
Quantum is:3
Enter burst time for each process
 At   Process  Burst time(ms)
  0       P1              4
  1       P2              5
  2       P3              4
  3       P4              3
  4       P5              5


Scheduling scheme: Round Robin
|P1P1P1|P2P2P2|P3P3P3|P4P4P4|P5P5P5|P1P2P2|P3P5P5

In summary:
P1 waited 12.00 ms
P2 waited 13.00 ms
P3 waited 15.00 ms
P4 waited  9.00 ms
P5 waited 16.00 ms
Since there are 5 processes,
The average waiting time is 13.0 ms
```

Hence a difference in waiting time calculation

```
Number of process:5
Quantum is:3
Enter burst time for each process
 At   Process  Burst time(ms)
  0       P1              4
  1       P2              5
  2       P3              4
  3       P4              3
  4       P5              5


Scheduling scheme: Round Robin
|P1P1P1|P2P2P2|P3P3P3|P4P4P4|P5P5P5|P1P2P2|P3P5P5

In summary:
P1 waited 12.00 ms
P2 waited 13.00 ms
P3 waited 15.00 ms
P4 waited  9.00 ms
P5 waited 16.00 ms
Since there are 5 processes,
The average waiting time is 13.0 ms
```



Hence a difference in waiting time calculation

Finished time

–

Burst time

```
Number of process:5
Quantum is:3
Enter burst time for each process
 At   Process  Burst time(ms)
  0       P1              4
  1       P2              5
  2       P3              4
  3       P4              3
  4       P5              5


Scheduling scheme: Round Robin
|P1P1P1|P2P2P2|P3P3P3|P4P4P4|P5P5P5|P1P2P2|P3P5P5

In summary:
P1 waited 12.00 ms
P2 waited 13.00 ms
P3 waited 15.00 ms
P4 waited  9.00 ms
P5 waited 16.00 ms
Since there are 5 processes,
The average waiting time is 13.0 ms
```

# Happy Coding!

THANK YOU



U5911694 Rachatha Pramualsuk