Shortest Job First Scheduling

(Java Program)

COMSATS University Islamabad
Sahiwal Campus

Usama Sarwar

FA17-BS(CS)-090-B

Mr. Umer

Operating System Concepts

October 29, 2019

Table of Contents

List of Tables

List of Figures

Shortest Job First

I. Non-Primitive Algorithm

Suppose we have set of processes are in ready queue. The SJF scheduling algorithm will choose the job which has shortest remaining time to complete. We have 2 variations of this SJF algorithm that are preemptive and non-preemptive. **Preemptive version of SJF also known as SRTF**. (Li, Kavi et al. 2007)

A. Example:

Process id	Arrival time	Burst time
P1	0	3
P2	0	1
P3	0	2

Table 1. Values for Non-Preemptive SJF Algorithm

We have 3 processes in our ready queue. As we discussed SJF will schedule the job which is having least execution time or burst time.

1. Gantt Chart

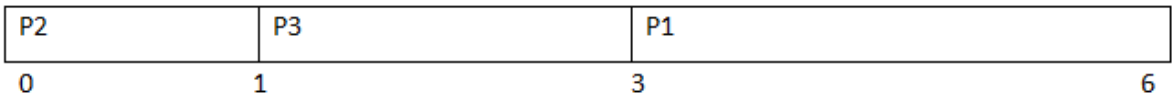

Table 2. Gantt Chart (Non-Preemptive)

Now we will calculate the completion time, waiting time, turnaround time of each process.

Process id	Completion time	Waiting time	Turnaround time
P1	6	3	6
P2	1	0	1
P3	3	1	3

Table 3. Results of Non-Preemptive SJF Algorithm

2. Calculations

Average waiting time = (3+0+1)/3 = 1.33
Turnaround time = (6+1+3)/3 = 3.33

B. Java Method for Non-Preemptive

```
public void NonPreemptive() {
        System.out.print("Number of Processes: ");
        int n = sc.nextInt();
        int pid[] = new int[n];
        int at[] = new int[n]; // at means arrival time
        int bt[] = new int[n]; // bt means burst time
        int ct[] = new int[n]; // ct means complete time
        int ta[] = new int[n]; // ta means turn around time
        int wt[] = new int[n];  //wt means waiting time
        int f[] = new int[n];  // f means it is flag it checks process is
completed or not
        int st = 0, tot = 0;
        float avgwt = 0, avgta = 0;
        for (int i = 0; i < n; i++) {
            System.out.print("Process " + (i + 1) + " Arrival Time:");
            at[i] = sc.nextInt();
            System.out.print("Process " + (i + 1) + " Brust Time:");
            bt[i] = sc.nextInt();
            pid[i] = i + 1;
            f[i] = 0;
        }
        boolean a = true;
        while (true) {
            int c = n, min = 999;
            if (tot == n) // total no of process = completed process loop will be
terminated
            {
                break;
            }
            for (int i = 0; i < n; i++) {
                            if ((at[i] <= st) && (f[i] == 0) && (bt[i] < min)) {
                min = bt[i];
                c = i;
            }
            }
```

```
            if (c == n) {
                st++;
            } else {
                ct[c] = st + bt[c];
                st += bt[c];
                ta[c] = ct[c] - at[c];
                wt[c] = ta[c] - bt[c];
                f[c] = 1;
                tot++;
            }       }
        System.out.println("\nPid\tArrival\tBrust\tComplete\tTurn\tWaiting");
        for (int i = 0; i < n; i++) {
            avgwt += wt[i];
            avgta += ta[i];
            System.out.println(pid[i] + "\t" + at[i] + "\t" + bt[i] + "\t" + ct[i]
+ "\t\t" + ta[i] + "\t" + wt[i]);
        }
        System.out.println("\nAverage TAT is " + (float) (avgta / n));
        System.out.println("Average Waiting Time is " + (float) (avgwt / n));
        sc.close();
    }
```

1. **Output:**

```
Number of Processes: 2
Process 1 Arrival Time:4
Process 1 Brust Time:5
Process 2 Arrival Time:6
Process 2 Brust Time:7

Pid     Arrival Brust   Complete        Turn    Waiting
1       4       5       9               5       0
2       6       7       16              10      3

Average TAT is 7.5
Average Waiting Time is 1.5
```

Figure 1. Console Output for Non-Preemptive Algorithm

II. Primitive Algorithm

In SRTF the selection of job is same like in SJF. But the difference is In SJF process will run till completion. In SRTF process will run till completion or a new process added into queue which is having smaller execution time than the current process remaining execution time. (Bandarupalli, Nutulapati et al. 2012)

A. Example

Process id	Arrival time	Burst time
P1	2	3
P2	1	2
P3	3	4
P4	5	6

Table 4. Values for Preemptive SJF Algorithm

When process is added to queue or process is completed then only CPU may switch the process.

1. Gantt Chart

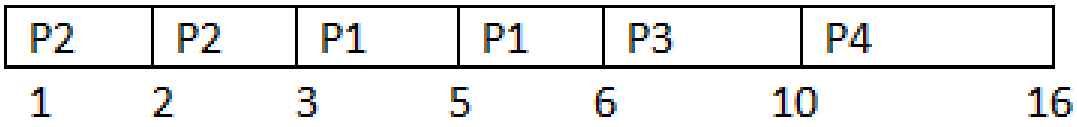

Table 5. Jantt Chart (Preemtive)

Process id	Completion time	Waiting time	Turnaround time
P1	6	1	4
P2	3	0	2
P3	10	3	7
P4	16	5	11

Table 6. Results of Preemptive SJF Algorithm

2. Calculations

Average turnaround time = (4+2+7+11)/4 = 6.0
Average waiting time = (1+0+3+5)/4 = 2.25

B. Java Method for Preemptive Algorithm

```
public void Preemptive() {
        System.out.print("Number of Process: ");
        int n = sc.nextInt();
        int pid[] = new int[n]; // it takes pid of process
        int at[] = new int[n]; // at means arrival time
        int bt[] = new int[n]; // bt means burst time
        int ct[] = new int[n]; // ct means complete time
        int ta[] = new int[n];// ta means turn around time
        int wt[] = new int[n];  // wt means waiting time
        int f[] = new int[n];  // f means it is flag it checks process is
completed or not
        int k[] = new int[n];   // it is also stores brust time
        int i, st = 0, tot = 0;
        float avgwt = 0, avgta = 0;
        for (i = 0; i < n; i++) {
            pid[i] = i + 1;
            System.out.print("Process " + (i + 1) + " Arrival Time:");
            at[i] = sc.nextInt();
            System.out.print("Process " + (i + 1) + " Burst Time:");
            bt[i] = sc.nextInt();
            k[i] = bt[i];
            f[i] = 0;        }
        while (true) {
            int min = 99, c = n;
            if (tot == n) {
                break;            }
            for (i = 0; i < n; i++) {
                if ((at[i] <= st) && (f[i] == 0) && (bt[i] < min)) {
                    min = bt[i];
                    c = i;               }           }
            if (c == n) {
                st++;
            } else {
                bt[c]--;
                st++;
```

```
                if (bt[c] == 0) {
                    ct[c] = st;
                    f[c] = 1;
                    tot++;
                }           }       }
        for (i = 0; i < n; i++) {
            ta[i] = ct[i] - at[i];
            wt[i] = ta[i] - k[i];
            avgwt += wt[i];
            avgta += ta[i];        }
        System.out.println("PID\tArrival\tBurst\tComplete\tTurn\tWaiting");
        for (i = 0; i < n; i++) {
            System.out.println(pid[i] + "\t" + at[i] + "\t" + k[i] + "\t" + ct[i]
+ "\t\t" + ta[i] + "\t" + wt[i]);        }
        System.out.println("\nAverage TAT is " + (float) (avgta / n));
        System.out.println("Average Waiting Time is " + (float) (avgwt / n));
        sc.close();
    }
```

1. Output

```
Number of Process: 2
Process 1 Arrival Time:8
Process 1 Burst Time:6
Process 2 Arrival Time:4
Process 2 Burst Time:9
PID     Arrival Burst   Complete        Turn    Waiting
1       8       6       19              11      5
2       4       9       13              9       0

Average TAT is 10.0
Average Waiting Time is 2.5
```

Figure 2. Console Output for Preemptive Algorithm

References

Bandarupalli, S. B., et al. (2012). "A Novel CPU Scheduling Algorithm–Preemptive & Non-Preemptive." International Journal of Modern Engineering Research (IJMER) **2**(6): 4484-4490.

Li, W., et al. (2007). "A non-preemptive scheduling algorithm for soft real-time systems." Computers & Electrical Engineering **33**(1): 12-29.