19BCE7346 VIT-AP UNIVERSITY Taran

CSE-2008 Assignment

Academic year: 2020-2021 **Semester:** WIN

Faculty Name: Mr. Sanket Mishra sir **Date:** 03 /11/2021

Student name: M.Taran **Reg. no.:** 19BCE7346

QUESTION:

First Come First Serve scheduling algorithmImplementationin JAVA:

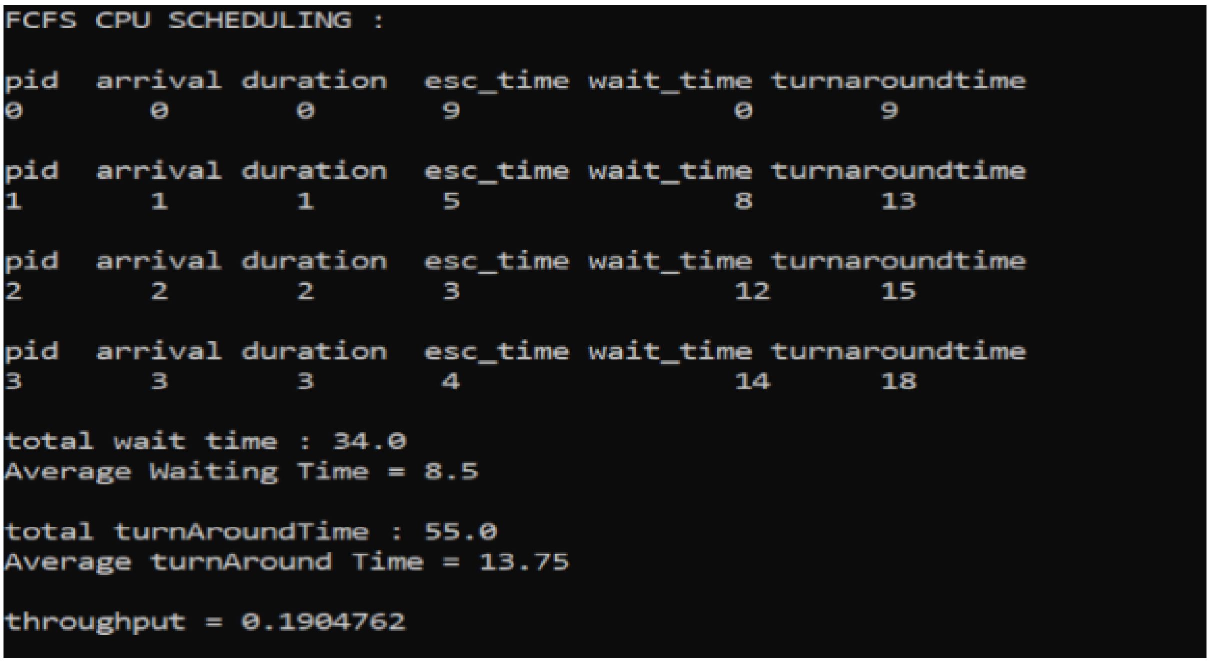
FCFS.JAVA:

Code:

```
import java.util.*;publicclassFCFS{
//process{id,arrival,duration}

staticint[][]process={{0,0,9},{1,1,5},{2,2,3},{3,3,4}};

staticArrayList<int[]>complete=newArrayList<>();


public static void main(String[] args){while(!isAllcomplete()){
complete.add(getNextProcess());

}

System.out.println("FCFS CPU SCHEDULING : ");intstartTime=complete.get(0)[1];
for(int i = 0 ; i < complete.size(); i++){int[]p=complete.get(i);
System.out.println("\npidarrival durationesc_time wait_time
turnaroundtime");System.out.println(i+"\t"+p[0]+"\t"+p[1]+"\t"+p[2]
+"\t"+"\t"+(startTime-p[1])+"\t"+((startTime+p[2])-p[1]));startTime+=p[2];
}

System.out.println("");

System.out.println("Average Waiting Time =
"+getAvgWaitingTime());System.out.println("");
System.out.println("AverageturnAroundTime="+getAvgTurnAroundTime());

System.out.println("");System.out.println("throughput="+getThroughput());
}

public static float getAvgWaitingTime(){float startTime =
complete.get(0)[1];floatsumofwait=0;
for(int i = 0; i < complete.size(); i++){int[] p = complete.get(i);sumofwait +=
(startTime - p[1]);startTime+=p[2];
}

System.out.println("total wait time :
"+sumofwait);return(sumofwait/(process.length));
}

public static float
getAvgTurnAroundTime(){floatstartTime=complete.get(0)[1];floatsumofturnAround=0;
for(int i = 0; i < complete.size(); i++){int[]p=complete.get(i);
sumofturnAround += ((startTime+p[2]) - p[1]);startTime+=p[2];
}

System.out.println("total turnAroundTime :
"+sumofturnAround);return(sumofturnAround/(process.length));
}

public static float getThroughput(){floatsumOfDuration=0;
for (int[] each : process){sumOfDuration+=each[2];
}

return(process.length/sumOfDuration);

}

public static int[] getNextProcess(){int pid=0;
for(inti=0;i<process.length;i++){if(!complete.contains(process[i])){pid=i;break;
}}

for(int i = 0; i < process.length; i++){if(!complete.contains(process[i])){
if(process[i][1] < process[pid][1]){pid=i;
}

}

}

return process[pid];

}

public static boolean isAllcomplete(){boolean isComplete =
true;for(int[]each:process){
if(!complete.contains(each)){isComplete=false;}

}
```

```
return isComplete;

}

}
```

OUTPUT:-



```
FCFS CPU SCHEDULING :

pid  arrival duration  esc_time wait_time turnaroundtime
0       0       0       9               0       9

pid  arrival duration  esc_time wait_time turnaroundtime
1       1       1       5               8       13

pid  arrival duration  esc_time wait_time turnaroundtime
2       2       2       3               12      15

pid  arrival duration  esc_time wait_time turnaroundtime
3       3       3       4               14      18

total wait time : 34.0
Average Waiting Time = 8.5

total turnAroundTime : 55.0
Average turnAround Time = 13.75

throughput = 0.1904762
```

Shortest job first scheduling and Shortest job firstShortest remaining time first algorithmImplementation in JAVA:

SJF.java :

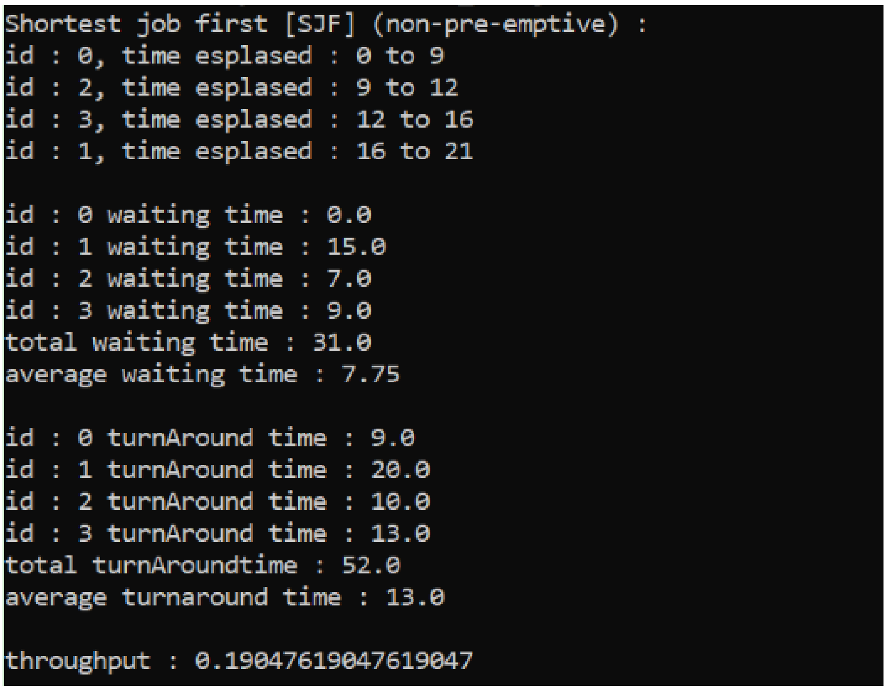
Code:

```
import java.util.*;
publicclassSJF {
    //process{id,arrival,duration}
```

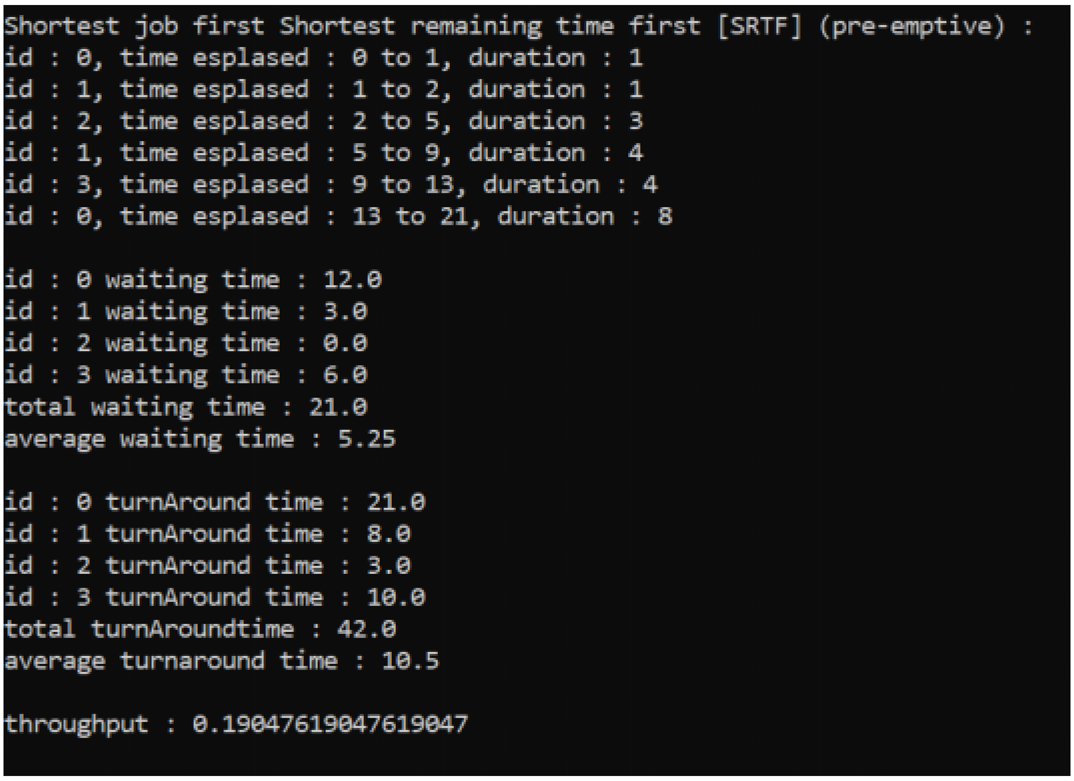
```
  finalstaticint[][]initProcess={{0,0,9},{1,1,5},{2,2,3},{3,3,4}};


    //don't modify furtherstaticint[][]process;
    static ArrayList < int[] > complete;
    staticArrayList < int[] > readyq;
    static ArrayList < int[] > sequence; //pid, start time, end
time,staticint time;


    publicstaticvoidmain(String[] args) {

        System.out.println("Shortest job first [SJF] (non-pre-emptive) :")
        nonPreemtive();

System.out.println("==========================================================
======");

        System.out.println("Shortest job first Shortest remaining time fir
[SRTF] (pre-emptive) :");
        preEmptive();
    }


    publicstaticvoidnonPreemtive() {

        process = new int[initProcess.length][initProcess[0].length];
        for (inti = 0; i < initProcess.length; i++) {
            for (int j = 0; j < initProcess[0].length; j++) {
                process[i][j] = initProcess[i][j];

            }

        }

        complete = new ArrayList < > ();
        readyq = new ArrayList < > ();
        sequence = new ArrayList < > ();
        time = 0;
        while (!isAllcomplete()) {
            updateReadyqnonContineous();
            int[] pexecute = readyq.get(0);
            for (inti = 0; i < readyq.size(); i++) {
                if (readyq.get(i)[2] < pexecute[2]) {
                    pexecute = readyq.get(i);
                }

            }

            int[] info = {
                pexecute[0],
                time,
                (pexecute[2] + time)
            };
            sequence.add(info);
            complete.add(pexecute);
            readyq.remove(pexecute);
            time += pexecute[2];
        }


        for (int[] each: sequence) {

            System.out.println("id:" + each[0] + ",timeesplased:" + each[1
+ "to" + each[2]);

        }


        System.out.println("");

        System.out.println("average waiting time : " + avgWaitingTime());
        System.out.println("");
        System.out.println("average turnaround time : " +
avgTurnAroundTime());
        System.out.println("");
        System.out.println("throughput:" + throughPut());


    }


    publicstaticvoidpreEmptive() {

        process = new int[initProcess.length][initProcess[0].length];
        for (inti = 0; i < initProcess.length; i++) {
            for (int j = 0; j < initProcess[0].length; j++) {
                process[i][j] = initProcess[i][j];
            }
        }

        complete = new ArrayList < > ();
        readyq = new ArrayList < > ();
        sequence = new ArrayList < > ();
        time = 0;
        while (!isAllcomplete()) {
            updateReadyq();

            int[] pexecute = readyq.get(0);
            for (inti = 0; i < readyq.size(); i++) {
                if (readyq.get(i)[2] < pexecute[2]) {
                    pexecute = readyq.get(i);
                }

            }

            if (pexecute[2] == 0) {
                complete.add(pexecute);
                readyq.remove(pexecute);
                time--;
            } else {

                if (!sequence.isEmpty()) {

                    int[] prevProcess = sequence.get(sequence.size() - 1);
                    if (prevProcess[0] == pexecute[0]) {
                        int[] info = {
                            pexecute[0],
                            prevProcess[1],
                            (prevProcess[2] + 1)
                        };
                        sequence.set((sequence.size() - 1), info);
                        pexecute[2] -= 1;

                    } else {

                        int[] info = {
                            pexecute[0],
                            time,
                            (time + 1)
                        };
                        sequence.add(info);
                        pexecute[2] -= 1;
                    }

                } else {

                    int[] info = {
                        pexecute[0],
                        time,
                        (time + 1)
                    };
                    sequence.add(info);
                    pexecute[2] -= 1;

                }

            }

            time++;

        }

        for (int[] each: sequence) {

            System.out.println("id:" + each[0] + ",timeesplased:" + each[1
+ "to" + each[2] + ",duration:" + (each[2] - each[1]));

        }


        System.out.println("");

        System.out.println("average waiting time : " + avgWaitingTime());
        System.out.println("");
        System.out.println("average turnaround time : " +
avgTurnAroundTime());
        System.out.println("");
        System.out.println("throughput:" + throughPut());

    }


    publicstaticdoublethroughPut() {

        double endTime = sequence.get(sequence.size() - 1)[2];
        return (((double) process.length) / endTime);
    }
    public static double avgWaitingTime() {
        doublewt = 0;
        for (int[] each: process) {
            wt += waitingTime(each[0]);

        System.out.println("id:" + each[0] + "waitingtime:" +
waitingTime(each[0]));

        }

        System.out.println("total waiting time : " + wt);
        return (wt / process.length);

    }

    public static double avgTurnAroundTime() {
        doublett = 0;
        for (int[] each: process) {

            tt += turnAroundTime(each[0]);

      System.out.println("id:" + each[0] + "turnAroundtime:" +
turnAroundTime(each[0]));

        }

        System.out.println("total turnAroundtime : " + tt);
        return (tt / process.length);
    }

    public static double waitingTime(int id) {
        doublewt = 0;
        doublestartTime = 0;

        //setarrivaltime
        for (int i = 0; i < process.length; i++) {
            if (process[i][0] == id) {
                startTime = process[i][1];
                break;
            }
        }
        //calculatewaitingtime

        for (int i = 0; i < sequence.size(); i++) {
            int[] p = sequence.get(i);
            if (p[0] == id) {

                wt += (((double) p[1]) - startTime);
                startTime = (double) p[2];
            }
        }
        return wt;
    }
    public static double turnAroundTime(int id) {
        doublearrival = 0;
        doublefinish = 0;

        for (int i = 0; i < process.length; i++) {
            if (process[i][0] == id) {
                arrival = (double) process[i][1];
                break;
            }

        }

        for (int i = 0; i < sequence.size(); i++) {
            int[] p = sequence.get(i);
            if (p[0] == id) {

                finish = (double) p[2];

            }

        }

        return (finish - arrival);

    }

    public static void updateReadyqnonContineous() {
        for (int[] each: process) {
            if (!readyq.contains(each) && each[1] <= time &&
!complete.contains(each)) {
                readyq.add(each);
            }

        }

    }
```

```
public static void updateReadyq() {
    for (int[] each: process) {
        if (!isInReadyQueue(each[0]) && each[1] == time) {
            readyq.add(each);
        }

    }

}

public static boolean isInReadyQueue(int id) {
    if (readyq.isEmpty()) {
        returnfalse;
    }
    for (int[] each: readyq) {
        if (each[0] == id) {
            returntrue;
        }
    }

    return False;

}

public static boolean isComplete(int pid) {
    if (complete.isEmpty()) {
        return false;
    }
    for (int[] each: complete) {
        if (each[0] == pid) {
            return true;
        }

    }

    return false;

}

public static boolean isAllcomplete() {
    for (int[] each: process) {
        if (!isComplete(each[0])) {
            return false;
        }
        }

        return true;

    }

}
```

OUTPUT:



```
Shortest job first [SJF] (non-pre-emptive) :
id : 0, time esplased : 0 to 9
id : 2, time esplased : 9 to 12
id : 3, time esplased : 12 to 16
id : 1, time esplased : 16 to 21

id : 0 waiting time : 0.0
id : 1 waiting time : 15.0
id : 2 waiting time : 7.0
id : 3 waiting time : 9.0
total waiting time : 31.0
average waiting time : 7.75

id : 0 turnAround time : 9.0
id : 1 turnAround time : 20.0
id : 2 turnAround time : 10.0
id : 3 turnAround time : 13.0
total turnAroundtime : 52.0
average turnaround time : 13.0

throughput : 0.19047619047619047
```



```
Shortest job first Shortest remaining time first [SRTF] (pre-emptive) :
id : 0, time esplased : 0 to 1, duration : 1
id : 1, time esplased : 1 to 2, duration : 1
id : 2, time esplased : 2 to 5, duration : 3
id : 1, time esplased : 5 to 9, duration : 4
id : 3, time esplased : 9 to 13, duration : 4
id : 0, time esplased : 13 to 21, duration : 8

id : 0 waiting time : 12.0
id : 1 waiting time : 3.0
id : 2 waiting time : 0.0
id : 3 waiting time : 6.0
total waiting time : 21.0
average waiting time : 5.25

id : 0 turnAround time : 21.0
id : 1 turnAround time : 8.0
id : 2 turnAround time : 3.0
id : 3 turnAround time : 10.0
total turnAroundtime : 42.0
average turnaround time : 10.5

throughput : 0.19047619047619047
```