Scheduling Algorithm : First Come First Serve

Scheduling algorithm is used by CPU scheduler to select a process . There are many types of scheduling algorithm but we will discuss about the most common algorithm FCFS i.e. First come and First Serve .
By applying this scheduling algorithm , the CPU makes sure that the process which is run by user are lined
in queue , just like the queue for buying tickets of movie . The person who comes first , will have the chance to get the ticket , similarly , if CPU is idle and CPU is using First come and First Serve algorithm then ,it
executes the process which arrives first .

Here , User can calculate the average turnaround time and average waiting time along with the starting and finishing time of each process

Turnaround time : Its the total time taken by the process between starting and the completion

Waiting time: Its the time for which process is ready to run but not executed by CPU scheduler

for example ,
we have three processes

	Burst time	Waiting time	Turnaround time
P1	18	0	18
P2	5	18	23
P3	7	23	30

So here we can see the turnaround time for the process 1 is 18 while 23 and 30 for 2nd and 3rd process

Gantt chart for the turnaround time is

```
 |-------------------------------------------------|-----------------|----------------------|
 |                    P1                               |        P2               |        P3
 |--------------------------------------------------|-----------------|-----------------------|
0                                                           18            23
        30
```

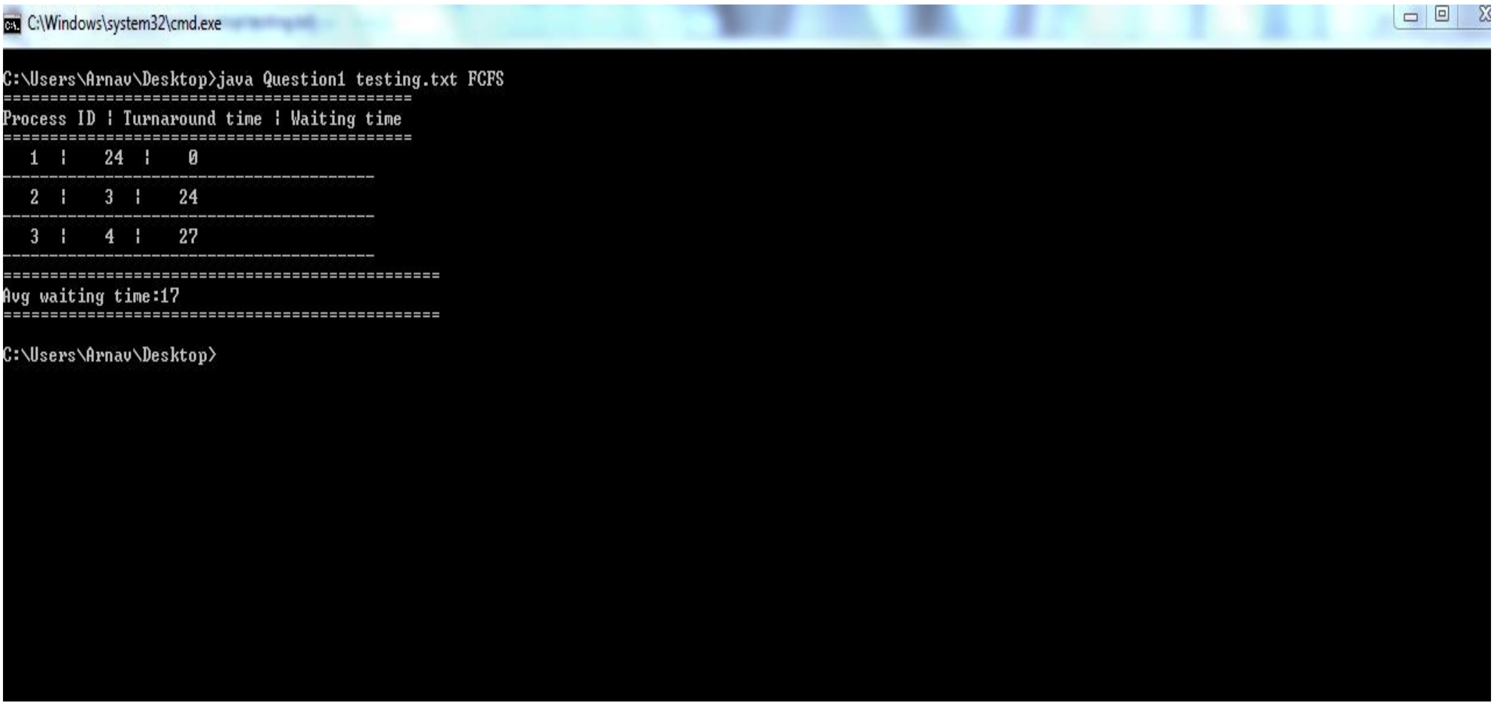
A Gantt chart is a chart which shows the start and finish times of all the processes .

Also first come first serve algorithm is non preemptive so if the process starts then the other process has to wait in the queue till the executing process finishes .

The major features of the First come first serve algorithm is that

* Throughput is low as the large process is holding up the Central processing unit for execution .
* The main disadvantage of FCFS is starving . As long as all processes completes the execution then we
dont have any trouble, But the problem starts when any of the process fails to complete . The incomplete
execution of any process leads to starvation .
* Queuing is done without using any prioritization of the processes.

Demo :



```
C:\Users\Arnav\Desktop>java Question1 testing.txt FCFS
============================================
Process ID | Turnaround time | Waiting time
============================================
   1  |    24  |    0
----------------------------------------
   2  |    3  |    24
----------------------------------------
   3  |    4  |    27
----------------------------------------
===============================================
Avg waiting time:17
===============================================

C:\Users\Arnav\Desktop>
```

Source code:

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import java.util.Queue;

/* implement this class for all three strategies */

public abstract class AllocationStrategy {
    protected List<Job> Jobs;
    protected ArrayList<Job> Queue;

    public AllocationStrategy(List<Job> jobs) {
        super();
        Jobs = jobs;
    }

    public abstract void run();
    // update current job by 1 tick
    // check if the job queue might need to be changed.
    // check for jobs to add to the queue
}
```

FirstComeFirstServed.java

```
import java.util.ArrayList;
import java.util.List;


public class FirstComeFirstServed extends AllocationStrategy {

    int temp;
    int proceessArrivalTime;
    int waitingTime;
    double avgWaitingTime;
    double avgTurnAroundTime;

    public FirstComeFirstServed(List<Job> jobs) {
        super(jobs);
    }

    @Override
    public void run() {
    }

    public void run(List<Job> jobList) {
        int count = 0;
        System.out.println("============================================ ");
        System.out.println("Process ID | Turnaround time | Waiting time ");
        System.out.println("============================================ ");
        for(Job job:jobList){
            if(count==0){
                job.processArrivalTime = job.getArrivalTime();
                job.ProcessCompletionTime = job.getArrivalTime()
+job.getCpuTime();
                }else{
                job.processArrivalTime = temp-job.getArrivalTime();
                job.ProcessCompletionTime = temp+job.getCpuTime();
            }

            temp = job.ProcessCompletionTime;
            job.turnAroundTime = temp-job.getArrivalTime();
            job.waitingTime = job.turnAroundTime-job.getCpuTime();
            count++;

            avgWaitingTime =  avgWaitingTime+job.waitingTime;
            avgTurnAroundTime = avgTurnAroundTime+job.turnAroundTime;
            System.out.println("   "+job.getProcessId()+"  | "+"    
"+job.turnAroundTime+"  | "+"   "+job.waitingTime+" ");
            System.out.println("----------------------------------------");
        }
        System.out.println("===============================================");
        System.out.println("Avg waiting time:"+avgWaitingTime/jobList.size());
        System.out.println("===============================================");
        System.out.println("Avg turn around 
time:"+avgTurnAroundTime/jobList.size());
        System.out.println("===============================================");

    }

}
```

Job.java

```
public class Job {
    private int id, submitTime, CPUTime, CPUTimeLeft;
```

```java
private int startTime = 0, endTime = 0;


public int ProcessCompletionTime;
public int processArrivalTime;
public int waitingTime;
public int turnAroundTime;
private JobFinishEvent evt;

private int arrivalTime,cpuTime,processId;

public Job(int id, int submitTime, int CPUTime, JobFinishEvent evt) {
    super();
    this.id = id;
    this.submitTime = submitTime;
    this.CPUTime = CPUTime;
    this.CPUTimeLeft = CPUTime;
    this.evt = evt;
}

public Job(int processId, int arrivalTime, int cpuTime) {

    this.processId = processId;
    this.arrivalTime = arrivalTime;
    this.cpuTime = cpuTime;

}

public void start(int sysTime) {
    startTime = sysTime;
}

public void tick(int sysTime) {
    CPUTimeLeft --;
    if (CPUTimeLeft <= 0){
        endTime = sysTime;
        evt.onFinish(this);
    }

}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public int getSubmitTime() {
    return submitTime;
}

public void setSubmitTime(int submitTime) {
    this.submitTime = submitTime;
}

public int getCPUTime() {
    return CPUTime;
}

public void setCPUTime(int cPUTime) {
    CPUTime = cPUTime;
}

public int getCPUTimeLeft() {
    return CPUTimeLeft;
}

public void setCPUTimeLeft(int cPUTimeLeft) {
    CPUTimeLeft = cPUTimeLeft;
}

public int getStartTime() {
    return startTime;
}

public void setStartTime(int startTime) {
    this.startTime = startTime;
}

public int getEndTime() {
    return endTime;
}

public void setEndTime(int endTime) {
    this.endTime = endTime;
}

public int getArrivalTime() {
    return arrivalTime;
}

public void setArrivalTime(int arrivalTime) {
    this.arrivalTime = arrivalTime;
}

public int getCpuTime() {
    return cpuTime;
}

public void setCpuTime(int cpuTime) {
    this.cpuTime = cpuTime;
}

public int getProcessId() {
    return processId;
}

public void setProcessId(int processId) {
    this.processId = processId;
}


}
```

JobFinishEvent.java

```java
public interface JobFinishEvent {
    public void onFinish(Job j);
}
```

Question1.java

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
* Application class for Assignment 1, Question 1, compsci215 2013
*
* @author dber021
*
*/
public class Question1 {

    public static void main(String[] args) {
        // Process command line arguments
        // read the file
        Scanner sc = new Scanner(System.in);
        Scanner sc1 = new Scanner(System.in);
        Scanner sc2 = new Scanner(System.in);


        String filename ;
        String allocationStrategy;
        int quantum=20;

        /*filename = args[0];
        allocationStrategy = args[1];*/

        filename = "testing.txt";
        allocationStrategy = "FCFS";


        //filename = sc.nextLine();

        if(args.length==3){
            quantum = new Integer(args[2]);
        }

        BufferedReader br = null;

        try {

            String sCurrentLine;

            br = new BufferedReader(new
FileReader("C://Users/Arnav/Desktop/"+filename));
            //System.out.println("processId  arrivalTime  cpuTime");

            List<Job> jobList = new ArrayList<Job>();
            while ((sCurrentLine = br.readLine()) != null) {

                String a[] = sCurrentLine.split(",");
                int processId = new Integer(a[0]);
                int arrivalTime = new Integer(a[1]);
                int cpuTime = new Integer(a[2]);

                Job job = new Job(processId,arrivalTime,cpuTime);

                jobList.add(job);

                //System.out.println(processId+" "+ arrivalTime+" " + cpuTime);
            }

            if("FCFS".equalsIgnoreCase(allocationStrategy)){

                FirstComeFirstServed firstComeFirstServed = new
FirstComeFirstServed(jobList);
```

```
                firstComeFirstServed.run(jobList);

                }else if("SRT".equalsIgnoreCase(allocationStrategy)){

                ShortestRemainingTime shortestRemainingTime = new
ShortestRemainingTime(jobList);
                shortestRemainingTime.run(jobList);

                }else if("RR".equalsIgnoreCase(allocationStrategy)){

                RoundRobin roundRobin = new RoundRobin();
                roundRobin.run(jobList,quantum);

            }

            } catch (IOException e) {
            e.printStackTrace();
            } finally {
            try {
                if (br != null)br.close();
                } catch (IOException ex) {
                ex.printStackTrace();
            }
        }

        JobFinishEvent callback = new JobFinishEvent() {
            @Override
            public void onFinish(Job j) {
                // this will be called when a job is finished.
            }
        };


        /*// example job addition:
        ArrayList jobs = new ArrayList();
        jobs.add(new Job(1, 0, 2, callback));
        jobs.add(new Job(2, 1, 3, callback));
        FirstComeFirstServed fcfs = new FirstComeFirstServed(jobs);
        fcfs.run();
        */
    }

}
```