OS project (scheduler)

# Team members :

## *Leader*: Ziyad mohammed fallatah sID: 444000314

## Abdulrahman Ahmed Yateem sID: 444005937

## *Author*: AL Hussain Mohamed Lohmadi sID: 444001990

## Eyad Kamil Sughayyir sID: 444003001

# Design Decisions

processID: int

burstTime: int

arrivalTime: int

waitingTime: int

turnAroundTime: int

responseTime: int

Note: Each Process should have it's own variables of above

Program as whole is responsibe for the total time

* class Process

Handle each process alone

Process should not generate arrival

\*exception "sjf"

* class Scheduler

Handle All processes

Program is the generator of arrival

Program can reset Arrival if another operation is done

can have multiple options -algorithms- to work

# Implementation

* Shortest-Job-First (Al Hussain Lohmadi) Classes: -process: this class includes the main attributes for each process, each with it SET and GET method. Writing a class for process helped me output-ing the waiting time, turn around time and ID. private String PId (process Id)

private int BurstTime

private int arrival

private int PStart (process start)

private int PFinish (process finish)

private float turnAround

private float waiting

Method:

- calcwaiting(procces[]array,int proNum)

A method to calculate the waiting time

WT = TAT - AT "ARRIVAL TIME"

AWT = WT/proNum (total process)

- calcTrnAround(procces[]array,int proNum)

A method to calculate the turn around time

TAT = PFinish – AT

avgTAT = TAT "for all process" / proNum

-clacStartFinish(procces[]array)

Is done after the array is sorted

-sort(procces[]array)

Sort the array from the start than sort it

from the end

* RoundRobin (Ziyad mohammed fallatah)

- The class RoundRobin is the responsible of handling Processes

- RoundRobin will accept burstTime and create process inside RoundRobin

- RoundRobin will start tick with quantum and maintain if quantum is bigger then the remaining burst time

- RoundRobin will send Queue<RoundRobinProcess> which will be sent to RoundRobinScreen

- RoundRobinScreen class is the responsible of (printing / reading) input from user and it can accept Queue<RoundRobinProcess>

- One of the RoundRobinProcesses

* First came first serve (Abdulrahman Ahmed Yateem)

Description:

1. \*\*Number Of process Method\*\*: Asks the user to enter the number of processes and returns this number.

2. \*\*Burst time Method\*\*: Asks the user to enter the burst time (how long each process takes to run) for each process and stores these times in an array, then returns the array.

3. \*\*Waiting time Method\*\*: Calculates the waiting time for each process based on the burst times. The first process waits for 0 time, and each next process waits for the total time of all previous processes. It returns an array of waiting times.

4. \*\*Avg Turnaround Time Method\*\*: Calculates the average turnaround time. Turnaround time for each process is its waiting time plus its burst time. This method adds up all turnaround times and divides by the number of processes.

5. \*\*Avg Waiting Time Method\*\*: Calculates the average waiting time. This method adds up all waiting times and divides by the number of processes.

6. \*\*print Method\*\*: Prints a table showing each process's burst time, waiting time, and turnaround time. It also prints the average turnaround time and average waiting time.

* Priority (Eyad Kamil Sughayyir)

Description:

1. \*\*swap1 Method\*\*: This method swaps two elements in an integer array. It takes an array and two indices `i` and `j` and swaps the values ​​at these positions.

2. \*\*swap2 Method\*\*: Like `swap1`, this method swaps two elements in a string array. It takes an array and two indices `i` and `j` and swaps the values ​​at these positions.

3. \*\*rankp Method\*\*: This method sorts the processes based on their priority. It takes three arrays as input: `BursT(burst times), `priority` (priorities), and `process Name` (names of the processes). The method comes sorts these arrays such that the process with the highest priority (smallest priority number) first. It uses a nested loop and the `swap1` and `swap2` methods to achieve this.

4. \*\*countingT Method\*\*: This method calculates the waiting times and turnaround times for the processes. It takes five arrays as input: `BurstT` (burst times), `priority` (priorities), `processName` (names of the processes), `witintT` (waiting times), and `turnAroundT` (turnaround times). It initializes the current time to 0 and iterates through each process, updating the waiting time, turnaround time, and current time.

5. \*\*final print method\*\*: This method prints the details of each process. It takes five arrays as input: `BurstT` (burst times), `priority` (priorities), `processName` (names of the processes), `witintT` (waiting times), and `turnAroundT` (turnaround times). It prints a table showing the process name, burst time, priority, waiting time, and turnaround time for each process.

6. \*\*AverageWaitingTime Method\*\*: This method calculates the average waiting time. It takes an array of waiting times (`witintT`) as input, sums all the waiting times, and divides by the number of processes.

7. \*\*AverageTurnaroundTime Method\*\*: This method calculates the average turnaround time. It takes an array of turnaround times (`turnAroundT`) as input, sums all the turnaround times, and divides by the number of processes.

In summary, the `priority` class sorts of processes based on their priority, calculates waiting and turnaround times, prints the results, and computes the average waiting and turnaround times. It uses methods to swap elements in arrays, sort processes, and perform calculations.

* Examples:
* SJF:

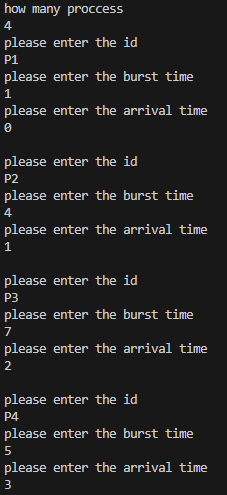
INPUT

P1 burst time = 1 AT = 0

P2 burst time = 4 AT = 1

P3 burst time = 7 AT = 2

P4 burst time = 5 AT = 3



OUTPUT

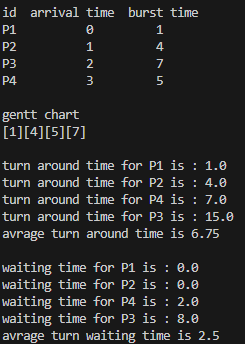
P1TAT = 1 , WT = 0

P2 TAT = 4 , WT = 0

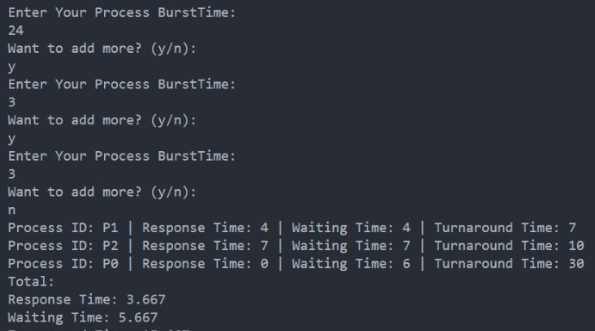
P3 TAT = 7 , WT =2

P4 TAT = 5 , WT = 8

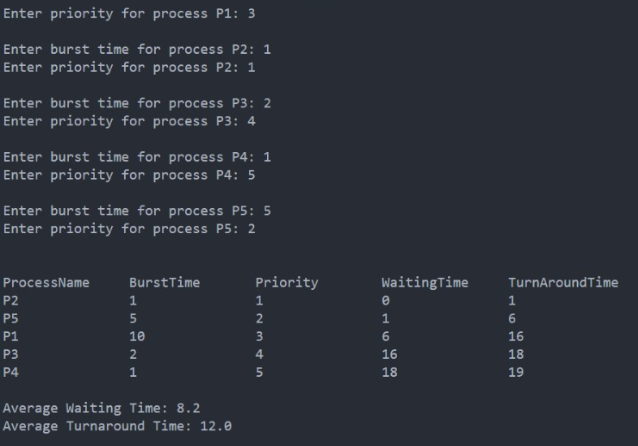
AVREG TAT = 6.75 , AVREG WT = 2.5



* Round robin



* Priority



* FCFS

