Operating Systems Project

Instructor: Dr. Soha Safwat
TA: Eng. Nancy El-Arabawy

- Submission Date: Wed 14/12/200

Mohamed Taha, Youssef Hossam, Ahmed Mahmoud, Abdelrhman Gaber
192000280 192000175 192000193 192000272

Table of Contents

List of figures

Workload Distribution

Name	ID	Assign	
Youssef Hossam	192000175	HPF-Preemptive	SJF-Nonpreemptive
Mohamed Taha	192000280	Read From File	RR
Ahmed Mahmoud	192000193	HPF-NonPreemptive	FCFS
Abdulrahman Gaber	192000272	SRTF-Preemptive	Report

Glossary Table

Parameter	Description
P.	- **Process -** Is a running program
P.list	- **Process list –** A list contains all processes to be executed
Ai	- **Arrival Time -** Time at which the process enters ready queue
Ti	- **Timer -**
Fi	- **Final Time** - Time at which the process is terminated
Wi	- **Initial Burst -** Time at which the process requires to terminate
F	- **Pointer -**
RQ	- **Ready Queue -** Keeps a set of all processes waiting to execute
Burst	- **Burst Time -** Time left for the process to terminate
SCH	- **Scheduler list -**
TATi	- **Turn Around Time -** Time which the process took to terminate
W	- **Waiting Time -** when the process is ready & not being executed
Q	- **Quantum -** Time given to each process to be executed in RR
Clk	- **Counter -** To calculate the context switch time

INTRO

The OS scheduler is a vital component int the OS.

It performs CPU scheduling which is the task of selecting a process out of a set of waiting processes in the ready queue and allocate the CPU to it based on a certain criterion; mainly the chosen algorithm and whether it's preemptive or non-preemptive.

This dispatcher is the component of the OS scheduler that assigns CPU resources to the chosen process. It is required to implement an OS scheduler using different scheduling algorithms that we learned in our course.

The OS scheduler should have a read-from-file module that takes an input text file and read processes and AT, BT and Priority either by manually writing the file or the input file will be the output of process generator module in case you implement process generator module

2 Classes are made (Process , Scheduler) ,
& Program (main file) -in summary-

1- Process Class contains the following members :
{ Id , Ai , Burst , Priority , Fi , Wi } & the following functions
Ai sort - Burst sort - Priority sort – TATi – W

When a Process Is created it gets it's own copy from these members

2- Scheduler Class contains all algorithms to be used to execute and terminate the processes :
FCFS , HPF-Nonpreemptive , HPF- Preemptive , SJF-Nonpreemptive , SRTF- Preemptive

In the following pages we will talk about each algorithm and how it is implemented

FCFS

Definition

FCFS Scheduling algorithm automatically executes the queued processes and requests in the order of their arrival.

Pseudocode

1-The definitions

Inherits All Process class members + Ti =0

IDEAL list

2-step 1:

Sort all prosses by their arrival time

3-step 2:

By using For loop with counter =0 & less than no. of prosses

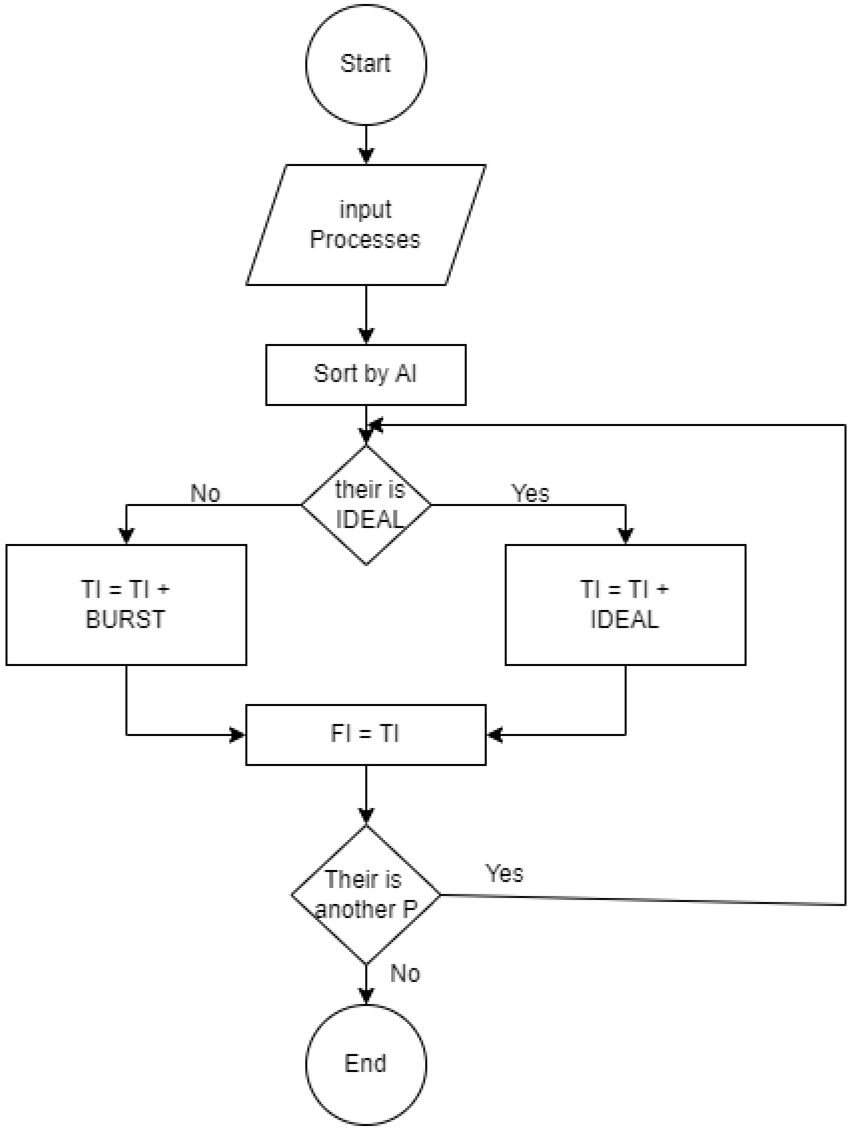
An IF condition is thrown to check if there is an IDEIAL or not If TRUE , TI = TI + IDEAL … if FULSE then go to step 3

"There will be ideal if Ai>Ti"

4- step 3:

The TI = TI + BRUST & THIS NUMER is then stored in it's Fi

Flowchart



Start
input
Processes
Sort by AI
their is
IDEAL
No
Yes
TI = TI +
BURST
TI = TI +
IDEAL
FI = TI
Their is
another P
Yes
No
End

HPF-NonPreemptive

Definition

is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority.

Pseudocode

1-The definitions

Inherits All Process class members + Ti =0

, RQ , IDEAL & Sch lists , F “pointer”

2-step 1:

There are now 2 cases , 1) there is a P. in the Process list.
2) there is no P.

In the first case we have 4 sub-cases

3-step 2:

Case 1.1: the arrival time of P. is greater than the timer & there are P. in RQ

Add first P. in RQ to SCH then add it’s burst to the timer and store it in Fi then remove this P. from RQ and repeat as long as Ai is greater than the timer

4-step 3:

You have repeated step 3 until there are no P. left in RQ but Ai still > timer (Case 1.2)

IDEAL = Ai – timer , timer = timer + IDEAL

5-step 4:

Now the Ai is less or = to the timer & SCH !=0 (Case 1.3).

Add this P. to RQ the sort it then remove this P.

From the process list.

6-step 5:

This case is for the first P. to arrive with Ai<=timer & there is no P. in SCH (Case 1.4).

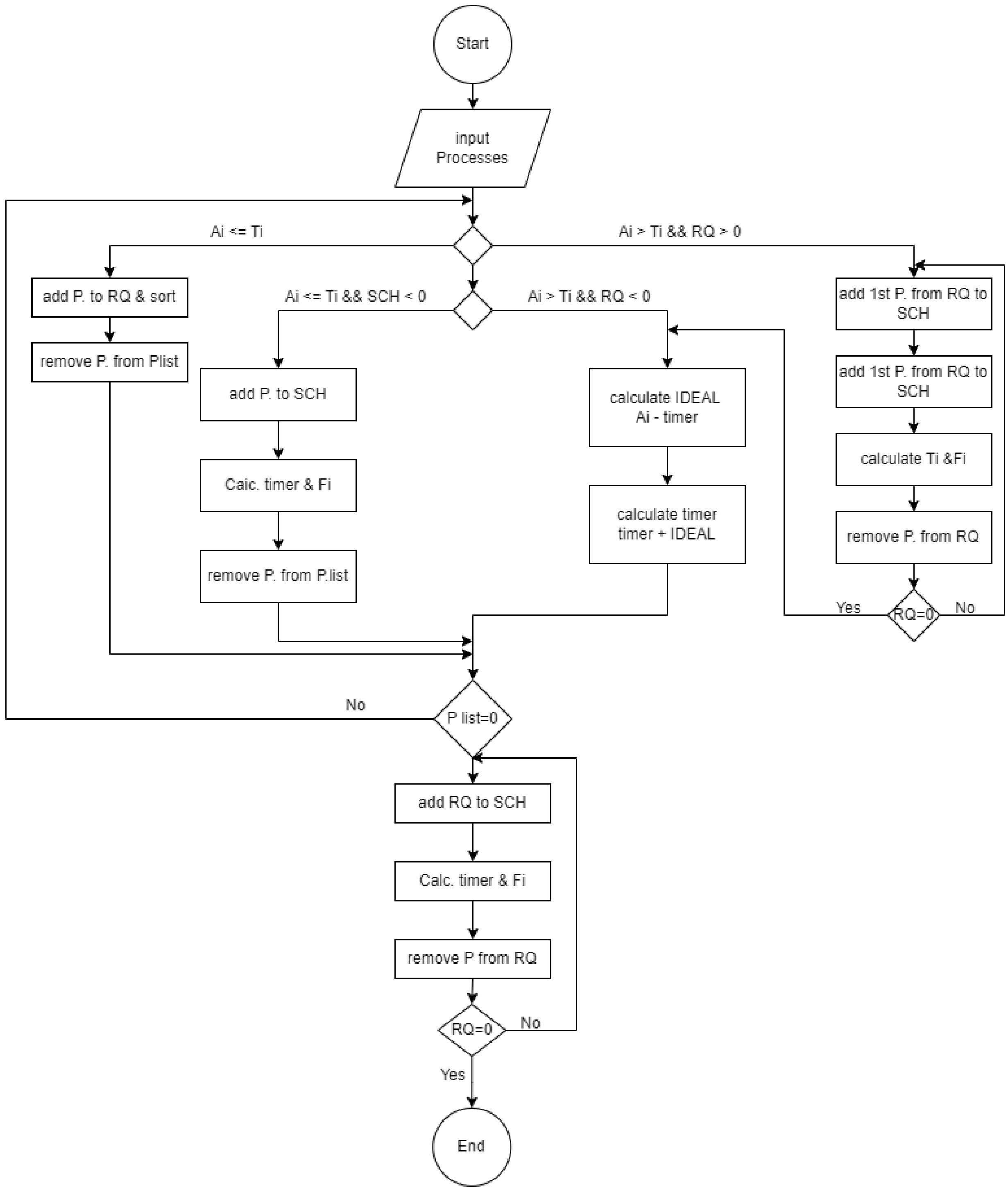
Add this P. to SCH , timer += burst and Fi= the new timer , remove this P. from process list

7-step 6:

Back to case 2 , there are no P. in Process list

Add first P. from RQ to SCH and calculate the new timer and Fi as usual then remove it from RQ and repeat till you finish all the P.

Flowchart



Start
input Processes
Ai <= Ti
Ai > Ti && RQ > 0
add P. to RQ & sort
remove P. from Plist
Ai <= Ti && SCH < 0
Ai > Ti && RQ < 0
add P. to SCH
Caic. timer & Fi
remove P. from P.list
calculate IDEAL Ai - timer
calculate timer timer + IDEAL
add 1st P. from RQ to SCH
add 1st P. from RQ to SCH
calculate Ti &Fi
remove P. from RQ
Yes
RQ=0
No
No
P list=0
add RQ to SCH
Calc. timer & Fi
remove P from RQ
RQ=0
No
Yes
End

HPF-Preemptive

Pseudocode

1-The definitions

Inherits All Process class members + Ti =0

, RQ , IDEAL & Sch lists , F “pointer”

2-step 1:

There are now 2 cases , 1) there is a P. in the Process list.
2) there is no P.

In the first case we have 5 sub-cases

3-step 2:

Case 1.1: the arrival time of P. is greater than the timer & there are P. in RQ

Add first P. in RQ to SCH then add it’s burst to the timer and store it in Fi then remove this P. from RQ and repeat

4-step 3:

You have repeated step 3 until there are no P. left in RQ but Ai still > timer (Case 1.2)

IDEAL = Ai – timer , timer = timer + IDEAL

5-step 4:

Now the Ai is less or = to the timer & it’s priority < the priority of the rung P. in SCH (Case 1.3).

Add this P. to RQ the sort it then remove this P.

From the process list.

6-step 5:

This case is for the first P. to arrive with Ai<=timer & there is no P. in SCH (Case 1.4).

Add this P. to SCH , timer += burst and Fi= the new timer , remove this P. from process list

7-step 6:

Now the Ai is less or = to the timer & it's priority > the priority of the running P. in SCH (Case 1.5).

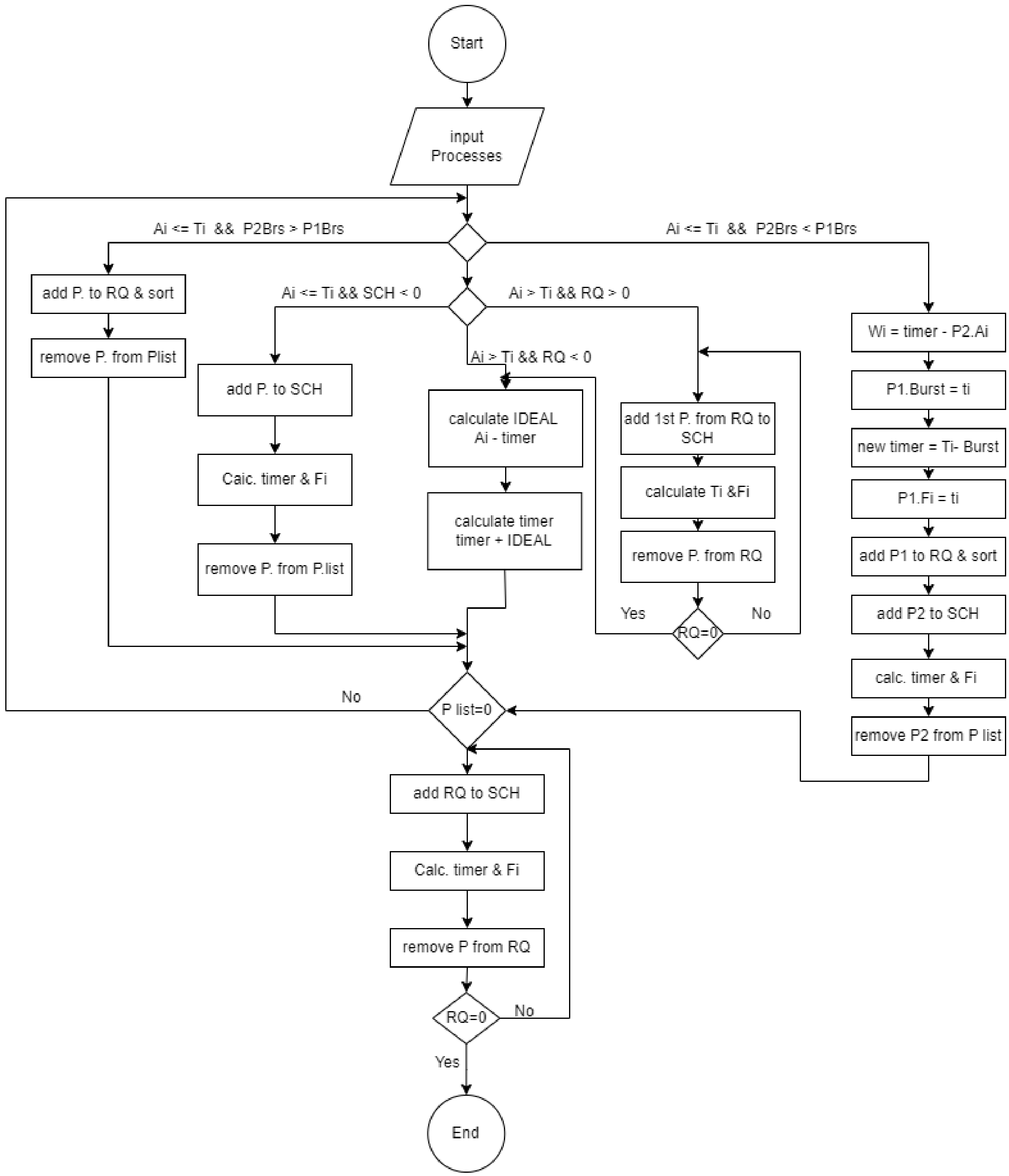
Assume a running P1 which will end at time 5 , a P2 with higher Pri. Arrived at timer = 2 , so the Burst of the P1 = Wi = timer – Ai of P2 , the new timer will be = the old one – Burst time , store the new timer as the Fi1 for P1 then push it to RQ & sort , Push P2 to SCH & remove it from P.list

8-step 6:

Back to case 2 , there are no P. in Process list

Add first P. from RQ to SCH and calculate the new timer and Fi as usual then remove it from RQ and repeat till you finish all the P.

Flowchart


Start
input Processes
Ai <= Ti && P2Brs > P1Brs
Ai <= Ti && P2Brs < P1Brs
add P. to RQ & sort
remove P. from Plist
Ai <= Ti && SCH < 0
Ai > Ti && RQ > 0
Ai > Ti && RQ < 0
add P. to SCH
Caic. timer & Fi
remove P. from P.list
calculate IDEAL
Ai - timer
calculate timer
timer + IDEAL
add 1st P. from RQ to SCH
calculate Ti &Fi
remove P. from RQ
Yes
RQ=0
No
Wi = timer - P2.Ai
P1.Burst = ti
new timer = Ti- Burst
P1.Fi = ti
add P1 to RQ & sort
add P2 to SCH
calc. timer & Fi
remove P2 from P list
No
P list=0
add RQ to SCH
Calc. timer & Fi
remove P from RQ
RQ=0
No
Yes
End

SRTF-Preemptive

Definition

is a method of scheduling processes that is based on Burst time. In this algorithm, the scheduler selects the tasks to work as per the remaining burst time.

Pseudocode

1-The definitions

Inherits All Process class members + Ti =0

, RQ , IDEAL & Sch lists , F “pointer”

2-step 1:

There are now 2 cases , 1) there is a P. in the Process list.
2) there is no P.

In the first case we have 5 sub-cases

3-step 2:

Case 1.1: the arrival time of P. is greater than the timer & there are P. in RQ

Add first P. in RQ to SCH then add it’s burst to the timer and store it in Fi then remove this P. from RQ and repeat

4-step 3:

You have repeated step 3 until there are no P. left in RQ but Ai still > timer (Case 1.2)

IDEAL = Ai – timer , timer = timer + IDEAL

5-step 4:

Now the Ai is less or = to the timer & it’s burst > the burst of the running P. in SCH (Case 1.3).

Add this P. to RQ and sort it then remove this P. From the process list.

6-step 5:

This case is for the first P. to arrive with Ai<=timer & there is no P. in SCH (Case 1.4).

Add this P. to SCH , timer += burst and Fi= the new timer , remove this P. from process list

7-step 6:

Now the Ai is less or = to the timer & it’s Brust < the Brust of the running P. in SCH (Case 1.5).

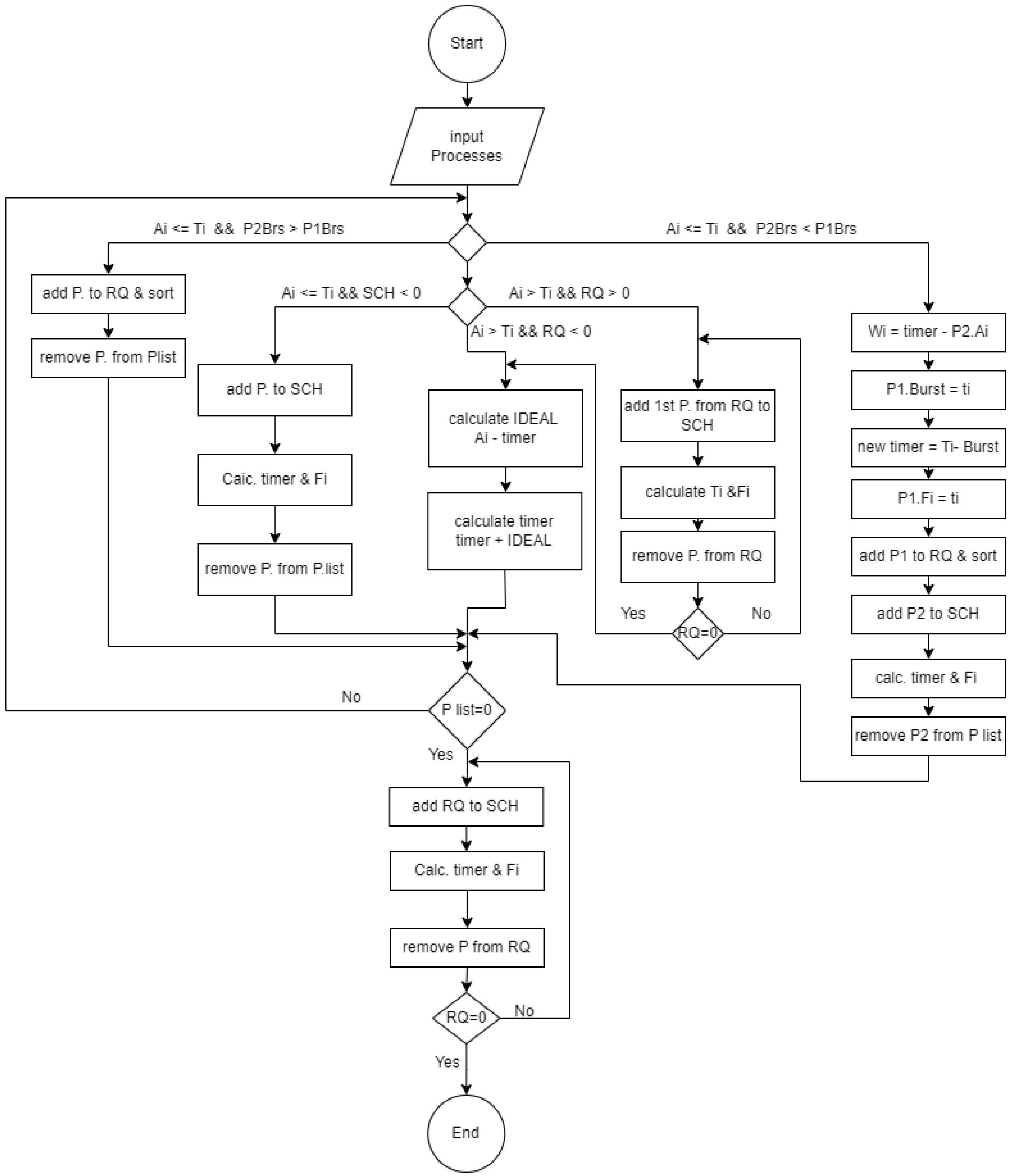
Assume a running P1 which will end at time 5 , a P2 with less Burst time . Arrived at timer = 2 , so the Burst of the P1 = Wi = timer – Ai of P2 , the new timer will be = the old one – Burst time , store the new timer as the Fi1 for P1 then push it to RQ & sort , Push P2 to SCH & remove it from P.list

8-step 6:

Back to case 2 , there are no P. in Process list

Add first P. from RQ to SCH and calculate the new timer and Fi as usual then remove it from RQ and repeat till you finish all the P.

Flowchart



Start
input Processes
Ai <= Ti && P2Brs > P1Brs
Ai <= Ti && P2Brs < P1Brs
add P. to RQ & sort
remove P. from Plist
Ai <= Ti && SCH < 0
Ai > Ti && RQ > 0
Ai > Ti && RQ < 0
add P. to SCH
Caic. timer & Fi
remove P. from P.list
calculate IDEAL Ai - timer
calculate timer timer + IDEAL
add 1st P. from RQ to SCH
calculate Ti &Fi
remove P. from RQ
Yes
RQ=0
No
Wi = timer - P2.Ai
P1.Burst = ti
new timer = Ti- Burst
P1.Fi = ti
add P1 to RQ & sort
add P2 to SCH
calc. timer & Fi
remove P2 from P list
No
P list=0
Yes
add RQ to SCH
Calc. timer & Fi
remove P from RQ
RQ=0
No
Yes
End

RR

Definition

In computer operations, round robin is used in a scheduling algorithm that distributes work evenly among all available resources. This ensures that no single resource is overworked, which can lead to errors and other issues down the line. This is often described as round robin process scheduling

Pseudocode

1-The definitions

Inherits All Process class members + Ti =0

, RQ , IDEAL & Sch lists , F “pointer” , Quantum

2-step 1:

Case 1: the arrival time of P. is greater than the timer & there are no P. in RQ

IDEAL = Ai – timer , timer = timer + IDEAL

3-step 2:

Case 2: there is no P. in P.list **&&** there are P. in RQ
(2 Sub-Cases)

Case 2.1 (Burst < Q): add P. to SCH , timer += burst , burst = 0 ,Fi = Ti then remove P. from RQ

Case 2.2 (Burst >= Q): add P. to SCH , add the Q value to timer and sub it from Burst , if new Burst =0 then Fi = Ti , if not add P. to RQ

4-step 3:

Case 2: the arrival time of P. is greater than the timer **&&** there are P. in RQ (2 Sub-Cases)

Case 2.1 (Burst < Q): add P. in RQ to SCH , timer += burst , burst = 0 ,Fi = Ti then remove P. from RQ

Case 2.2 (Burst >= Q): add P. to SCH , add the Q value to timer and sub it from Burst , if new Burst =0 then Fi = Ti , if not add P. to RQ

5-step 4:

Case 3: the arrival time of P. is less than or equal timer (2 Sub-Cases)

Case 3.1 (Burst < Q): add P. to SCH , timer += burst , burst = 0 ,Fi = Ti then remove P. from P.list

Case 3.2 (Burst >= Q): add P. to SCH , add the Q value to timer and sub it from Burst , if new Burst =0 then Fi = Ti , if not add P. to RQ then remove it from P.list

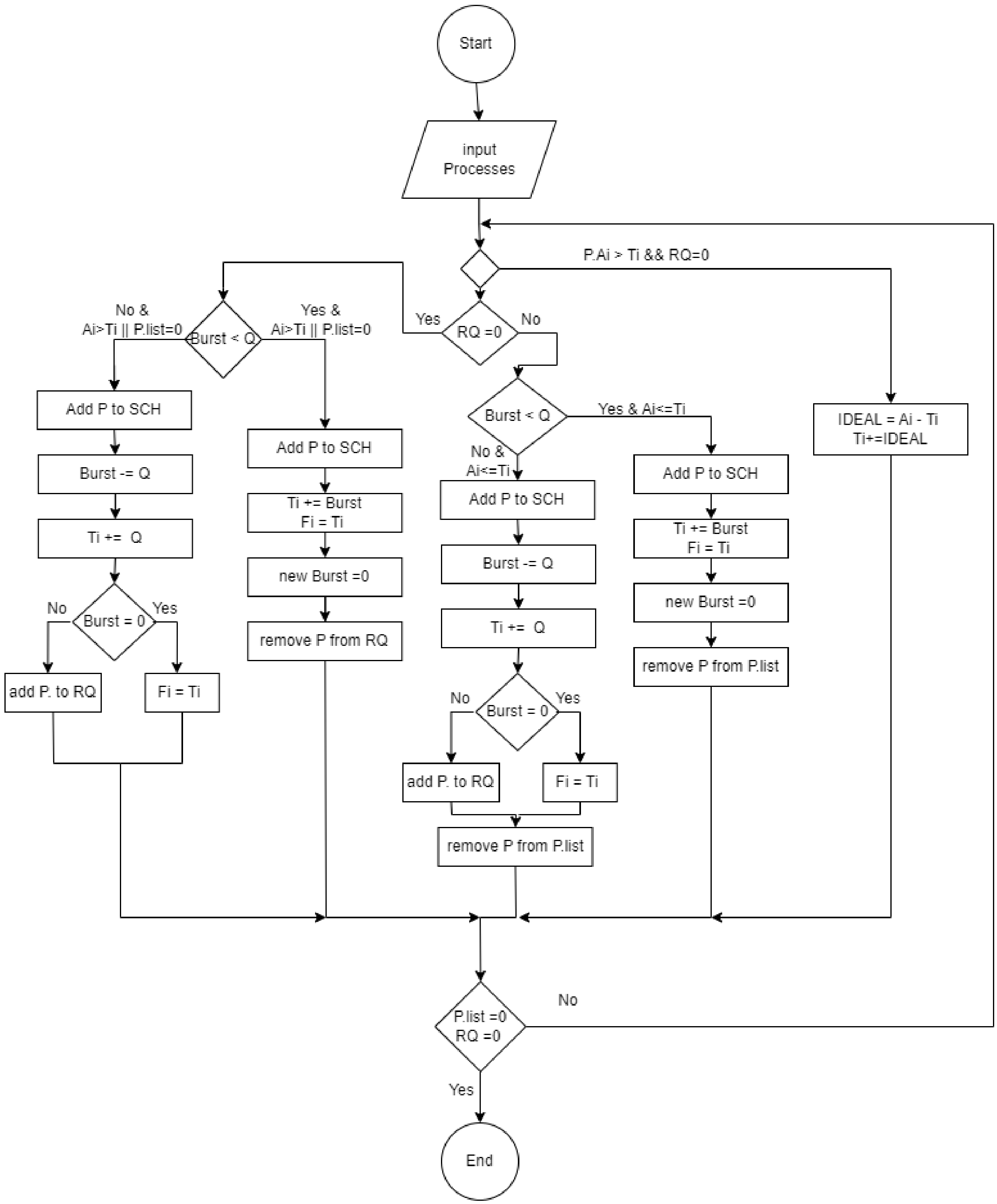
6-step 5:

Repeat all these steps until P.list & RQ =0

CASE HANDLER

When a Process leaves the SCH due to quantum time and this same process should re-enter to the SCH , a context switch with 1 second is added to the timer

Flowchart


Start
input
Processes
P.Ai > Ti && RQ=0
Yes
RQ =0
No
No &
Ai>Ti || P.list=0
Burst < Q
Yes &
Ai>Ti || P.list=0
Add P to SCH
Burst -= Q
Ti += Q
No
Burst = 0
Yes
add P. to RQ
Fi = Ti
Add P to SCH
Ti += Burst
Fi = Ti
new Burst =0
remove P from RQ
Burst < Q
Yes & Ai<=Ti
No &
Ai<=Ti
Add P to SCH
Burst -= Q
Ti += Q
No
Burst = 0
Yes
add P. to RQ
Fi = Ti
remove P from P.list
Add P to SCH
Ti += Burst
Fi = Ti
new Burst =0
remove P from P.list
IDEAL = Ai - Ti
Ti+=IDEAL
No
P.list =0
RQ =0
Yes
End

IMPORTANT NOTE

A context switch with 1 second is existed , each time a process is removed and another process enters to be executed the timer is increased by 1 second

SUMMARY

1- Processes are being entered to the program by a READ FROM FILE
2- Each process take a copy from the members of PROCESS class
3- In PROGRAM file the 6 algorithms are being called , the processes then is executed by any algorithm the user whishes to use
4- In each algorithm there are an equations to calculate Total TAT , Average TAT & Fi
5- The GUI is then used to present the output