Operating System

Our Team

01

03

04

Mohamed Rabei

Sara Adel

Faten TareK

Alaa Tarek

Martina Farag

Agenda

01

Shortest-remaining time first

Introduction

Algorithm description

Algorithm implementation

Example

Video explain code

02

FCFS CPU Scheduling

Introduction

Algorithm description

Algorithm implementation

Example

Video explain code

03

Rate monotonic scheduling

Introduction

Algorithm description

Algorithm implementation

Example

Video explain code

04

Earliest deadline first scheduling (EDF)

Introduction

Algorithm description

Algorithm implementation

Example

Video explain code

GUI

Used Programming language and IDE

How to use

Shortest-remaining time first

01

Introduction

➢ What is the Shortest remaining time first Algorithm ?

- is the preemptive version of Shortest Job Next (SJN) algorithm, where the processor is allocated to the job closest to completion.
- This algorithm requires advanced concept and knowledge of CPU time required to process the job in an interactive system, and hence can't be implemented there. But, in a batch system where it is desirable to give preference to short jobs, SRT algorithm is used.

Shortest-remaining time first

02 Algorithm description

- Advantages:

  SRTF algorithm makes the processing of the jobs faster than SJN algorithm, given it’s overhead charges are not counted.

- Disadvantages:

  The context switch is done a lot more times in SRTF than in SJN, and consumes CPU’s valuable time for processing. This adds up to it’s processing time and diminishes it’s advantage of fast processing.

Shortest-remaining time first

03 Algorithm Implementation

Variables Declaration

Asking user about data

```
#include <bits/stdc++.h>
using namespace std;
    void getAverageWaitingtine()
    {


int arrival[10],burst[10],x[10];
/*
 a[i] --> Array to take Arrival time data
 b[i] --? Array to take burst time data

*/
    int waiting[10],turnaround[10],completion[10];
    int index,smallestIndex,Counter=0,time,numOfProcess;
    double averageWaitingTime=0,tt=0,End;

    cout<<"\nEnter the number of Processes: ";  //input
    cin>>numOfProcess;
    for(index=0; index<numOfProcess; index++)
    {
        cout<<"\nEnter arrival time of process: ";  //input
        cin>>arrival[index];
    }
    for(index=0; index<numOfProcess; index++)
    {
        cout<<"\nEnter burst time of process: ";  //input
        cin>>burst[index];
    }
```

Shortest-remaining time first

03 Algorithm Implementation

To Save burst time data

```
for(index=0; index<numOfProcess; index++) // TO save burst time data in Array x;
    x[index]=burst[index];

burst[9]=9999; //to initialize the highest number into last index
for(time=0; Counter!=numOfProcess; time++)
{
    smallestIndex=9; //assign the smallest equal to 9 to go to last index;
    for(index=0; index<numOfProcess; index++)
    {
        if(arrival[index]<=time && burst[index]<burst[smallestIndex] && burst[index]>0 )
            smallestIndex=index;
    }
    burst[smallestIndex]--;

    if(burst[smallestIndex]==0)
    {
        Counter++;
        End=time+1;
        completion[smallestIndex] = End;
        waiting[smallestIndex] = End - arrival[smallestIndex] - x[smallestIndex];
        turnaround[smallestIndex] = End - arrival[smallestIndex];
    }
}
```

Check the smallest

process time

and store

it and continue

his process

Check if process

ended for process

Shortest-remaining time first

03

Algorithm Implementation

Print shape of output

Printing Average

Call method

```
    cout<<"Process"<<"\t"<< "burst-time"<<"\t"<<"arrival-time" <<"\t"<<"waiting-time" <<"\t"<<"turnaround-time"<< "\t"<<"completion-time"<<endl;
        for(index=0; index<numOfProcess; index++)
        {
            cout<<"p"<<index+1<<"\t\t"<<x[index]<<"\t\t"<<arrival[index]<<"\t\t"<<waiting[index]<<"\t\t"<<turnaround[index]<<"\t\t"<<completion[index]<<endl;
            averageWaitingTime = averageWaitingTime + waiting[index];
            tt = tt + turnaround[index];
        }
        cout<<"\n\nAverage waiting time ="<<averageWaitingTime/numOfProcess;
        cout<<"  Average Turnaround time ="<<tt/numOfProcess<<endl;

}
int main()
{
    getAverageWaitingtine();
     system("pause");
}
```

Shortest-remaining time first

04 Example

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Average waiting time = 6.5

Average Turnaround time = 13

05 Video

FCFS CPU Scheduling

01

Introduction

➢ What is the FCFS CPU Scheduling ?

- Given n processes with their burst times, the task is to find average waiting time and average turn around time using FCFS scheduling algorithm. First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue. In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

FCFS CPU Scheduling

02

How to compute below times in Round Robin using a program

1.Completion Time: Time at which process completes its execution.

2.Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time

3.Waiting Time(W.T): Time Difference between turn around time and burst time.

Waiting Time = Turn Around Time – Burst Time

FCFS CPU Scheduling

03

Algorithm Implementation

```
// Function to find the waiting time for all processes
void Find_Waiting_Time(int n, int burst_time[], int waiting_time[])
{
    // waiting time for the first process is 0
    waiting_time[0] = 0;

    // calculating waiting time
    for (int  i = 1; i < n ; i++ ){

        // to find the waiting time for one process we need to summation
        // the burst time for previous process and the total waiting time
        // for all previous processes
        waiting_time[i] =  burst_time[i-1] + waiting_time[i-1] ;
    }

     // calling the function to find the Average waiting time
    Find_Average_Waiting_Time(n, burst_time ,waiting_time);
}
```

Loop for calculating the waiting time in prefix sum array

Calling function to find Average waiting time

FCFS CPU Scheduling

03 Algorithm Implementation

```
void Find_Average_Waiting_Time(int n, int burst_time[], int waiting_time[])
{
    // the counter we will get the summation of all processes in
    int total_waiting_time = 0;

    //Display processes along with all details
    cout << "Processes  "<< " Burst time  "<< " Waiting time\n" ;

    // Calculate total waiting time
    for (int  i=0; i<n; i++)
    {
        total_waiting_time = total_waiting_time + waiting_time[i];
        cout << "    " << i+1 << "\t\t" << burst_time[i] <<"\t    "
            << waiting_time[i] <<endl;
    }

    // To Calculate Average waiting time we divide
    // the total waiting time by number of processes
    cout << "Average waiting time = " << (float)total_waiting_time / (float)n <<endl;;

}
```

Calculating the summation of all waiting times

Print shape of output

Calculating the Average waiting time and using float here to handle if that summation not divisible by number of processes

FCFS CPU Scheduling

03 Algorithm Implementation

```
// Driver code
int main()
{
    // size of processes
    int n;
    cin>>n;

    // Burst time of all processes and the process id's
    // is the index of the arrey
    int  burst_time[n];

    // The array which we will save the waiting time in,
    // for all processes
    int waiting_time[n];

    for(int i=0;i<n;i++){
        cin>>burst_time[i];
    }

    // calling the function to find the waiting time for all processes
    Find_Waiting_Time(n, burst_time ,waiting_time);
    return 0;
}
```

Getting the number of processes

Loop for getting the burst time from user

Calling function to calculate the waiting time for all processes

FCFS CPU Scheduling

04 Example

Process	Burst Time
P1	24
P2	3
P3	3

Average waiting time = 17

05 Video

Rate-monotonic scheduling

01 Introduction

➢ What is the Rate-monotonic scheduling?

- is a priority algorithm that belongs to the static priority scheduling category of Real Time Operating Systems. It is preemptive in nature. The priority is decided according to the cycle time of the processes that are involved. If the process has a small job duration, then it has the highest priority. Thus if a process with highest priority starts execution, it will preempt the other running processes. The priority of a process is inversely proportional to the period it will run for.

Rate-monotonic scheduling

02

Algorithm description

- Advantages:

1. It is easy to implement.
2. If any static priority assignment algorithm can meet the deadlines then rate monotonic scheduling can also do the same. It is optimal.
3. It consists of calculated copy of the time periods unlike other time-sharing algorithms as Round robin which neglects the scheduling needs of the processes.

- Disadvantages:

It is very difficult to support aperiodic and sporadic tasks under RMA.

RMA is not optimal when tasks period and deadline differ.

Rate-monotonic scheduling

03 Algorithm Implementation

Check schedulability of the system

```
bool check(){
    double sum = 0;
    for(int i = 0; i < processCount; ++i)
        sum += (double) process[i].first / (double) process[i].second;

    //if sum <= 1, then the set of processes MAY be Schedulable .. Otherwise it cannot be scheduled
    return sum <= 1;
}

bool check_deadline(){
    for(int i = 0; i < processCount; ++i){
        /*
        for any process i: if the execution time exceeded the time period
        then process i has missed it's deadline which means that This set of processes is not schedulable..
        */
        if(updatedProcess[i].first > updatedProcess[i].second)
            return 1;
    }
    return 0;
}

//get the index of the next process to be executed
//Processes are assigned priorities inversely based on it's Time Period
int getPriority(){
    int leastPeriod = lcm, index = -1;
    /*
    variable index will store the index of the Process which has the highest Priority
    -> The shorter Time Period a Process has, The Higher Priority it will be assigned
    */
    for(int i = 0; i < processCount; ++i){
        if(updatedProcess[i].first == 0) continue;

        if(updatedProcess[i].second < leastPeriod)
            leastPeriod = updatedProcess[i].second, index = i;
    }
    return index;
}
```

Check if any process

has missed it's deadline

Asking user about data

Rate-monotonic scheduling

03 Algorithm Implementation

```cpp
void schedule(){
    for(int i = 0; i < processCount; ++i)
        updatedProcess.push_back(process[i]);

    ofstream file;
    file.open("output.txt");
    file << "\n";
    for(int time = 0; time < lcm; ++time){

        //Check if any process has missed it's deadline
        bool missed_deadline = check_deadline();
        if(missed_deadline){
            file << "This System is not schedulable!\n";
            return;
        }

        //Index of the next process to be executed based on it's priority
        int nxtProcess = getPriority();


        //if we found a process to be executed
        if(~nxtProcess)
            file << "\tTime [" << time <<" , "<< time + 1 << "] : " << "Process " << nxtProcess + 1 <<"\n";

        //else there is no process ready to be executed
        else
            file << "\tTime [" << time <<" , "<< time + 1 << "] : " << "NO Process Is Running\n";


       //reduce the execution time of the running process by 1 after each time unit
       if(~nxtProcess)
           updatedProcess[nxtProcess].first --;

       for(int i = 0; i < processCount; ++i){
           //reduce the time period of each process by 1 after each time unit
           updatedProcess[i].second --;

           /*
           if the current period of the process is completely finished,
           reset the process data (restore the original execution time & time period of the process)
           */
           if(!updatedProcess[i].second)
               updatedProcess[i] = process[i];
       }
   }
   file << "\n";
   file.close();
```

Find the process

with the highest priority

to be executed

Rate-monotonic scheduling

03 Algorithm Implementation

reads the processes' data,

checks for schedulability,

then schedule the set of

processes If possible

```
//read data
processCount = atoi(argv[1]);

int executionTime, timePeriod;
for(int i = 2; i < 2 * processCount + 2; i += 2){
    executionTime = atoi(argv[i]), timePeriod = atoi(argv[i + 1]);
    process.push_back({executionTime, timePeriod});
}


//check if the system can be scheduled or not
bool schedulable = check();

//if it cannot be scheduled, report this and terminate the program
if(!schedulable){
    ofstream file;
    file.open("RMA.txt");
    file << "This System is not schedulable!\n";
    file.close();
    return 0;
}

//Compute the Least Common Multiple of the Time periods of the whole set of processes
lcm = process[0].second;
for(int i = 1; i < processCount; ++i)
    lcm = LCM(lcm, process[i].second);

//schedule the set of processes
schedule();
return 0;
```

Rate-monotonic scheduling

04 Example

Process	Execution Time	Time period
P1	3	20
P2	2	5
P3	2	10

05 Video

Earliest Deadline First Scheduling (EDF)

01

Introduction

➢ What is the Earliest Deadline First Scheduling (EDF)?

- is an optimal dynamic priority scheduling algorithm used in real-time systems.
  It can be used for both static and dynamic real-time scheduling.
- EDF uses priorities to the jobs for scheduling. It assigns priorities to the task according to the absolute deadline. The task whose deadline is closest gets the highest priority. The priorities are assigned and changed in a dynamic fashion. EDF is very efficient as compared to other scheduling algorithms in real-time systems. It can make the CPU utilization to about 100% while still guaranteeing the deadlines of all the tasks.

Earliest Deadline First Scheduling (EDF)

02 Algorithm Include

EDF includes the kernel overload. In EDF, if the CPU usage is less than 100%, then it means that all the tasks have met the deadline. EDF finds an optimal feasible schedule. The feasible schedule is one in which all the tasks in the system are executed within the deadline. If EDF is not able to find a feasible schedule for all the tasks in the real-time system, then it means that no other task scheduling algorithms in real-time systems can give a feasible schedule. All the tasks which are ready for execution should announce their deadline to EDF when the task becomes runnable.

Earliest Deadline First Scheduling (EDF)

03

Algorithm Implementation

Struct includes all the process data

```
struct processTemp{
    int executionTime, deadline, period;
    processTemp(int e, int d, int p) : executionTime(e), deadline(d), period(p){}
};

//includes all the processes of the system
vector<processTemp> process, updatedProcess;

int processCount; //Number of the processes to be scheduled
int lcm; //Least Common Multiple

//Calculates the Least Common Multiple of two numbers
int LCM(int a, int b){
    return a / __gcd(a,b) * b;
}

//check schedulability
bool check(){
    double sum = 0;
    for(int i = 0; i < processCount; ++i)
        sum += (double) process[i].executionTime / (double) process[i].period;

    //if sum <= 1, then the set of processes MAY be Schedulable .. Otherwise it cannot be scheduled
    return sum <= 1;
}
int getEarliestDeadLine(){
    int earliestDeadline = lcm, index = -1;
    /*
    variable index will store the index of the Process which has the highest Priority
    -> The shorter Time Period a Process has, The Higher Priority it will be assigned
    */
    for(int i = 0; i < processCount; ++i){
        if(updatedProcess[i].executionTime == 0) continue;

        if(updatedProcess[i].deadline < earliestDeadline)
            earliestDeadline = updatedProcess[i].deadline, index = i;
    }
    return index;
}
```

Check schedulability of the system

Find the process with the highest priority to be executed based on the earliest deadline

Earliest Deadline First Scheduling (EDF)

03

```
void schedule(){

    int execTime, deadLine, period;
    for(int i = 0; i < processCount; ++i){

        execTime = process[i].executionTime;
        deadLine = process[i].deadline;
        period = 0;

        processTemp process(execTime, deadLine, period);
        updatedProcess.push_back(process);
    }

  ofstream file;
  file.open("EDF_Result.txt");
  file << "\n\tSchedule:\n\n";

  for(int time = 0; time < lcm; ++time){


        //getting the index of the next process to be executed based on it's priority
        int nxtProcess = getEarliestDeadLine();

        //if we found a process to be executed
        if(~nxtProcess)
            file << "\tTime [" << time <<" , "<< time + 1 << "] : " << "Process " << nxtProcess + 1 <<"\n";

        //else there is no process ready to be executed
        else
            file << "\tTime [" << time <<" , "<< time + 1 << "] : " << "NO Process Is Running\n";


        //reduce the execution time of the running process by 1 after each time unit
        if(~nxtProcess)
        updatedProcess[nxtProcess].executionTime --;

        //get the next deadline distance
        for(int i = 0; i < processCount; ++i){

            //reduce the time period of each process by 1 after each time unit
            updatedProcess[i].period ++;


             if(updatedProcess[i].period == process[i].period){
                 updatedProcess[i].deadline = process[i].deadline;
                 updatedProcess[i].executionTime = process[i].executionTime;
                 updatedProcess[i].period = 0;
             }
             else {
                 //if deadline is not reached yet, decrease it by 1
                 if(updatedProcess[i].deadline != 0)
                     updatedProcess[i].deadline --;

                 /*
                 if the deadline is reached & the current process hasn't completed it's execution time yet,
                 This means that it has missed it's deadline and cannot be completed..
                 -->>> The system is not schedulable
                 */
                 else if(updatedProcess[i].executionTime != 0)
                     file << "\n\tThe process "<< i + 1 <<" cannot be completed\n";
             }
          }
  }
  file << "\n";
  file.close();
  return;
```

Find the process with

the highest priority to

be executed based on

the earliest deadline

Earliest Deadline First Scheduling (EDF)

03

Algorithm Implementation

```
processCount = atoi(argv[1]);

int execTime, deadLine, period;
for(int i = 2; i < 3 * processCount + 2; i += 3){

    execTime = atoi(argv[i]);
    deadLine = atoi(argv[i + 1]);
    period = atoi(argv[i + 2]);

    processTemp oneProcess(execTime, deadLine, period);
    process.push_back(oneProcess);
}

//check if the system can be scheduled or not
bool schedulable = check();

//if it cannot be scheduled, report this and terminate the program
if(!schedulable){
    ofstream file;
    file.open("EDF_Result.txt");
    file << "This System is not schedulable!\n";
    file.close();
    return 0;
}

//Compute the Least Common Multiple of the Time periods of the whole set of processes
lcm = process[0].period;
for(int i = 1; i < processCount; ++i)
    lcm = LCM(lcm, process[i].period);

//schedule the set of processes
schedule();
```

reads the processes' data

, checks for schedulability,

then schedule the set of

processes If possible

Earliest Deadline First Scheduling (EDF)

04 Example

Process	Execution Time	Deadline	Time period
P1	3	7	20
P2	2	8	10
P3	2	4	5

05 Video

GUI

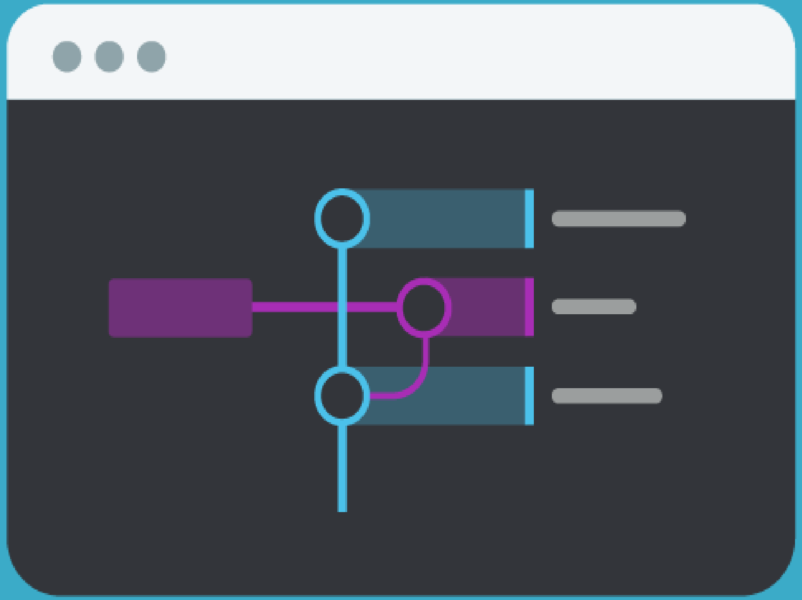
01

Platforms & Language

Visual Studio

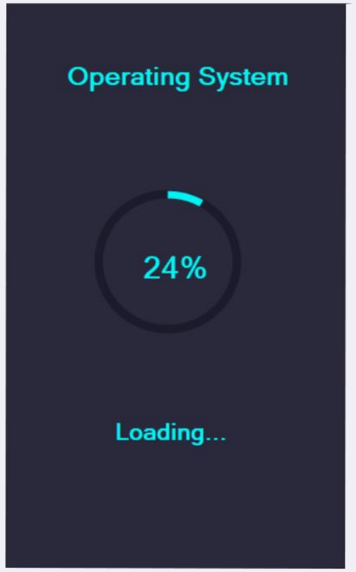
Used Language

Used Platforms



GUI

02 How to use : Screen 1,2

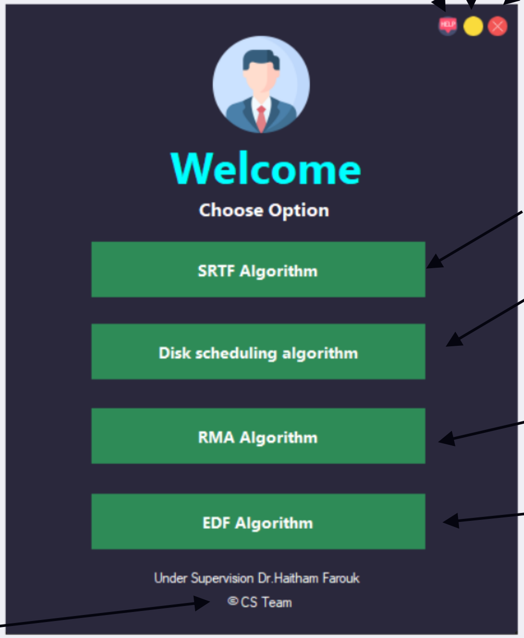


Loading Screen

Help button

Minimize button

Close button



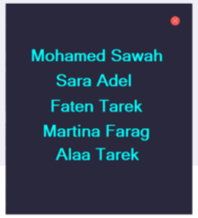
To open SRTF Algorithm

To open FCFS Algorithm

To open RMA Algorithm

To open EDF Algorithm

Copyrights button to show team names



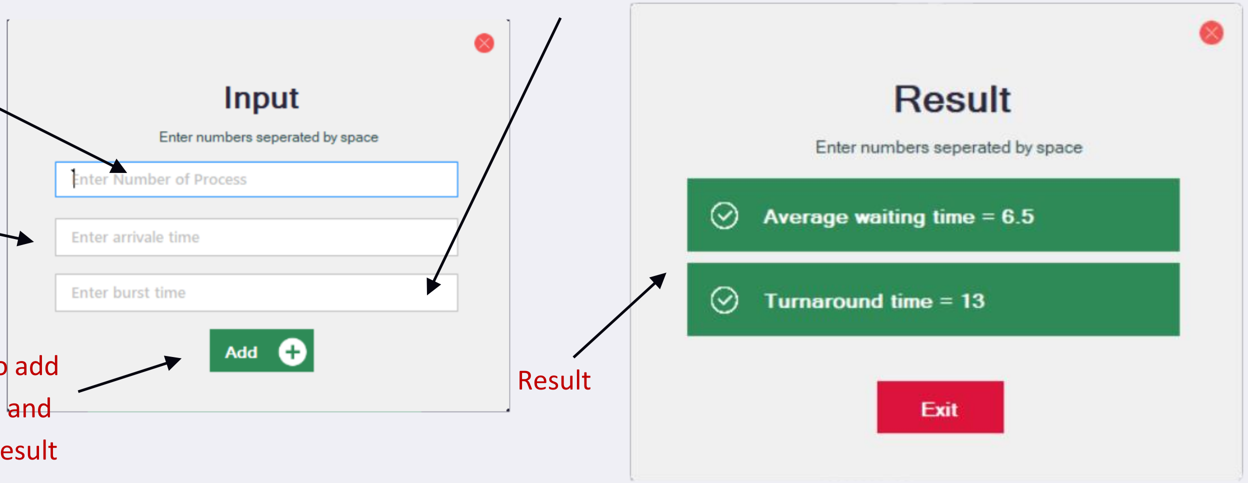
GUI

02 How to use : Screen 4,5

Here enter process numbers

Here enter arrival time

Here enter burst time



Here to add values and show result

Result

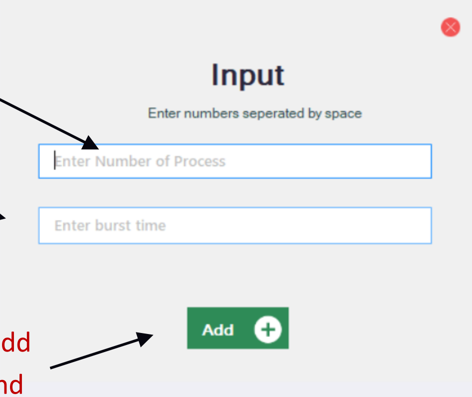
GUI

02 How to use : Screen 6,7

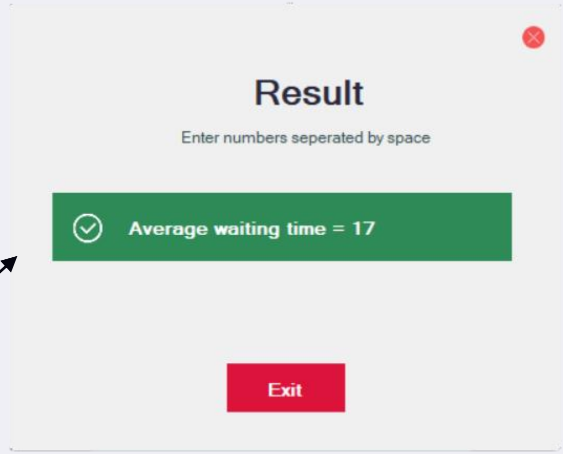
Here enter process numbers

Here enter burst time

Here to add values and show result



Result



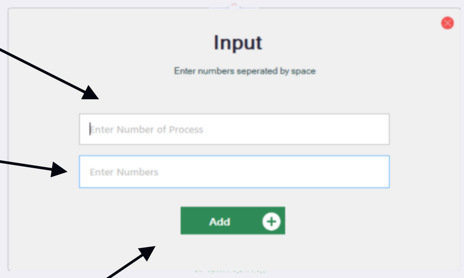
GUI

02 How to use : Screen 8,9

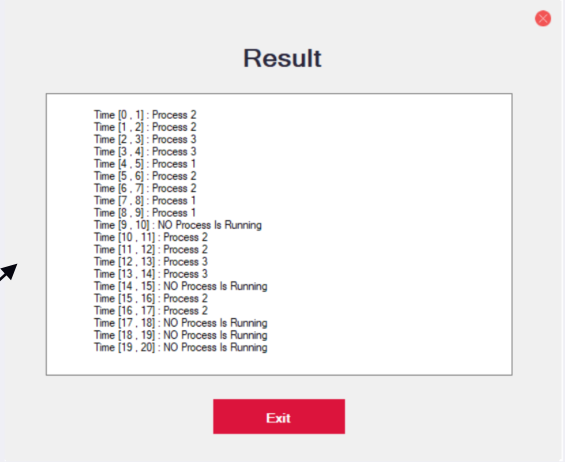
Here enter process numbers

Here enter numbers

Here to add values and show result



Result



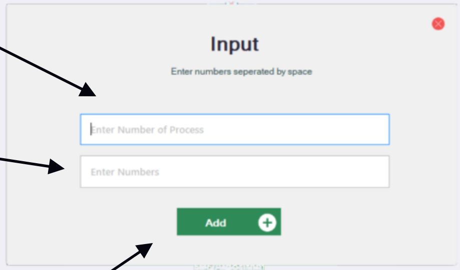
GUI

02

How to use : Screen 10,11

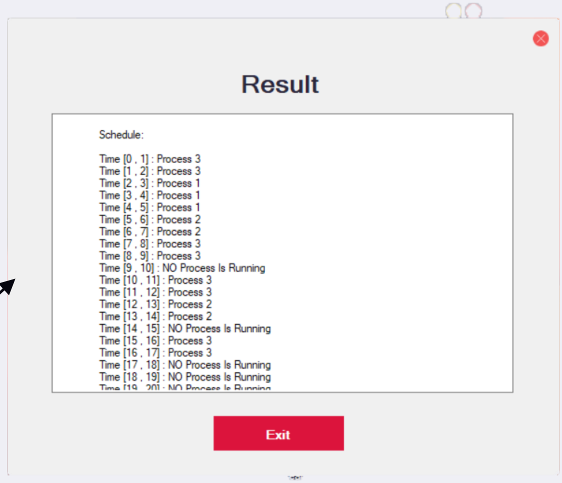
Here enter process numbers

Here enter numbers



Here to add values and show result

Result



Thank You