SIMULATION OF CPU SCHEDULING ALGORITHMS USING PYTHON TKINTER

A thesis submitted in partial fulfilment of the requirements for the award of the degree of

Minor Degree

in

COMPUTER SCIENCE AND ENGINEERING

by

YAMASANI BALAJI REDDY (179X1A02B0)
THARIGONDA MAHESH (179X1A03F9)
CHINTHAKAYALA VENKATA RAMAN (179X1A0346)

Under the Esteemed Guidance of

Dr. B. SANTHOSH KUMAR M.S., Ph.D

Associate Professor
Department of C.S.E.

Department of Computer Science and Engineering
G. Pulla Reddy Engineering College (Autonomous): Kurnool
(Affiliated to Jawaharlal Nehru Technological University-Anantapur, Ananthapuramu)

2020-2021

Department of Computer Science and Engineering
G. Pulla Reddy Engineering College (Autonomous): Kurnool
(Affiliated to Jawaharlal Nehru Technological University-Anantapur, Ananthapuramu)

Certificate

This is to certify that the project entitled **"Simulation of CPU scheduling algorithms using Python Tkinter"** is a bonafide work done by **Yamasani Balaji Reddy (179X1A02B0), Tharigonda Mahesh (179X1A03F6), Chinthakayala Venkata Raman (179X1A0346)** in partial fulfilment of the requirements for the award of degree of **Minor Degree** in **Computer Science and Engineering** during the academic year 2020-2021.

The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree.

Dr. B. Santhosh Kumar
Associate Professor,
C.S.E. Department,
G. Pulla Reddy Engineering College, Kurnool, A.P.

Dr. N .KasiViswanath
Professor & Head,
C.S.E. Department,
G. Pulla Reddy Engineering College, Kurnool, A.P.

DECLARATION

We here by declare that the project titled "**Simulation of CPU Scheduling Algorithms Using Python Tkinter**" is the authentic work carried out by us as students of **G. PULLA REDDY ENGINEERING COLLEGE (Autonomous), Kurnool**, during April – August 2021 and has not been submitted elsewhere for the award of degree in part or in full to any institute.

YAMASANI BALAJI REDDY (179X1A02B0)

THARIGONDA MAHESH (179X1A03F9)

CHINTHAKAYALA VENKATA RAMAN (179X1A0346)

ACKNOWLEDGEMENT

Before getting into the thickest of things, we would like to thank the people who were part of this project in numerous ways, those who gave us outstanding support for the implementation of the project.

We are grateful to our project in charge, **Dr. K.Govardhan Reddy,** Associate Professor of CSE Department, G. Pulla Reddy Engineering College, for helping us and giving us the required information needed for our project work.

We feel great pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected guide and project co-ordinator **Dr.B.Santhosh Kumar**, Associate Professor of Computer Science and Engineering Department, G. Pulla Reddy Engineering College, for his valuable guidance, encouragement and help for completing this work. His useful suggestions for this whole work and co-operative behaviour are sincerely acknowledged.

We wish to express our sense of gratitude to **Dr.N.Kasiviswanath Garu**, Professor & Head of the Computer Science and Engineering Department, G. Pulla Reddy Engineering College, for his support and encouragement during this project sessions.

We express our sincere thanks to our beloved principal **Dr.B.Sreenivasa Reddy** for providing the facilities extended to do our project.

We would like to express our sincere gratitude to the faculty and staff of Electrical and Electronics Engineering Department.

YAMASANI BALAJI REDDY (179X1A02B0)
THARIGONDA MAHESH (179X1A03F9)
CHINTHAKAYALA VENKATA RAMAN (179X1A0346)

ABSTRACT

CPU is the costliest and important resource in computer which need to be as busy as possible by scheduling the processes. CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to create multi-tasking or multi programming environment which improves the system efficiency.

In this project we are going to simulate process scheduling algorithms such as FCFS (First Come First Serve), SJF (Shortest Job First), Priority Scheduling and Round Robin algorithms with a user-friendly and mouse driven graphical user interface which is developed by using Python Tkinter framework, in order to know sequence of processes that are going to be executed and the time that each process waits for its execution.

Table of contents

List of Figures

INTRODUCTION

INTRODUCTION

Introduction:

An operating system is a program that manages the hardware and software resources of a computer. It is the first thing that is loaded into memory when we turn on the computer. Without the operating system, each programmer would have to create a way to send data to a printer, tell it how to read a disk file, and how to deal with other programs.

In the beginning programmers needed a way to handle complex input/output operations. The evolution of a computer programs and their complexities required new necessities. Because machines began to become more powerful, the time a program needed to run decreased. However, the time needed for handling off the equipment between different programs became evident and this led to program like DOS. As we can see the acronym DOS stands for Disk Operating System. This confirms that operating systems were originally made to handle these complex input/output operations like communicating among a variety of disk drives.

Earlier computers were not as powerful as they are today. In the early computer system you would only be able to run one program at a time. For instance, you could not be writing a paper and browsing the internet all at the same time. However, today's operating systems are very capable of handling not only two but multiple applications at the same time. In fact, if a computer is not able to do this it is considered useless by most computer users.

In order for a computer to be able to handle multiple applications simultaneously, there must be an effective way of using the CPU. Several processes may be running at the same time, so there has to be some kind of order to allow each process to get its share of CPU time.

An operating system must allocate computer resources among the potentially competing-requirements of multiple processes. In the case of the processor, the resource to be allocated is execution time on the processor and the means of allocation is scheduling. The scheduling function must be designed to satisfy a number of objectives, including fairness, lack of starvation of any particular process, efficient use of processor time, and low overhead. In addition, the scheduling function may need to take into account different levels of priority or real-time deadlines for the start or completion of certain process.

Over the years, scheduling has been the focus of intensive research, and many different algorithms have been implemented. Today, the emphasis in scheduling research is on exploiting multiprocessor systems, particularly for multithreaded applications, and real-time scheduling.

In a multi programming systems, multiple processes exist concurrently in main memory. Each process alternates between using a processor and waiting for some event to occur, such as the completion of an I/O operation. The processor is kept busy by executing one process while the others wait hence the key to multiprogramming is scheduling.

Motivation:

The CPU scheduling is one of the important problems in operating systems designing and build a program achieve these algorithms has been a challenge in this field and because of there is many scheduling algorithms so we choose some of them for enforcement one of Incentives for us to build this program.

Objective of the Project:

Simulation of the CPU scheduling algorithms using Python Tkinter existing in operating systems books with drawing a table with output parameters for algorithms. The system provides a clean and convenient way to test the given data and do the analysis of the CPU scheduling algorithms. For the purpose of analysis and testing the user first specifies each process along with its information such as arrival times and burst time and then algorithms can be computed producing output in appropriate format readability.

SYSTEM REQUIREMENTS

SYSTEMREQUIREMENTS

Software Requirements

• Framework	:	Tkinter is used to implement CPU scheduling algorithms
• Languages	:	Python
• Operating System	:	Windows 7 or above

Hardware Requirements

• RAM	:	4GB or above
• Hard disk	:	10GB or above
• Processor	:	Intel core i3 or above

BASIC CONCEPTS

BASIC CONCEPTS

Basic Concepts

In a single-processor system, only one process can run at a time; any others must wait until the CPU is free and can be rescheduled. The objective of multiprogramming is to have some processes running at all time, to maximize CPU utilization. The idea is relatively simple. A processor is executed until it must wait typically for the completion of some I/O request. In a simple computer system, the CPU then just sits idle. All this waiting time is wasted; no useful work is accomplished. With multiprogramming, we try use this time productively. Several processes are kept in memory at one time. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues. Every time one process has to wait, another process can take over use of the CPU Scheduling of this kind is a fundamental operating- system function. Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to operating-system design. So, in order to understand the process of scheduling the processes theses rudimentary concepts play a vital role to analyze and implement the process of theses algorithms.

CPU Scheduling

CPU scheduling is the basis of multi-programmed operating system. By switching the CPU among processes, the operating system can make the computer more productive. A multiprogramming operating system allows more than one processes to be loaded into the executable memory at a time and for the loaded processes to share the CPU using time-multiplexing. Part of the reason for using multiprogramming is that the operating system itself is implemented as one or more processes, so there must be a way for the operating system and application processes to share the CPU. Another main reason is the need for process to perform I/O operations in the normal course of computation. Since I/O operations ordinarily require orders of magnitude more time to complete than do CPU instructions, multiprogramming systems allocate the CPU to another process whenever a process invokes an I/O operation. Scheduling refers to the way processes are assigned to run on the available CPUs, since there are typically many more processes running than there are available CPUs.

CPU-I/O Burst Cycle

The success of CPU scheduling depends on an observed property of processes: process execution consists of a cycle of a CPU execution and I/O wait. Processes alternate between these two states.

A process begins with a CPU burst, followed by an I/O burst, followed by another CPU burst, then another I/O burst, and so on. Eventually, the last CPU burst ends with a system request to terminate the execution.

CPU Scheduler

Whenever, the CPU becomes idle, the operating system must select one of the processes in the ready-queue to be executed. The selection process is carried out the short-term scheduler or CPU scheduler. The CPU scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

The ready queue is not necessarily a First-In, First-Out (FIFO) queue. It can be implemented as a FIFO queue a priority queue. Conceptually, all the processes in the ready queue are lined up waiting for a chance to run on the CPU. The records in the queues are generally process control blocks (PCB) of the processes.

Preemptive Scheduling

CPU- scheduling decisions may take place under the following four circumstances:

1. when a process switches from the running state to the waiting state (e.g., as the result of an I/O request or an invocation of wait for the termination of one of the child processes)
2. when a process switches from the running state to the ready state (e.g., when an interrupt occurs)
3. when a process switches from the waiting state to the ready state (e.g., at completion of I/O)
4. When a process terminates for situations 1 and 4, there is no choice in terms of scheduling. A new process (if one existing in the ready queue) must be selected for execution. There is a choice, however, for situations 2 and 1.

5. When scheduling takes place only under circumstances 1 and 4, we say that the scheduling is non-preemptive or cooperative; otherwise, it is preemptive. Under non-preemptive scheduling once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state. Cooperative scheduling is the only method that can be used on certain hardware platforms, because it does not require the special hardware such as a timer needed for preemptive scheduling.

Dispatcher

Another component involved in the CPU- scheduling function is the dispatcher. The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler; this involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency.

SCHEDULING ALGORITHMS

SCHEDULING ALGORITHMS

Scheduling Algorithms

During the process of CPU Scheduling, scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them. CPU scheduling decisions may take place when a process:

- Switches from running to waiting state
- Switches from running to ready state
- Switches from waiting to ready
- Terminates.

CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. There are many different CPU scheduling algorithms; in this section we describe some of them.

They are:

- First Come First Serve
- Shortest Job First
- Priority Scheduling
- Round Robin.

First Come First–Serve Scheduling

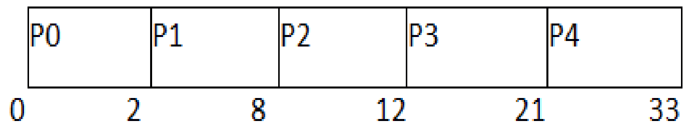
By far the simplest CPU-scheduling algorithm is the first-come, first-serve (FCFS) scheduling algorithm. With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue.

In the **First Come First Serve** scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

- First Come First Serve is just like **FIFO** (First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
- This is used in Batch Systems.
- It's **easy to understand and implement** programmatically, using a Queue data structure, where a new process enters through the **tail** of the queue, and the scheduler selects process from the **head** of the queue.
- A perfect real-life example of FCFS scheduling is **buying tickets at ticket counter**.

Consider the following set of processes with arrival time and length of the CPU burst given in milliseconds:

Process ID	**Arrival Time**	**Burst Time**	**Service Time**
0	0	2	0
1	1	6	2
2	2	4	8
3	3	9	12
4	4	12	33



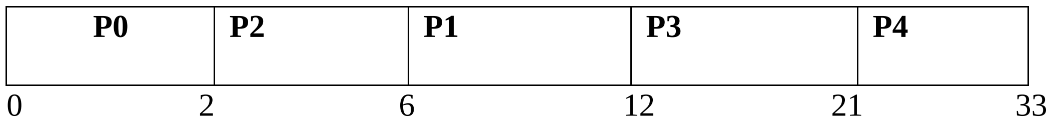
Shortest Job First

The SJF scheduling algorithm is provably optimal, in that it gives the minimum average waiting time for a given set of processes. Moving a short process before a long one decrease the waiting time of the short process more than it increases the waiting time of the long process. Consequently, the average waiting time decreases. The SJF algorithm can be either preemptive or non-preemptive. The choice arises when a new process arrives at the ready queue while a previous process is still executing. The next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process. A preemptive SJF algorithm will

Preempt the currently execute process, whereas a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst. Sometimes, non-preemptive SJF scheduling is also called Shortest Process Next (SPN) scheduling & preemptive SJF is called Shortest remaining time (SRT) scheduling.

As an example, consider the following four processes, with the length of the CPU burst given in milliseconds:

PID	Arrival Time	Burst Time	Completion Time
0	0	2	0
1	1	6	6
2	2	4	2
3	3	9	12
4	4	12	21



If the processes arrive at the ready queue at the times shown and need the identical burst times, then the resulting preemptive SJF scheduling is as depicted in the following Gantt chart:

Priority Scheduling

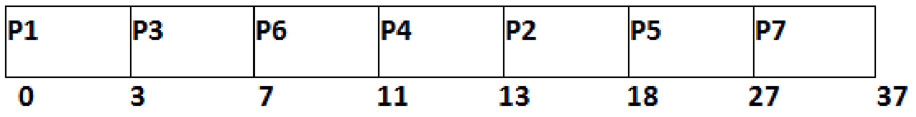
The SJF algorithm is a special case of the general priority scheduling algorithm. A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order. An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority and vice versa.

Note that we discuss scheduling in terms of high priority and low priority. Some systems use low numbers to represent low priority; others use low numbers for high priority. This difference can lead to confusion. In this text, we assume that high numbers represent high

Priority as an example, consider the following set of processes, assumed to have arrived at time 0, in the order P1, l2, ..., P5. With the length of the CPU burst given in milliseconds:

Using priority scheduling, we would schedule these processes according to the following Gantt chart:

Process ID	Priority	Arrival Time	Burst Time
1	2	0	3
2	6	2	5
3	3	1	4
4	5	4	2
5	7	6	9
6	4	5	4
7	10	7	10



Priorities can be defined either internally or externally. Internally defined priorities use some measurable quantity or quantities to compute the priority of a process. For example, time limits, memory requirements, the number of open files, and the ratio of average I/O burst to average CPU burst have been used in computing priorities. External priorities are set by criteria outside the operating system, such as the importance of the process, the type and amount of funds being paid for computer use, the department sponsoring the work, and other, often political, factors.

Priority scheduling can be either preemptive or non-preemptive. When a process arrives at the ready queue, its priority is compared with the priority of the currently running process. A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process. A non-preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.

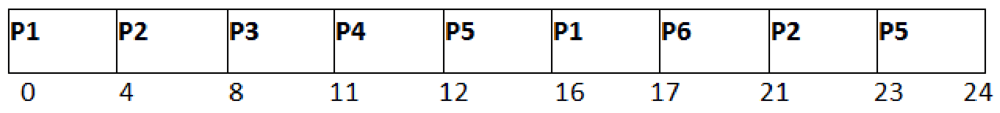
A major problem with priority scheduling algorithms is indefinite blocking, or starvation. A process that is ready to run but waiting for the CPU can be considered blocked. A priority scheduling algorithm can leave some low- priority processes waiting indefinitely. In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU. Generally, one of two things will happen. Either the process will eventually be run, or the computer system will eventually crash and lose all unfinished low-priority processes. A solution to the problem of indefinite blockage of low-priority processes is aging. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.

Round-Robin Scheduling

The round-robin (RR) scheduling algorithm is designed especially for timesharing systems. It is similar to FCFS scheduling, but preemption is added to switch between processes. A small unit of time, called a time quantum or time slice, is defined. A time quantum is generally from 10 to 100 milliseconds. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. To implement RR scheduling, we keep the ready queue as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process. One of two things will then happen. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

Consider the following set of processes that arrive time and the length of the CPU burst given in milliseconds:

Process ID	**Arrival Time**	**Burst Time**
1	0	5
2	1	6
3	2	3
4	3	1
5	4	5
6	6	4



OUTPUT SCREENS

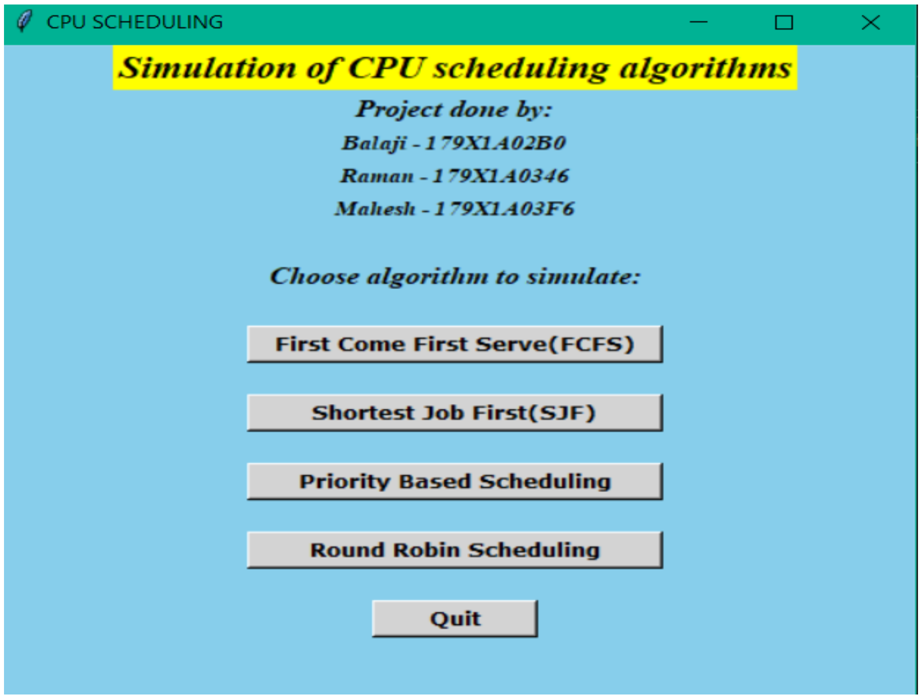
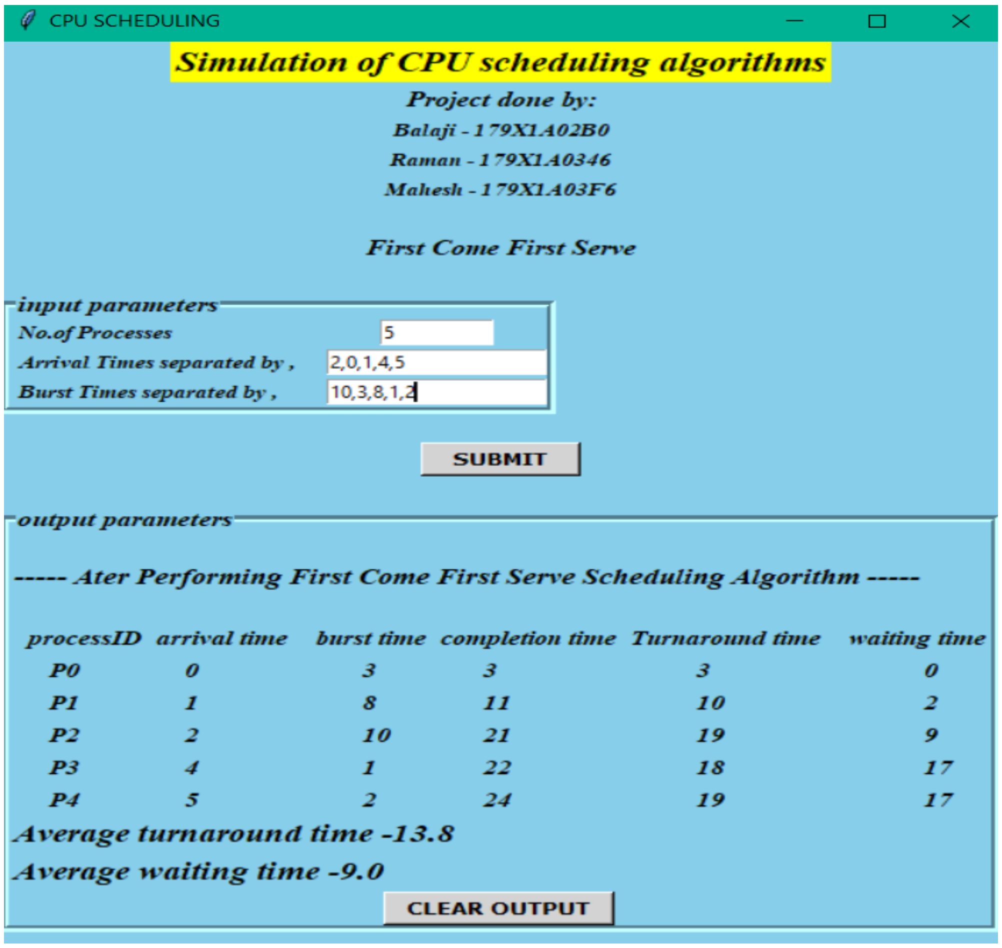


Figure 5.1 Menu Screen (FCFS, SJF, PRIORITY, RR)


CPU SCHEDULING

Simulation of CPU scheduling algorithms

Project done by:
Balaji - 179X1A02B0
Raman - 179X1A0346
Mahesh - 179X1A03F6

First Come First Serve

input parameters

No.of Processes	5
Arrival Times separated by ,	2,0,1,4,5
Burst Times separated by ,	10,3,8,1,2

SUBMIT

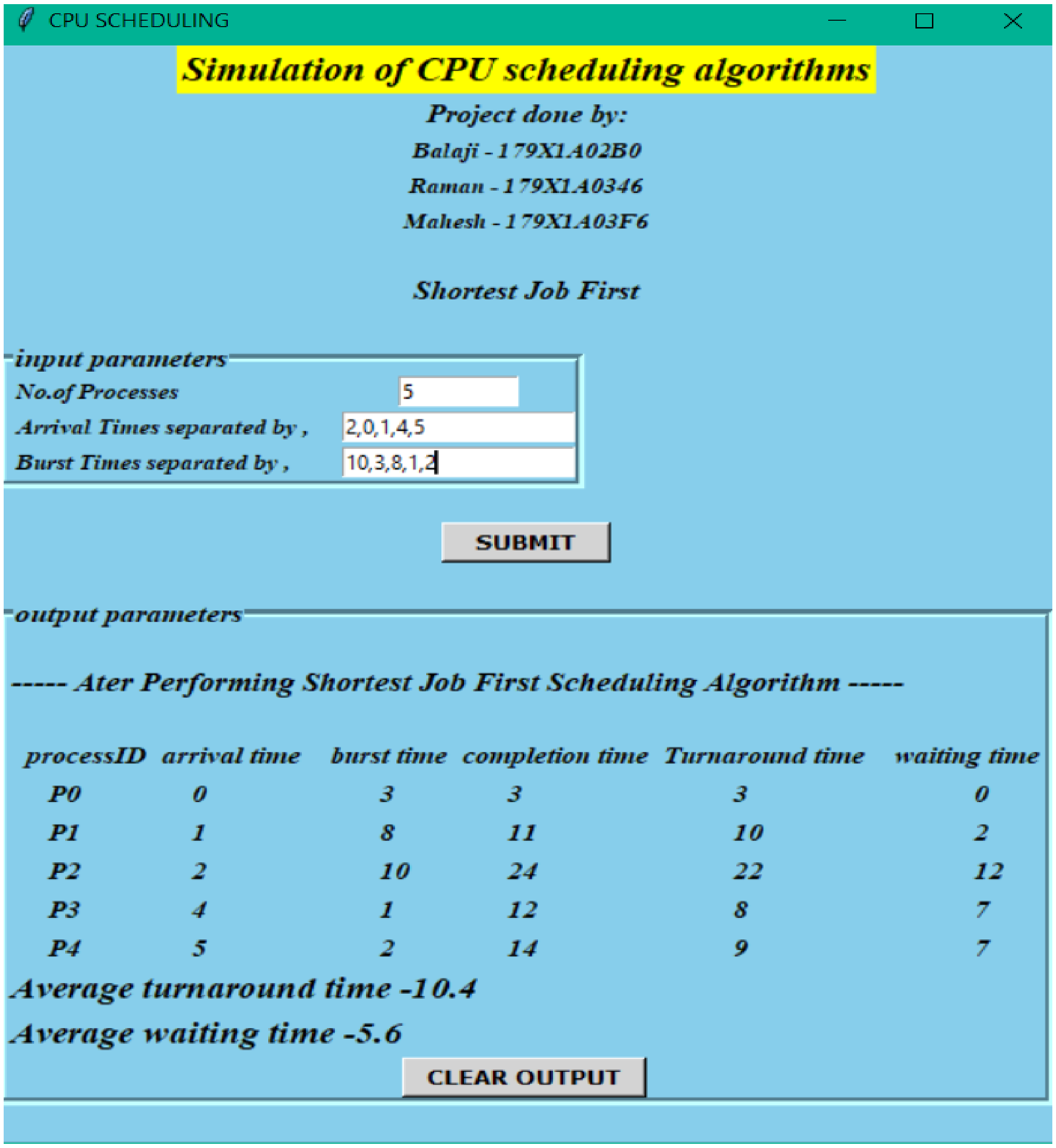
output parameters

```
----- Ater Performing First Come First Serve Scheduling Algorithm -----

 processID  arrival time   burst time  completion time  Turnaround time   waiting time
   P0          0              3          3                 3                 0
   P1          1              8          11                10                2
   P2          2              10         21                19                9
   P3          4              1          22                18                17
   P4          5              2          24                19                17
Average turnaround time -13.8
Average waiting time -9.0
```

CLEAR OUTPUT

Figure 5.2 implementation of FCFS (First Come First Serve) algorithm



Simulation of CPU scheduling algorithms

Project done by:

Balaji - 179X1A02B0

Raman - 179X1A0346

Mahesh - 179X1A03F6

Shortest Job First

input parameters

No.of Processes	5
Arrival Times separated by ,	2,0,1,4,5
Burst Times separated by ,	10,3,8,1,2

output parameters

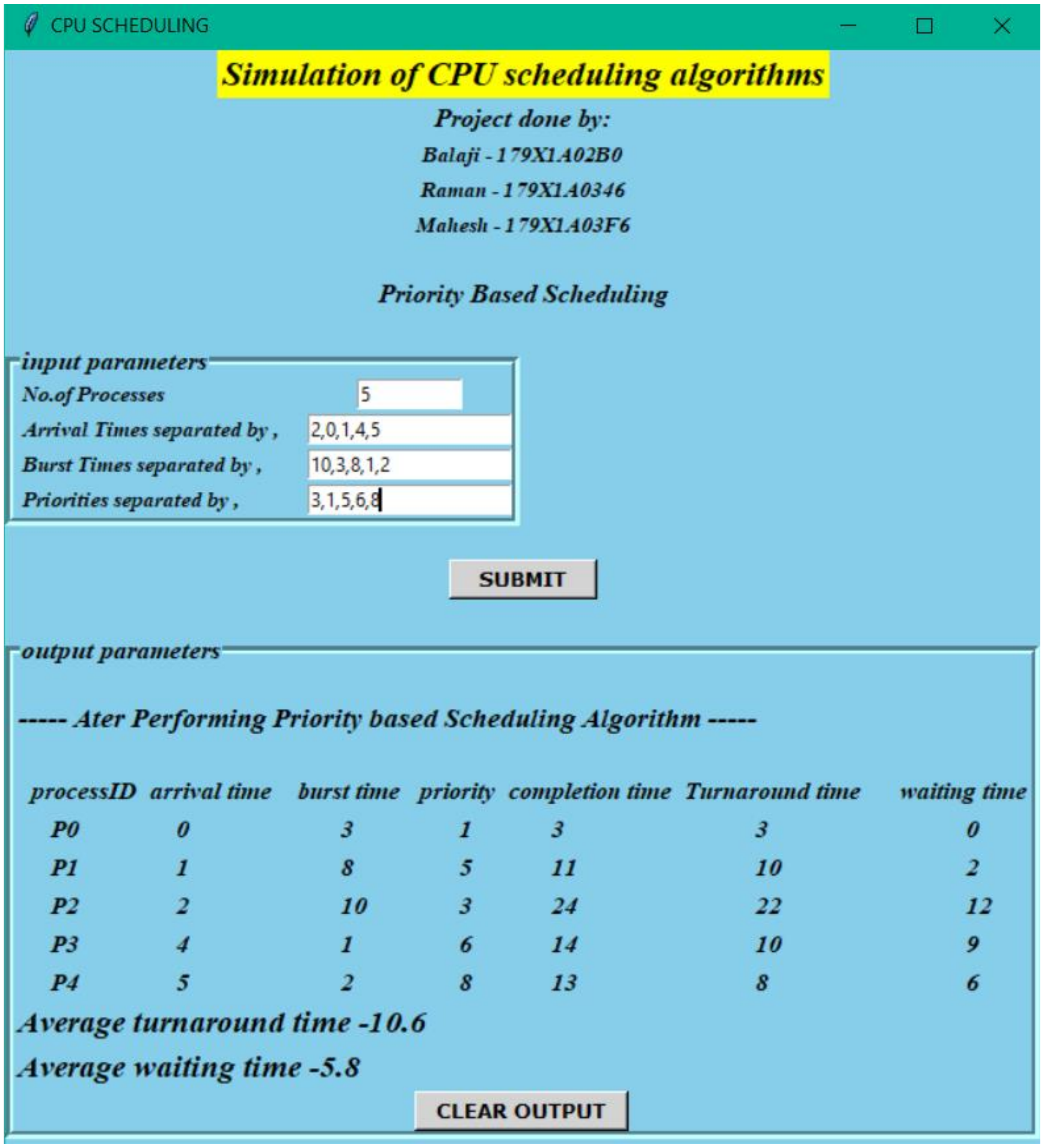
----- Ater Performing Shortest Job First Scheduling Algorithm -----

processID	arrival time	burst time	completion time	Turnaround time	waiting time
P0	0	3	3	3	0
P1	1	8	11	10	2
P2	2	10	24	22	12
P3	4	1	12	8	7
P4	5	2	14	9	7

Average turnaround time -10.4

Average waiting time -5.6

Figure 5.3 implementation of SJF (Shortest Job First) algorithm


CPU SCHEDULING

Simulation of CPU scheduling algorithms

Project done by:

Balaji - 179X1A02B0

Raman - 179X1A0346

Mahesh - 179X1A03F6

Priority Based Scheduling

input parameters

No.of Processes	5
Arrival Times separated by ,	2,0,1,4,5
Burst Times separated by ,	10,3,8,1,2
Priorities separated by ,	3,1,5,6,8

SUBMIT

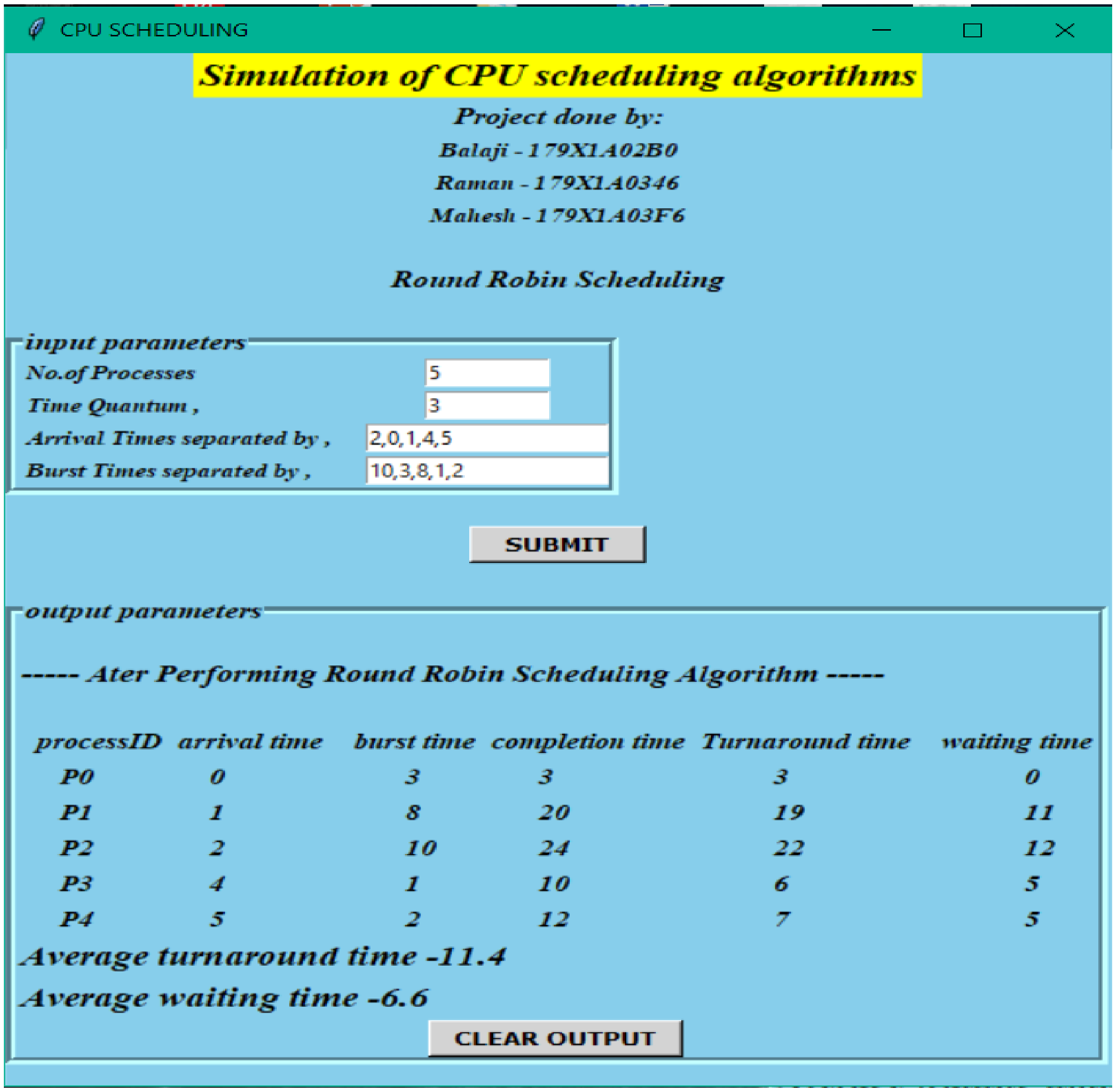
output parameters

----- Ater Performing Priority based Scheduling Algorithm -----

```
processID  arrival time  burst time  priority  completion time  Turnaround time  waiting time
   P0           0             3          1           3                3               0
   P1           1             8          5          11               10               2
   P2           2            10          3          24               22              12
   P3           4             1          6          14               10               9
   P4           5             2          8          13                8               6
Average turnaround time -10.6
Average waiting time -5.8
```

CLEAR OUTPUT

Figure 5.4 implementation of Priority Scheduling algorithm



Simulation of CPU scheduling algorithms

Project done by:

Balaji - 179X1A02B0

Raman - 179X1A0346

Mahesh - 179X1A03F6

Round Robin Scheduling

input parameters

No.of Processes	5
Time Quantum ,	3
Arrival Times separated by ,	2,0,1,4,5
Burst Times separated by ,	10,3,8,1,2

output parameters

----- Ater Performing Round Robin Scheduling Algorithm -----

```
processID  arrival time  burst time  completion time  Turnaround time  waiting time
   P0           0            3             3                3                0
   P1           1            8            20               19               11
   P2           2           10            24               22               12
   P3           4            1            10                6                5
   P4           5            2            12                7                5
Average turnaround time -11.4
Average waiting time -6.6
```

Figure 5.5 implementation of RR (Round Robin) algorithm

IMPLEMENTATION

IMPLIMENTATION

Source code

The code that implement algorithms

```
import tkinter as tk
import sys

def cpuschedu():
    win=tk.Tk()
    win.title('CPU SCHEDULING')
    win.geometry('500x500')
    win.configure(bg='skyblue')
    tit=tk.Label(win,text='Simulation of CPU scheduling algorithms',fg='bla
ck',bg='yellow',font='Times 16 bold italic')
    tit.pack()
    tk.Label(win,text = 'Project done by:',fg='black',bg='skyblue',font='Ti
mes 12 bold italic').pack()
    tk.Label(win,text = 'Balaji - 179X1A02B0',fg='black',bg='skyblue',font=
'Times 10 bold italic').pack()
    tk.Label(win,text = 'Raman - 179X1A0346',fg='black',bg='skyblue',font='
Times 10 bold italic').pack()
    tk.Label(win,text = 'Mahesh - 179X1A03F6',fg='black',bg='skyblue',font=
'Times 10 bold italic').pack()
    tk.Label(win,bg='skyblue').pack()
    tk.Label(win,text = 'Choose algorithm to simulate:',fg='black',bg='skyb
lue',font='Times 12 bold italic').pack()
    tk.Label(win,bg='skyblue').pack()

    def FCFS():

        def sub():
            def clear():
                frame_1.destroy()

            arl=list((entry2.get()).split(','))
            arl=[int(i) for i in arl]
            brl=list((entry3.get()).split(','))
            brl=[int(i) for i in brl]
            n=len(arl)

            for i in range(n):
                for j in range(i+1,n):
                    if(arl[i] > arl[j]):
                        arl[i],arl[j]=arl[j],arl[i]
                        brl[i],brl[j]=brl[j],brl[i]
            p=['P'+str(i) for i in range(n)]
            CompletionTime = list()
            TatTime = list()
            CompletionTime.append(arl[0]+brl[0])
            for i in range(1,n):
                CompletionTime.append(brl[i]+CompletionTime[i-1])
            for i in range(0,n):
                TatTime.append(CompletionTime[i]-arl[i])
            waitTime = [TatTime[i]-brl[i] for i in range(n)]

            turn=sum(TatTime)/float(n)
            avg = sum(waitTime)/float(n)



            frame_1=tk.LabelFrame(fcfs,text='output parameters',fg='black',
bg='skyblue',font='Times 12 bold italic',bd=5,width=250,height=100)
            frame_1.pack(anchor=tk.W)


            tk.Label(frame_1,text = "\n----
- Ater Performing First Come First Serve Scheduling Algorithm -----
\n",fg='black',bg='skyblue',font='Times 13 bold italic').pack(anchor = tk.W
)

            tk.Label(frame_1,text='  processID  arrival time    burst time
 completion time  Turnaround time    waiting time',fg='black',bg='skyblue',
font='Times 11 bold italic').pack(anchor=tk.W)
            for i in range(n):
                tk.Label(frame_1,text='     '+str(p[i])+'\t        '+str(ar
l[i])+'\t\t '+str(brl[i])+'\t  '+str(CompletionTime[i])+'\t\t'+str(TatTime[
i])+'\t\t'+str(waitTime[i]),fg='black',bg='skyblue',font='Times 11 bold ita
lic').pack(anchor=tk.W)

            tk.Label(frame_1,text='Average turnaround time -
'+str(round(turn,4)),fg='black',bg='skyblue',font='Times 14 bold italic').p
ack(anchor=tk.W)
            tk.Label(frame_1,text='Average waiting time -
'+str(round(avg,4)),fg='black',bg='skyblue',font='Times 14 bold italic').pa
ck(anchor=tk.W)

            tk.Button(frame_1,text='CLEAR OUTPUT',command=clear,width='15',
font='Verdana 9 bold',fg='black',bg='light gray',activebackground='green',a
ctiveforeground='yellow').pack()
```

```
        fcfs=tk.Tk()
        fcfs.title('CPU SCHEDULING')
        fcfs.geometry('565x650')
        fcfs.configure(bg='skyblue')
        tit=tk.Label(fcfs,text='Simulation of CPU scheduling algorithms',fg
='black',bg='yellow',font='Times 16 bold italic')
        tit.pack()
        tk.Label(fcfs,text = 'Project done by:',fg='black',bg='skyblue',fon
t='Times 12 bold italic').pack()
        tk.Label(fcfs,text = 'Balaji - 179X1A02B0',fg='black',bg='skyblue',
font='Times 10 bold italic').pack()
        tk.Label(fcfs,text = 'Raman - 179X1A0346',fg='black',bg='skyblue',f
ont='Times 10 bold italic').pack()
        tk.Label(fcfs,text = 'Mahesh - 179X1A03F6',fg='black',bg='skyblue',
font='Times 10 bold italic').pack()

        tk.Label(fcfs,text = '\nFirst Come First Serve\n',fg='black',bg='sk
yblue',font='Times 12 bold italic').pack()

        frame=tk.LabelFrame(fcfs,text='input parameters',fg='black',bg='sky
blue',font='Times 12 bold italic',bd=5,width=250,height=100)
        frame.pack(anchor=tk.W,fill=tk.Y)

        tk.Label(frame,text=' No.of Processes ',fg='black',bg='skyblue',fon
t='Times 10 bold italic').grid(row=0,column=0,sticky=tk.W)
        tk.Label(frame,text=' Arrival Times separated by ,      ',fg='black'
,bg='skyblue',font='Times 10 bold italic').grid(row=1,column=0,sticky=tk.W)
        tk.Label(frame,text=' Burst Times separated by ,       ',fg='black',
bg='skyblue',font='Times 10 bold italic').grid(row=2,column=0,sticky=tk.W)


        entry1=tk.Entry(frame,width=10)
        entry1.insert(0,'3')
        entry1.grid(row=0,column=1)
        entry2=tk.Entry(frame,width=20)
        entry2.insert(0,'0,1,2')
        entry2.grid(row=1,column=1)
        entry3=tk.Entry(frame,width=20)
        entry3.insert(0,'10,2,5')
        entry3.grid(row=2,column=1)

        tk.Label(fcfs,bg='skyblue').pack()
        tk.Button(fcfs,text='SUBMIT',command=sub,width='10',font='Verdana 9
 bold',fg='black',bg='light gray',activebackground='green',activeforeground
='yellow').pack()
        tk.Label(fcfs,bg='skyblue').pack()


    def SJF():
        def sub():
            def clear():
                frame_1.destroy()

            arl=list((entry2.get()).split(','))
            arl=[int(i) for i in arl]
            brl=list((entry3.get()).split(','))
            brl=[int(i) for i in brl]
            n=len(arl)
            for i in range(n):
                for j in range(i+1,n):
                    if(arl[i] > arl[j]):
                        arl[i],arl[j]=arl[j],arl[i]
                        brl[i],brl[j]=brl[j],brl[i]
            p=['P'+str(i) for i in range(n)]

            pro_data=[]
            for i in range(n):
                te=[]
                te.extend([p[i],arl[i],brl[i],0])
                pro_data.append(te)

            start_time=[]
            exit_time=[]
            s_time=0
            pro_data.sort(key=lambda x:x[1])

            for i in range(n):
                ready_que=[]
                temp=[]
                normal_que=[]
                for j in range(n):
                    if (pro_data[j][1] <= s_time) and (pro_data[j][3] == 0)
:
                        temp.extend([pro_data[j][0], pro_data[j][1]
, pro_data[j][2]])
                        ready_que.append(temp)
                        temp = []
                    elif pro_data[j][3] == 0:
                        temp.extend([pro_data[j][0], pro_data[j][1], pro_da
ta[j][2]])
                        normal_que.append(temp)
                        temp = []

                if(len(ready_que) != 0):
                    ready_que.sort(key=lambda x: x[2])

                    start_time.append(s_time)
                    s_time = s_time + ready_que[0][2]
                    e_time = s_time
                    exit_time.append(e_time)
                    for k in range(len(pro_data)):
                        if pro_data[k][0] == ready_que[0][0]:
                            break
                    pro_data[k][3] = 1
                    pro_data[k].append(e_time)

                elif(len(ready_que) == 0):
                    if s_time < normal_que[0][1]:
                        s_time = normal_que[0][1]
                    start_time.append(s_time)
                    s_time = s_time + normal_que[0][2]
                    e_time = s_time
                    exit_time.append(e_time)
                    for k in range(len(pro_data)):
                        if pro_data[k][0] == normal_que[0][0]:
                            break
                    pro_data[k][3] = 1
                    pro_data[k].append(e_time)



            total_turnaround_time=0
            for i in range(n):
                turnaround_time = pro_data[i][4] - pro_data[i][1]
                total_turnaround_time = total_turnaround_time + turnaround_
time
                pro_data[i].append(turnaround_time)
            turn=total_turnaround_time/float(n)
            total_waiting_time = 0
            for i in range(len(pro_data)):
                waiting_time = pro_data[i][5] - pro_data[i][2]
                total_waiting_time = total_waiting_time + waiting_time
                pro_data[i].append(waiting_time)
            avg = total_waiting_time / float(n)

            pro_data.sort(key=lambda x: x[0])



            frame_1=tk.LabelFrame(sjf,text='output parameters',fg='black',b
g='skyblue',font='Times 12 bold italic',bd=5,width=250,height=100)
            frame_1.pack(anchor=tk.W)


            tk.Label(frame_1,text = "\n----
- Ater Performing Shortest Job First Scheduling Algorithm -----
\n",fg='black',bg='skyblue',font='Times 13 bold italic').pack(anchor = tk.W
)

            tk.Label(frame_1,text='  processID  arrival time    burst time
 completion time  Turnaround time    waiting time',fg='black',bg='skyblue',
font='Times 11 bold italic').pack(anchor=tk.W)
            for i in range(n):

                tk.Label(frame_1,text='     '+str(pro_data[i][0])+'\t
  '+str(pro_data[i][1])+'\t\t '+str(pro_data[i][2])+'\t  '+str(pro_data[i][
4])+'\t\t'+str(pro_data[i][5])+'\t\t'+str(pro_data[i][6]),fg='black',bg='sk
yblue',font='Times 11 bold italic').pack(anchor=tk.W)

            tk.Label(frame_1,text='Average turnaround time -
'+str(round(turn,4)),fg='black',bg='skyblue',font='Times 14 bold italic').p
ack(anchor=tk.W)
            tk.Label(frame_1,text='Average waiting time -
'+str(round(avg,4)),fg='black',bg='skyblue',font='Times 14 bold italic').pa
ck(anchor=tk.W)

            tk.Button(frame_1,text='CLEAR OUTPUT',command=clear,width='15',
font='Verdana 9 bold',fg='black',bg='light gray',activebackground='green',a
ctiveforeground='yellow').pack()

        sjf=tk.Tk()
        sjf.title('CPU SCHEDULING')
        sjf.geometry('565x670')
        sjf.configure(bg='skyblue')
        tit=tk.Label(sjf,text='Simulation of CPU scheduling algorithms',fg=
'black',bg='yellow',font='Times 16 bold italic')
        tit.pack()
        tk.Label(sjf,text = 'Project done by:',fg='black',bg='skyblue',font
='Times 12 bold italic').pack()
        tk.Label(sjf,text = 'Balaji - 179X1A02B0',fg='black',bg='skyblue',f
ont='Times 10 bold italic').pack()
        tk.Label(sjf,text = 'Raman - 179X1A0346',fg='black',bg='skyblue',fo
nt='Times 10 bold italic').pack()
        tk.Label(sjf,text = 'Mahesh - 179X1A03F6',fg='black',bg='skyblue',f
ont='Times 10 bold italic').pack()

        tk.Label(sjf,text = '\nShortest Job First\n',fg='black',bg='skyblue
',font='Times 12 bold italic').pack()

        frame=tk.LabelFrame(sjf,text='input parameters',fg='black',bg='skyb
lue',font='Times 12 bold italic',bd=5,width=250,height=100)
        frame.pack(anchor=tk.W,fill=tk.Y)

        tk.Label(frame,text=' No.of Processes ',fg='black',bg='skyblue',fon
t='Times 10 bold italic').grid(row=0,column=0,sticky=tk.W)
        tk.Label(frame,text=' Arrival Times separated by ,      ',fg='black'
,bg='skyblue',font='Times 10 bold italic').grid(row=1,column=0,sticky=tk.W)
        tk.Label(frame,text=' Burst Times separated by ,       ',fg='black',
bg='skyblue',font='Times 10 bold italic').grid(row=2,column=0,sticky=tk.W)


        entry1=tk.Entry(frame,width=10)
        entry1.insert(0,'3')
        entry1.grid(row=0,column=1)
        entry2=tk.Entry(frame,width=20)
        entry2.insert(0,'0,1,2')
        entry2.grid(row=1,column=1)
        entry3=tk.Entry(frame,width=20)
        entry3.insert(0,'10,2,5')
        entry3.grid(row=2,column=1)

        tk.Label(sjf,bg='skyblue').pack()

        tk.Button(sjf,text='SUBMIT',command=sub,width='10',font='Verdana 9
bold',fg='black',bg='light gray',activebackground='green',activeforeground=
'yellow').pack()
        tk.Label(sjf,bg='skyblue').pack()
    def PB():
        def sub():
            def clear():
                frame_1.destroy()
            arl=list((entry2.get()).split(','))
            arl=[int(i) for i in arl]
            brl=list((entry3.get()).split(','))
            brl=[int(i) for i in brl]
            pri=list((entry4.get()).split(','))
            pri=[int(i) for i in pri]
            n=len(arl)

            for i in range(n):
                for j in range(i+1,n):
                    if(arl[i]>arl[j]):
                        arl[i],arl[j]=arl[j],arl[i]
                        brl[i],brl[j]=brl[j],brl[i]
                        pri[i],pri[j]=pri[j],pri[i]

            p=['P'+str(i) for i in range(n)]

            pro_data=[]
            for i in range(n):
                te=[]
                te.extend([p[i],arl[i],brl[i],pri[i],0])
                pro_data.append(te)

            start_time = []
            exit_time = []
            s_time = 0
            pro_data.sort(key=lambda x: x[1])

            for i in range(len(pro_data)):
                ready_queue = []
                temp = []
                normal_queue = []
                for j in range(len(pro_data)):
                    if (pro_data[j][1] <= s_time) and (pro_data[j][4] == 0)
:
                        temp.extend([pro_data[j][0], pro_data[j][1], pro_da
ta[j][2], pro_data[j][3]])
                        ready_queue.append(temp)
                        temp = []
                    elif pro_data[j][4] == 0:
                        temp.extend([pro_data[j][0], pro_data[j][1], pro_da
ta[j][2], pro_data[j][3]])
                        normal_queue.append(temp)
                        temp = []
                if len(ready_queue) != 0:
                    ready_queue.sort(key=lambda x: x[3], reverse=True)
                    start_time.append(s_time)
                    s_time = s_time + ready_queue[0][2]
                    e_time = s_time
                    exit_time.append(e_time)
                    for k in range(len(pro_data)):
                        if pro_data[k][0] == ready_queue[0][0]:
                            break
                    pro_data[k][4] = 1
                    pro_data[k].append(e_time)
                elif len(ready_queue) == 0:
                    if s_time < normal_queue[0][1]:
                        s_time = normal_queue[0][1]
                    start_time.append(s_time)
                    s_time = s_time + normal_queue[0][2]
                    e_time = s_time
                    exit_time.append(e_time)
                    for k in range(len(pro_data)):
                        if pro_data[k][0] == normal_queue[0][0]:
                            break
                    pro_data[k][4] = 1
                    pro_data[k].append(e_time)

            total_turnaround_time=0
            for i in range(n):
                turnaround_time = pro_data[i][5] - pro_data[i][1]
                total_turnaround_time = total_turnaround_time + turnaround_
time

                pro_data[i].append(turnaround_time)
            turn=total_turnaround_time/float(n)

            total_waiting_time = 0
            for i in range(len(pro_data)):
                waiting_time = pro_data[i][6] - pro_data[i][2]
                total_waiting_time = total_waiting_time + waiting_time
                pro_data[i].append(waiting_time)
            avg = total_waiting_time / float(n)

            pro_data.sort(key=lambda x: x[0])



            frame_1=tk.LabelFrame(pb,text='output parameters',fg='black',bg
='skyblue',font='Times 12 bold italic',bd=5,width=250,height=100)
            frame_1.pack(anchor=tk.W)
            tk.Label(frame_1,text = "\n----
- Ater Performing Priority based Scheduling Algorithm -----
\n",fg='black',bg='skyblue',font='Times 13 bold italic').pack(anchor = tk.W
)

            tk.Label(frame_1,text='  processID  arrival time    burst time
  priority  completion time  Turnaround time      waiting time',fg='black',
bg='skyblue',font='Times 11 bold italic').pack(anchor=tk.W)

            for i in range(n):
                tk.Label(frame_1,text='     '+str(pro_data[i][0])+'\t
  '+str(pro_data[i][1])+'\t\t '+str(pro_data[i][2])+'\t   '+str(pro_data[i]
[3])+'\t '+str(pro_data[i][5])+'\t\t'+str(pro_data[i][6])+'\t\t'+str(pro_da
ta[i][7]),fg='black',bg='skyblue',font='Times 11 bold italic').pack(anchor=
tk.W)

            tk.Label(frame_1,text='Average turnaround time -
'+str(round(turn,4)),fg='black',bg='skyblue',font='Times 14 bold italic').p
ack(anchor=tk.W)
            tk.Label(frame_1,text='Average waiting time -
'+str(round(avg,4)),fg='black',bg='skyblue',font='Times 14 bold italic').pa
ck(anchor=tk.W)

            tk.Button(frame_1,text='CLEAR OUTPUT',command=clear,width='15',
font='Verdana 9 bold',fg='black',bg='light gray',activebackground='green',a
ctiveforeground='yellow').pack()

        pb=tk.Tk()
        pb.title('CPU SCHEDULING')
        pb.geometry('630x670')
        pb.configure(bg='skyblue')
        tit=tk.Label(pb,text='Simulation of CPU scheduling algorithms',fg='
black',bg='yellow',font='Times 16 bold italic')
        tit.pack()
        tk.Label(pb,text = 'Project done by:',fg='black',bg='skyblue',font=
'Times 12 bold italic').pack()
        tk.Label(pb,text = 'Balaji - 179X1A02B0',fg='black',bg='skyblue',fo
nt='Times 10 bold italic').pack()
        tk.Label(pb,text = 'Raman - 179X1A0346',fg='black',bg='skyblue',fon
t='Times 10 bold italic').pack()
        tk.Label(pb,text = 'Mahesh - 179X1A03F6',fg='black',bg='skyblue',fo
nt='Times 10 bold italic').pack()
```

```
        tk.Label(pb,text = '\nPriority Based Scheduling\n',fg='black',bg='s
kyblue',font='Times 12 bold italic').pack()

        frame=tk.LabelFrame(pb,text='input parameters',fg='black',bg='skybl
ue',font='Times 12 bold italic',bd=5,width=250,height=100)
        frame.pack(anchor=tk.W,fill=tk.Y)

        tk.Label(frame,text=' No.of Processes ',fg='black',bg='skyblue',fon
t='Times 10 bold italic').grid(row=0,column=0,sticky=tk.W)
        tk.Label(frame,text=' Arrival Times separated by ,      ',fg='black'
,bg='skyblue',font='Times 10 bold italic').grid(row=1,column=0,sticky=tk.W)
        tk.Label(frame,text=' Burst Times separated by ,       ',fg='black',
bg='skyblue',font='Times 10 bold italic').grid(row=2,column=0,sticky=tk.W)
        tk.Label(frame,text=' Priorities separated by , ',fg='black',bg='sk
yblue',font='Times 10 bold italic').grid(row=3,column=0,sticky=tk.W)


        entry1=tk.Entry(frame,width=10)
        entry1.insert(0,'3')
        entry1.grid(row=0,column=1)
        entry2=tk.Entry(frame,width=20)
        entry2.insert(0,'0,1,2')
        entry2.grid(row=1,column=1)
        entry3=tk.Entry(frame,width=20)
        entry3.insert(0,'10,2,5')
        entry3.grid(row=2,column=1)
        entry4=tk.Entry(frame,width=20)
        entry4.insert(0,'1,2,3')
        entry4.grid(row=3,column=1)

        tk.Label(pb,bg='skyblue').pack()

        tk.Button(pb,text='SUBMIT',command=sub,width='10',font='Verdana 9 b
old',fg='black',bg='light gray',activebackground='green',activeforeground='
yellow').pack()
        tk.Label(pb,bg='skyblue').pack()
    def RR():
        def sub():
            def clear():
                frame_1.destroy()
            t=int(entry4.get())
            arl=list((entry2.get()).split(','))
            arl=[int(i) for i in arl]
            brl=list((entry3.get()).split(','))
            brl=[int(i) for i in brl]
            n=len(arl)
            for i in range(n):
                for j in range(i+1,n):
                    if(arl[i] > arl[j]):
                        arl[i],arl[j]=arl[j],arl[i]
                        brl[i],brl[j]=brl[j],brl[i]

            p=['P'+str(i) for i in range(n)]

            pro_data=[]
            for i in range(n):
                te=[]
                te.extend([p[i],arl[i],brl[i],0,brl[i]])
                pro_data.append(te)

            start_time=[]
            exit_time=[]
            executed_pro=[]
            ready_que=[]
            s_time=0
            pro_data.sort(key=lambda x:x[1])

            time_slice=t

            while 1:
                normal_que = []
                temp = []
                for i in range(len(pro_data)):
                    if pro_data[i][1] <= s_time and pro_data[i][3] == 0:
                        present = 0
                        if len(ready_que) != 0:
                            for k in range(len(ready_que)):
                                if pro_data[i][0] == ready_que[k][0]:
                                    present = 1

                        if present == 0:
                            temp.extend([pro_data[i][0], pro_data[i][1], pr
o_data[i][2], pro_data[i][4]])
                            ready_que.append(temp)
                            temp = []

                        if len(ready_que) != 0 and len(executed_pro) != 0:
                            for k in range(len(ready_que)):
                                if ready_que[k][0] == executed_pro[len(exec
uted_pro) - 1]:
```

```
                                    ready_que.insert((len(ready_que) - 1),
ready_que.pop(k))

                    elif pro_data[i][3] == 0:
                        temp.extend([pro_data[i][0], pro_data[i][1], pro_da
ta[i][2], pro_data[i][4]])
                        normal_que.append(temp)
                        temp = []
                if len(ready_que) == 0 and len(normal_que) == 0:
                    break
                if len(ready_que) != 0:
                    if ready_que[0][2] > time_slice:
                        start_time.append(s_time)
                        s_time = s_time + time_slice
                        e_time = s_time
                        exit_time.append(e_time)
                        executed_pro.append(ready_que[0][0])
                        for j in range(len(pro_data)):
                            if pro_data[j][0] == ready_que[0][0]:
                                break
                        pro_data[j][2] = pro_data[j][2] - time_slice
                        ready_que.pop(0)
                    elif ready_que[0][2] <= time_slice:
                        start_time.append(s_time)
                        s_time = s_time + ready_que[0][2]
                        e_time = s_time
                        exit_time.append(e_time)
                        executed_pro.append(ready_que[0][0])
                        for j in range(len(pro_data)):
                            if pro_data[j][0] == ready_que[0][0]:
                                break
                        pro_data[j][2] = 0
                        pro_data[j][3] = 1
                        pro_data[j].append(e_time)
                        ready_que.pop(0)
                elif len(ready_que) == 0:
                    if s_time < normal_que[0][1]:
                        s_time = normal_que[0][1]
                    if normal_que[0][2] > time_slice:
                        start_time.append(s_time)
                        s_time = s_time + time_slice
                        e_time = s_time
                        exit_time.append(e_time)
                        executed_pro.append(normal_que[0][0])
                        for j in range(len(pro_data)):
```

```
                        if pro_data[j][0] == normal_que[0][0]:
                            break
                    pro_data[j][2] = pro_data[j][2] - time_slice
                elif normal_que[0][2] <= time_slice:
                    start_time.append(s_time)
                    s_time = s_time + normal_que[0][2]
                    e_time = s_time
                    exit_time.append(e_time)
                    executed_pro.append(normal_que[0][0])
                    for j in range(len(pro_data)):
                        if pro_data[j][0] == normal_que[0][0]:
                            break
                    pro_data[j][2] = 0
                    pro_data[j][3] = 1
                    pro_data[j].append(e_time)



            total_turnaround_time=0
            for i in range(n):
                turnaround_time = pro_data[i][5] - pro_data[i][1]
                total_turnaround_time = total_turnaround_time + turnaround_
time

                pro_data[i].append(turnaround_time)
            turn=total_turnaround_time/float(n)
            total_waiting_time = 0
            for i in range(len(pro_data)):
                waiting_time = pro_data[i][6] - pro_data[i][4]
                total_waiting_time = total_waiting_time + waiting_time
                pro_data[i].append(waiting_time)
            avg = total_waiting_time / float(n)

            pro_data.sort(key=lambda x: x[0])



            frame_1=tk.LabelFrame(rrb,text='output parameters',fg='black',b
g='skyblue',font='Times 12 bold italic',bd=5,width=250,height=100)
            frame_1.pack(anchor=tk.W)


            tk.Label(frame_1,text = "\n----
- Ater Performing Round Robin Scheduling Algorithm -----
\n",fg='black',bg='skyblue',font='Times 13 bold italic').pack(anchor = tk.W
)
```

```
            tk.Label(frame_1,text='  processID  arrival time    burst time
 completion time  Turnaround time    waiting time',fg='black',bg='skyblue',
font='Times 11 bold italic').pack(anchor=tk.W)
            for i in range(n):

                tk.Label(frame_1,text='     '+str(pro_data[i][0])+'\t
  '+str(pro_data[i][1])+'\t\t '+str(pro_data[i][4])+'\t  '+str(pro_data[i][
5])+'\t\t'+str(pro_data[i][6])+'\t\t'+str(pro_data[i][7]),fg='black',bg='sk
yblue',font='Times 11 bold italic').pack(anchor=tk.W)

            tk.Label(frame_1,text='Average turnaround time -
'+str(round(turn,4)),fg='black',bg='skyblue',font='Times 14 bold italic').p
ack(anchor=tk.W)
            tk.Label(frame_1,text='Average waiting time -
'+str(round(avg,4)),fg='black',bg='skyblue',font='Times 14 bold italic').pa
ck(anchor=tk.W)


            tk.Button(frame_1,text='CLEAR OUTPUT',command=clear,width='15',
font='Verdana 9 bold',fg='black',bg='light gray',activebackground='green',a
ctiveforeground='yellow').pack()

        rrb=tk.Tk()
        rrb.title('CPU SCHEDULING')
        rrb.geometry('565x650')
        rrb.configure(bg='skyblue')
        tit=tk.Label(rrb,text='Simulation of CPU scheduling algorithms',fg=
'black',bg='yellow',font='Times 16 bold italic')
        tit.pack()
        tk.Label(rrb,text = 'Project done by:',fg='black',bg='skyblue',font
='Times 12 bold italic').pack()
        tk.Label(rrb,text = 'Balaji - 179X1A02B0',fg='black',bg='skyblue',f
ont='Times 10 bold italic').pack()
        tk.Label(rrb,text = 'Raman - 179X1A0346',fg='black',bg='skyblue',fo
nt='Times 10 bold italic').pack()
        tk.Label(rrb,text = 'Mahesh - 179X1A03F6',fg='black',bg='skyblue',f
ont='Times 10 bold italic').pack()

        tk.Label(rrb,text = '\nRound Robin Scheduling\n',fg='black',bg='sky
blue',font='Times 12 bold italic').pack()

        frame=tk.LabelFrame(rrb,text='input parameters',fg='black',bg='skyb
lue',font='Times 12 bold italic',bd=5,width=250,height=100)
        frame.pack(anchor=tk.W,fill=tk.Y)
        tk.Label(frame,text=' No.of Processes ',fg='black',bg='skyblue',fon
t='Times 10 bold italic').grid(row=0,column=0,sticky=tk.W)
        tk.Label(frame,text=' Arrival Times separated by ,      ',fg='black'
,bg='skyblue',font='Times 10 bold italic').grid(row=2,column=0,sticky=tk.W)
        tk.Label(frame,text=' Burst Times separated by ,       ',fg='black',
bg='skyblue',font='Times 10 bold italic').grid(row=3,column=0,sticky=tk.W)
        tk.Label(frame,text=' Time Quantum ,       ',fg='black',bg='skyblue'
,font='Times 10 bold italic').grid(row=1,column=0,sticky=tk.W)


        entry1=tk.Entry(frame,width=10)
        entry1.insert(0,'3')
        entry1.grid(row=0,column=1)
        entry2=tk.Entry(frame,width=20)
        entry2.insert(0,'0,1,2')
        entry2.grid(row=2,column=1)
        entry3=tk.Entry(frame,width=20)
        entry3.insert(0,'10,2,5')
        entry3.grid(row=3,column=1)
        entry4=tk.Entry(frame,width=10)
        entry4.insert(0,'2')
        entry4.grid(row=1,column=1)

        tk.Label(rrb,bg='skyblue').pack()

        tk.Button(rrb,text='SUBMIT',command=sub,width='10',font='Verdana 9
bold',fg='black',bg='light gray',activebackground='green',activeforeground=
'yellow').pack()
        tk.Label(rrb,bg='skyblue').pack()
    def quit():
        sys.exit()

    tk.Button(win,text='First Come First Serve(FCFS)',command=FCFS,width='2
7',font='Verdana 9 bold',fg='black',bg='light gray',activebackground='green
',activeforeground='yellow').pack()
    tk.Label(win,bg='skyblue').pack()
    tk.Button(win,text='Shortest Job First(SJF)',command=SJF,width='27',fon
t='Verdana 9 bold',fg='black',bg='light gray',activebackground='green',acti
veforeground='yellow').pack()
    tk.Label(win,bg='skyblue').pack()
    tk.Button(win,text='Priority Based Scheduling',command=PB,width='27',fo
nt='Verdana 9 bold',fg='black',bg='light gray',activebackground='green',act
iveforeground='yellow').pack()
    tk.Label(win,bg='skyblue').pack()
    tk.Button(win,text='Round Robin Scheduling',command=RR,width='27',font=
'Verdana 9 bold',fg='black',bg='light gray',activebackground='green',active
foreground='yellow').pack()
    tk.Label(win,bg='skyblue').pack()
    tk.Button(win,text='Quit',command=quit,width='10',font='Verdana 9 bold'
,fg='black',bg='light gray',activebackground='green',activeforeground='yell
ow').pack()


    win.mainloop()



if __name__=="__main__":
    cpuschedu()
```

CONCLUSION

CONCLUSION

Results

The proposed project able to simulate CPU scheduling algorithms such as FCFS (First Come First Serve), SJF (Shortest Job First), Priority Scheduling and Round Robin algorithms with a user-friendly and mouse driven graphical user interface which is developed by using Python Tkinter framework. It takes number of processes, processes each with burst time, arrival time, priority for priority scheduling algorithm and time quantum for Round Robin algorithm and gives a table of output parameters such as completion time, turnaround time, waiting time for each process and average waiting time and average turnaround time for the given set of processes.

With this a student able to analyze and compare the CPU scheduling algorithms for different set of processes with different input parameters and can identify the best algorithm based on the following CPU scheduling criteria:

- Max throughput [Number of processes that complete their execution per time unit]
- Min turnaround time [Time taken by a process to finish execution]
- Min waiting time [Time a process waits in ready queue]

Future Work and Recommendation

The project can be extended further by adding following:

- Add visual effects to the program which will get more attention for the user when implementing and algorithm.
- Add sound effects to the program to make it more attractive.
- Use other CPU scheduling algorithms like multilevel queue and others and add it to this program.

REFERENCES

REFERENCES

References

[1] Silberschatz, Galivin, Gagne, 2002, “Operating System Concepts”, Sixth Edition, John Wiley & Sons, Inc.

[2] HTML5 Black Book, Edition 2019, Dreamtech Press.

[3] https://www.w3schools.com

[4] www.wikipedia.com

[5] https://www.geeksforgeeks.org/python-tkinter-tutorial/

[6] https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm

[7] https://www.javatpoint.com/os-scheduling-algorithms