|  |
| --- |
| **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**  **JNANASANGAMA, BELAGAVI – 590018**    **Mini Project Report**  **On**  **Implementation of Round Robin Scheduling Algorithm with Different Arrival Time**  *Submitted in partial fulfilment for the award of degree of*  **Bachelor of Engineering**  **In**  **Information Science and Engineering**  Submitted by  **Aadarsh Raj**-1RF19IS001    **RV Institute of Technology and Management®**  (Affiliated to VTU, Belagavi)  **JP Nagar, Bengaluru - 560076**  **Department of Information Science and Engineering** |

**RV Institute of Technology and Management®**

(Affiliated to VTU, Belagavi)

**JP Nagar, Bengaluru – 560076**

**Department of Information Science and Engineering**



**CERTIFICATE**

Certified that the Mini project entitled “**Implementation of Round Robin Scheduling Algorithm”** carried out by

**Aadarsh Raj**-1RF19IS001

Is a bonafied student of IV Semester B.E, **RV Institute of Technology and Management** in partial fulfilment for the Bachelor of Engineering in INFORMATION SCIENCE AND ENGINEERING, of the **Visvesvaraya Technological University,** Belagavi, during the academic year 2020 - 2021. The Mini project report has been approved as it satisfies the academic requirements in respect of **Design and Analysis of Algorithms(18CS42)**.

Deepak D J

Assistant Professor

Dept. ISE

RVITM, Bengaluru - 560076

B.E./Dept. of ISE/RVITM Page- 2 2020-2021

**Table of Contents**

**CONTENTS PageNo.**

1. Cover Page. 1.

2. Table of Content. 3.

3. Table of Figure and Tables. 4.

4. ABSTRACT 5.

4. Chapter-1 – Introduction 6.

1.1 Introduction to the project 6.

1.2 Introduction to the application 7.

1.3 Relevance of the application. 10.

5. Chapter-2 – Design -Algorithm 12.

6. Chapter-3 – Implementation details / Code 15.

7. Chapter-4 – Experimental Results/Snapshots 19.

8. Conclusion and future extension. 21.

9. References. 22.

B.E./Dept. of ISE/RVITM Page- 3 2020-2021

**Table of Figure and Tables:**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Topic** | **Page No.** |
| Fig. 1.1 | State Transition | 7 |
| Fig. 1.2 | Round-Round Flow Chart | 14 |
| Fig. 1.3 | Round-Robin Design | 12 |
| Fig. 1.4 | Output 1 | 21 |
| Fig. 1.5 | Output 2 | 22 |

B.E./Dept. of ISE/RVITM Page- 4 2020-2021

**ABSTRACT**

CPU scheduling is one of the most crucial operations performed by operating systems. Different conventional algorithms like FCFS, SJF, Priority, and RR (Round Robin) are available for CPU Scheduling. The effectiveness of Priority and Round Robin scheduling algorithm completely depends on selection of priority features of processes and on the choice of time quantum. In this project we are going to implement this algorithm which assigns time slice to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is both simple and easy to implement, and starvation -free. Round-Robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks. Round robin is most efficiently used method in operating system.

B.E./Dept. of ISE/RVITM Page- 5 2020-2021

**Chapter 1**

**INTRODUCTION:**

**1.1 CPU Scheduling:**

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 input/output etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fare.

* **CPU utilization**

To make the best use of CPU and not waste any CPU cycle, CPU would be working most of the time (Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded).

* **Throughput**

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

* **Turn Around Time (TAT):**

Turnaround time may simply deal with the total time it takes for a program to provide the required output, means the amount of time taken to fulfil a request. The concept thus overlaps with lead time and can be contrasted with cycle time.

In computing, turnaround time is the total time taken between the submission of a program/process/thread/task for execution and the return of the complete output to the customer/user. It may vary for various programming languages depending on the developer of the software or the program e user after the program is started.

* **Arrival Time:**

Respect to a process, Arrival Time is the Time at which the process arrives in the ready queue. Completion Scheduling of processes/work is done to finish the work on time.

B.E./Dept. of ISE/RVITM Page- 6 2020-2021

* **Burst Time:**

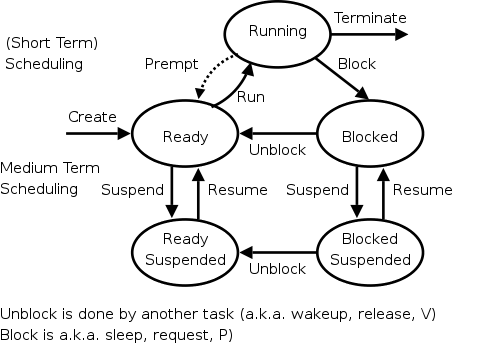
Burst Time is actually time that is required to complete execution of particular task or process. CPU Scheduling algorithms require Burst time as input.

* **Waiting Time:**

Throughput: number of processes completed per unit time. Turnaround Time: mean time from submission to completion of process. Waiting Time: Amount of time spent ready to run but not running. Response Time: Time between submission of requests and first response to the request.

**1.2 Operating System Schedulers:**

The scheduler is an operating system module that selects the next jobs to be admitted into the system and the next process to run. Operating systems may feature up to three distinct scheduler types: a long-term scheduler (also known as an admission scheduler or high-level scheduler), a mid-term or medium-term scheduler, and a short-term scheduler. The names suggest the relative frequency with which their functions are performed.

 **Fig.1.1 State Transition diagram of processes with different queues and schedulers**

B.E./Dept. of ISE/RVITM Page- 7 2020-2021

**Process scheduler:**

The process scheduler is a part of the operating system that decides which process runs at a certain point in time. It usually has the ability to pause a running process, move it to the back of the running queue and start a new process; such a scheduler is known pre-emptive scheduling otherwise it is a co-operative scheduling.

* **Long term Scheduling:**

The long-term scheduling decides which jobs or processes are to be admitted to the ready queue (in main memory); that is, when an attempt is made to execute a program, its admission to the set of currently executing processes is either authorized or delayed by the long-term scheduler. Thus, this scheduler dictates what processes are to run on a system, and the degree of concurrency to be supported at any one time – whether many or few processes are to be executed concurrently, and how the split between I/O-intensive and CPU-intensive processes is to be handled. The long-term scheduler is responsible for controlling the degree of multiprogramming.

In general, most processes can be described as either. An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations. A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations. It is important that a long-term scheduler selects a good process mix of I/O-bound and CPU-bound processes. If all processes, are I/O-bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do. On the other hand, if all processes are CPU-bound, the I/O waiting queue will almost always be empty, devices will go unused, and again the system will be unbalanced. The system with the best performance will thus have a combination of CPU-bound and I/O-bound processes. In modern operating systems, this is used to make sure that real-time processes get enough CPU time to finish their tasks

Long-term scheduling is also important in large-scale systems such as batch processing systems, computer clusters, supercomputers, and render farms. For example, in concurrent systems, schedule interacting processes is often required to prevent them from blocking due to waiting on each other. In these cases, special-purpose job scheduler software is typically use to assist these functions, in addition to any underlying admission scheduling support in the operating system.

* **Medium term Scheduling:**

The medium-term scheduling temporarily removes processes from main memory and places them in secondary memory (such as a hard disk drive) or vice versa, which is commonly referred

B.E./Dept. of ISE/RVITM Page- 8 2020-2021

to as "swapping out" or "swapping in" (also incorrectly as "paging out" or "paging in"). The medium-term scheduler may decide to swap out a process which has not been active for some time, or a process which has a low priority, or a process which is page faulting frequently, or a process which is taking up a large amount of memory in order to free up main memory for other processes, swapping the process back in later when more memory is available, or when the process has been unblocked and is no longer waiting for a resource. In many systems today (those that support mapping virtual address space to secondary storage other than the swap file), the medium-term scheduler may actually perform the role of the long-term scheduler, by treating binaries as "swapped out processes" upon their execution. In this way, when a segment of the binary is required, it can be swapped in on demand, or "lazy loaded".

* **Short term Scheduling:**

The short-term scheduling (also known as the CPU scheduler) decides which of the ready, in-memory processes is to be executed (allocated a CPU) after a clock interrupt, an I/O interrupt, an operating system call or another form of signal. Thus, the short-term scheduler makes scheduling decisions much more frequently than the long-term or mid-term schedulers – a scheduling decision will at a minimum have to be made after every time slice, and these are very short. This scheduler can be pre-emptive, implying that it is capable of forcibly removing processes from a CPU when it decides to allocate that CPU to another process, or non-pre-emptive (also known as "voluntary" or "co-operative"), in which case the scheduler is unable to "force" processes off the CPU.

A pre-emptive scheduler relies upon a programmable interval timer which invokes an interrupt handler that runs in and implements the scheduling function.

* **Dispatcher:**

Another component that is involved in the CPU-scheduling function is the dispatcher, which is the module that gives control of the CPU to the process selected by the short-term scheduler. It receives control in kernel mode as the result of an interrupt or system call. The functions of a dispatcher involve the following:

• Context switches, in which the dispatcher saves the state (also known as context) of the process or thread that was previously running; the dispatcher then loads the initial or previously saved state of the new process.

• Switching to user mode.

• Jumping to the proper location in the user program to restart that program indicated by its new state.

B.E./Dept. of ISE/RVITM Page- 9 2020-2021

The dispatcher should be as fast as possible, since it is invoked during every process switch. During the context switches, the processor is virtually idle for a fraction of time, thus unnecessary context switches should be avoided. The time it takes for the dispatcher to stop one process and start another is known as the dispatch latency

**1.3 ROUND ROBIN ALGORITHM:**

One of the oldest, simplest, fairest and most widely used algorithm is round robin (RR).

In the round robin scheduling, processes are dispatched in a FIFO manner but are given a limited amount of CPU time called a time-slice or a quantum.

If a process does not complete before its CPU-time expires, the CPU is pre-empted and given to the next process waiting in a queue. The pre-empted process is then placed at the back of the ready list.

Round Robin Scheduling is pre-emptive (at the end of time-slice) therefore it is effective in time-

sharing environments in which the system needs to guarantee reasonable response times for

interactive users.

The only interesting issue with round robin scheme is the length of the quantum. Setting the quantum too short causes too many context switches and lower the CPU efficiency. On the other hand, setting the quantum too long may cause poor response time and approximates FCFS.

In any event, the average waiting time under round robin scheduling is often quite long.

In the round robin each process will be given a special kind of an input known as TIME QUANTUM in the order of arrival time and we repeat the process until the process is terminated.

**TIME QUANTUM:**

A special kind of an input is given with a specific time slice which is fixed, based on which processes are terminated.

B.E./Dept. of ISE/RVITM Page- 10 2020-2021

**Advantages of Round – Robin Algorithm:**

* Every process gets an equal share of the CPU.
* RR is cyclic in nature, so there is no starvation.

**Disadvantages of Round – Robin Algorithm:**

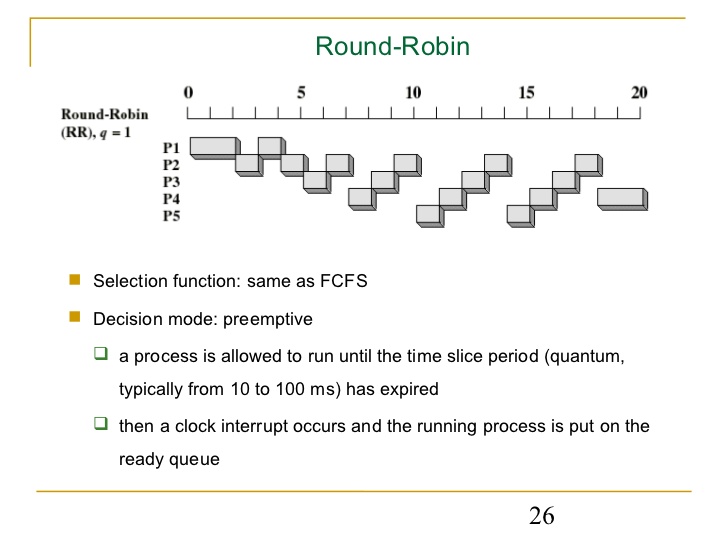
* Setting the quantum too short, increases the overhead and lowers the CPU efficiency, but setting it too long may cause poor response to short processes.
* Average waiting time under the RR policy is often long.

B.E./Dept. of ISE/RVITM Page- 11 2020-2021

**Chapter 2**

**Design -Algorithm:**

**Round Robin:**



**Fig. 1.3 Round–Robin Design**

**It is a time sharing (pre-emptive) scheduler where each process is given access to the CPU for 1 time quantum (slice) (e.g., 20 milliseconds)**

**1. a process may block itself before its time slice expires.**

**2. if it uses its entire time slice, it is then pre-empted and put at the end of the ready queue.**

**3. the ready queue is managed as a FIFO queue and treated as a circular.**

**4. if there are n processes on the ready queue and the time quantum is q, then each process**

**gets 1/n time on the CPU in chunks of at most q time units.**

**5. no process waits for more than (n-1) q time units.**

**6. the choice of how big to make the time slice (q) is extremely important.**

* **if q is very large, Round Robin degenerates into FCFS.**
* **if q is very small, the context switch overhead defeats the benefits.**

B.E./Dept. of ISE/RVITM Page- 12 2020-2021

**Pseudo Code:**

New process P arrives

P Enters ready queue

Update SR and AR

Process p is loaded from ready queue into the CPU to be executed

IF (Ready Queue is Empty)

TQ 🡨 BT (p)

Update SR and AR

End if

IF (Ready Queue is not empty)

TQ 🡨AVG (Sum BT of processes in ready queue)

Update SR and AR

End if

CPU executes P by TQ time

IF (P is terminated)

Update SR and AR

End if

IF (P is not terminated)

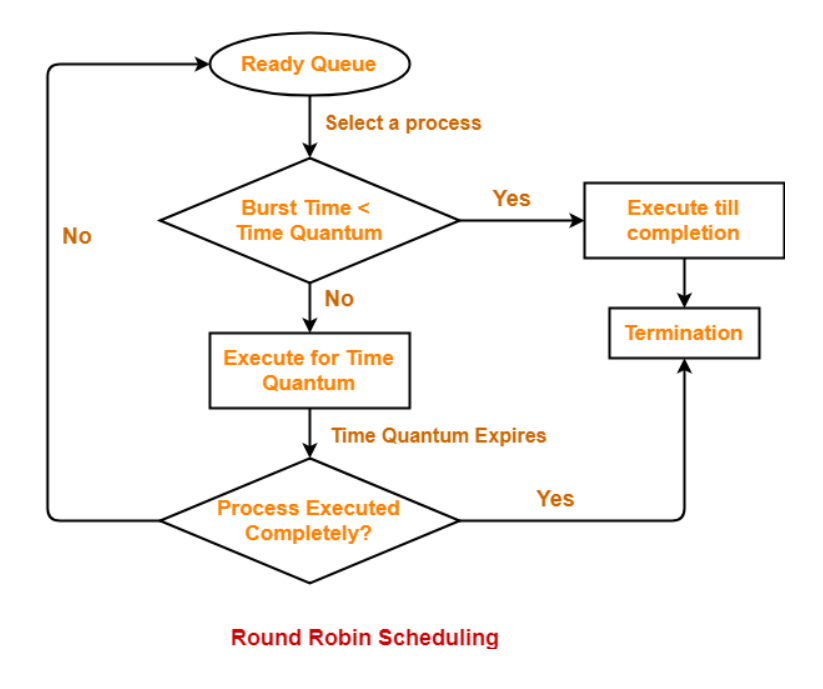
Return p to the ready queue with its updated burst time

Update SR and AR

End if

B.E./Dept. of ISE/RVITM Page- 13 2020-2021

**Flow Chart :-**

****

**Fig. 1.2 Round – Robin Flow Chart**

B.E./Dept. of ISE/RVITM Page- 14 2020-2021

**Example 1:** Processes = 4

Time Quantum = 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Processes | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time |
| 1 | 0 | 5 | 12 | 12 | 7 |
| 2 | 1 | 4 | 11 | 10 | 6 |
| 3 | 2 | 2 | 6 | 4 | 2 |
| 4 | 3 | 1 | 9 | 6 | 5 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P1 | P4 | P2 | P1 |

Ready Queue

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P1 | P4 | P2 | P1 |

Gantt Chart or Running Queue

0 2 4 6 8 9 11 12

* Completion Time = Last Time in Running Queue
* Turnaround Time = (Completion Time – Arrival Time)
* Waiting Time = (Turnaround Time – Arrival Time)

Number of Context Switching = 6

Average Turnaround Time = (Total Turnaround Time / Total Processes)

= (12+10+4+6) / 4

= 8

Average Waiting Time = (Total Waiting Time / Total Processes)

= (7+6+2+5) / 4

= 5

B.E./Dept. of ISE/RVITM Page- 15 2020-2021

**Example 2:** Processes = 5

Time Quantum = 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Processes | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time |
| 1 | 3 | 5 | 17 | 14 | 9 |
| 2 | 4 | 6 | 23 | 19 | 13 |
| 3 | 5 | 8 | 32 | 27 | 19 |
| 4 | 6 | 7 | 33 | 27 | 20 |
| 5 | 7 | 4 | 30 | 23 | 19 |

Ready Queue

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P1 | P5 | P2 | P3 | P4 | P5 | P3 | P4 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P1 | P5 | P2 | P3 | P4 | P5 | P3 | P4 |

Gantt Chart or Running Queue

3 6 9 12 15 17 20 23 26 29 30 32 33

* Completion Time = Last Time in Running Queue
* Turnaround Time = (Completion Time – Arrival Time)
* Waiting Time = (Turnaround Time – Arrival Time)

Number of Context Switching = 11

Average Turnaround Time = (Total Turnaround Time / Total Processes)

= (14+19+27+27+23) / 5

= 22

Average Waiting Time = (Total Waiting Time / Total Processes)

= (9+13+19+20+19) / 5

= 16

B.E./Dept. of ISE/RVITM Page- 16 2020-2021

**Chapter 3**

**Code:**

**package** roundrobin;

**import** java.util.\*;

**public** **class** RrScheduling {

**private** **static** Scanner *sc* = **new** Scanner(System.***in***);

**public** **static** **void** main(String[] args)

{

**int** n,tq, timer = 0, maxProccessIndex = 0,sum=0,s=0;

**float** avgWait = 0, avgTT = 0;

System.***out***.print("\nEnter the Time Quantum : ");

tq = *sc*.nextInt();

System.***out***.print("\nEnter the Number of Processess : ");

n = *sc*.nextInt();

**int** arrival[] = **new** **int**[n];

**int** burst[] = **new** **int**[n];

**int** wait[] = **new** **int**[n];

**int** turn[] = **new** **int**[n];

**int** queue[] = **new** **int**[n];

**int** temp\_burst[] = **new** **int**[n];

**boolean** complete[] = **new** **boolean**[n];

System.***out***.print("\nEnter the Arrival Time of the Processess : ");

**for**(**int** i = 0; i < n; i++)

arrival[i] = *sc*.nextInt();

System.***out***.print("\nEnter the Burst Time of the Processess : ");

**for**(**int** i = 0; i < n; i++){

burst[i] = *sc*.nextInt();

temp\_burst[i] = burst[i];

sum += burst[i];

}

**for**(**int** i = 0; i < n; i++){

complete[i] = **false**;

queue[i] = 0;

}

**while**(timer < arrival[0])

timer++;

queue[0] = 1;

**while**(s<sum)

{

s++;

**boolean** flag = **true**;

B.E./Dept. of ISE/RVITM Page- 17 2020-2021

**for**(**int** i = 0; i < n; i++)

{

**if**(temp\_burst[i] != 0)

{

flag = **false**;

**break**;

}

}

**if**(flag)

**break**;

**for**(**int** i = 0; (i < n) && (queue[i] != 0); i++)

{

**int** ctr = 0;

**while**((ctr < tq) && (temp\_burst[queue[0]-1] > 0))

{

temp\_burst[queue[0]-1] -= 1;

timer += 1;

ctr++;

*checkNewArrival*(timer, arrival, n, maxProccessIndex, queue);

}

**if**((temp\_burst[queue[0]-1] == 0) && (complete[queue[0]-1] == **false**))

{

turn[queue[0]-1] = timer;

complete[queue[0]-1] = **true**;

}

**boolean** idle = **true**;

**if**(queue[n-1] == 0)

{

**for**(**int** k = 0; k < n && queue[k] != 0; k++)

{

**if**(complete[queue[k]-1] == **false**)

{

idle = **false**;

}

}

}

**else**

idle = **false**;

**if**(idle)

{

timer++;

*checkNewArrival*(timer, arrival, n, maxProccessIndex, queue);

B.E./Dept. of ISE/RVITM Page- 18 2020-2021

}

*queueMaintainence*(queue,n);

}

}

**for**(**int** i = 0; i < n; i++)

{

turn[i] = turn[i] - arrival[i];

wait[i] = turn[i] - burst[i];

}

System.***out***.print("\nProgram No.\tArrival Time\tBurst Time\tWait Time\tTurnAround Time"

+ "\n");

**for**(**int** i = 0; i < n; i++)

{

**if**(wait[i] < 0)

wait[i] = -wait[i];

**if**(turn[i] < 0)

turn[i] = -turn[i];

System.***out***.print(i+1+"\t\t"+arrival[i]+"\t\t"+burst[i]

+"\t\t"+wait[i]+"\t\t"+turn[i]+ "\n");

}

**for**(**int** i =0; i< n; i++)

{

avgWait += wait[i];

avgTT += turn[i];

}

System.***out***.print("\nAverage Wait Time : "+(avgWait/n)

+"\nAverage Turn Around Time : "+(avgTT/n));

}

**public** **static** **void** queueUpdation(**int** queue[],**int** timer,**int** arrival[],**int** n, **int** maxProccessIndex){

**int** zeroIndex = -1;

**for**(**int** i = 0; i < n; i++){

**if**(queue[i] == 0){

zeroIndex = i;

**break**;

}

}

**if**(zeroIndex == -1)

**return**;

queue[zeroIndex] = maxProccessIndex + 1;

}

**public** **static** **void** checkNewArrival(**int** timer, **int** arrival[], **int** n, **int** maxProccessIndex,**int** queue[]){

**if**(timer <= arrival[n-1]){

**boolean** newArrival = **false**;

**for**(**int** j = (maxProccessIndex+1); j < n; j++){

B.E./Dept. of ISE/RVITM Page- 19 2020-2021

` **if**(arrival[j] <= timer){

**if**(maxProccessIndex < j){

maxProccessIndex = j;

newArrival = **true**;

}

}

}

**if**(newArrival)

*queueUpdation*(queue,timer,arrival,n, maxProccessIndex);

}

}

**public** **static** **void** queueMaintainence(**int** queue[], **int** n){

**for**(**int** i = 0; (i < n-1) && (queue[i+1] != 0) ; i++){

**int** temp = queue[i];

queue[i] = queue[i+1];

queue[i+1] = temp;

}

}

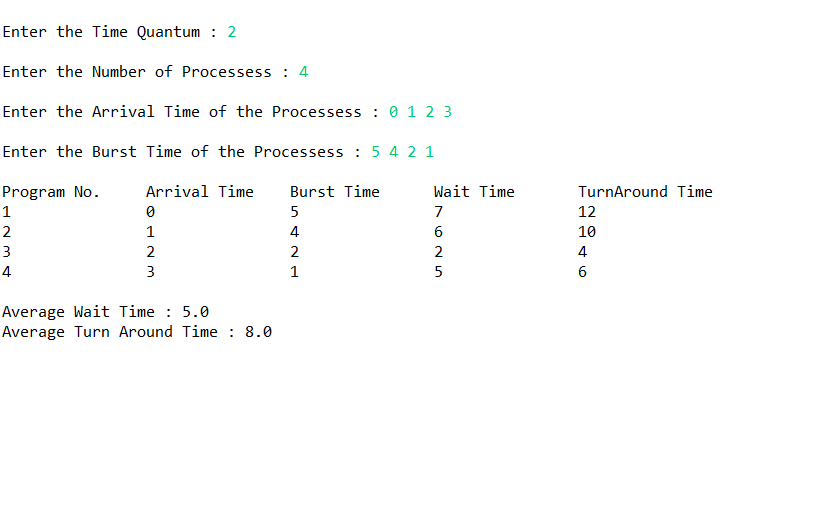
}

B.E./Dept. of ISE/RVITM Page- 20 2020-20

**Chapter 4**

**Experimental Results:**

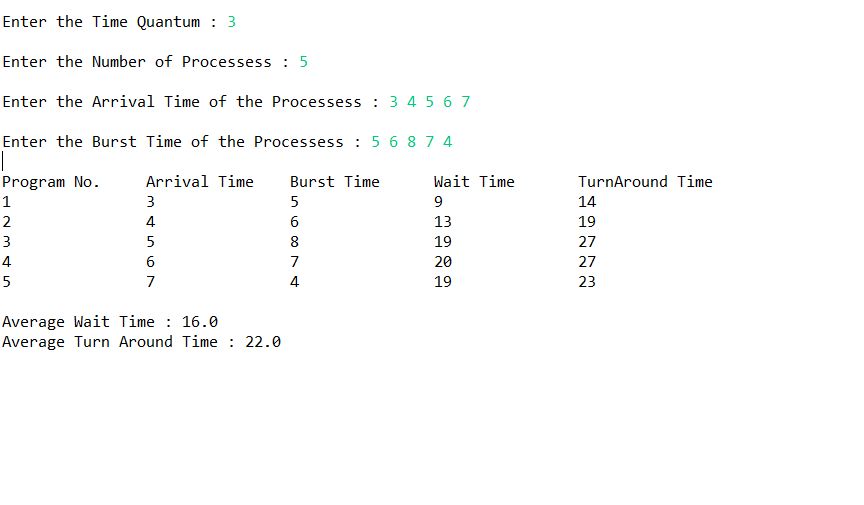
Output for Example 1 :-



**Fig. 1.4 Output of Example 1**

B.E./Dept. of ISE/RVITM Page- 21 2020-2021

Output for Example 2 :-



**Fig. 1.5 Output of Example 2**

B.E./Dept. of ISE/RVITM Page- 22 2020-2021

**Conclusion:**

So, with these results we get to know that the overall performance of the algorithm depends on

**1. Size of the time quantum**

* If time quantum is large than the CPU burst then this algorithm become same as FCFS and thus performance degrade.
* If the time quantum size is very small, then the number of content switches increases and the time quantum almost equal the time taken to switch the CPU from one process to another. Therefore 50% of time spent in switching of processes.

**2. Number of contexts switching:**

The number of context switches should not be too many to slow down the overall execution of all the processes. Time quantum should be large with respect to the context switch time. This is to ensure that a process keeps CPU for a maximum time as compared to the time spent in the context switching.

**Future Extensions:**

Round-robin scheduling can be applied to other scheduling problems, such as data packet scheduling in computer networks, which is an Operating System concept.

Since we know that Round-Robin scheduling algorithm is designed especially for time sharing system. So, in near future we can improve time sharing system by using this algorithm.

B.E./Dept. of ISE/RVITM Page- 23 2020-2021

**References:**

**Modules:**

* RVITM Modules for Data Structures & Applications (18CS42)
* RVITM Modules for Operating Systems (18CS43)

**Websites:**

* <https://www.geeksforgeeks.org/>
* <https://en.wikipedia.org/wiki/Main_Page>
* <https://www.youtube.com/c/nesoacademy>

**Textbooks:**

Abraham Silber Schatz, Peter Baer Galvin, Greg Gagne,  [**Operating System Principles**](https://drive.google.com/file/d/1MnvIYA-kzqydV_4EoYkfgLTKUVc4BACC/view?usp=sharing), 7" edition, Wiley-India, 2006

B.E./Dept. of ISE/RVITM Page- 24 2020-2021