OS Simulator Report

Course Title : CS252

Course Instructor : Shashidhar G Koolagudi

Group Members:

191CS101 : Aaidan Ram Meghwal

191CS102 : Aakarshee Jain

191CS103 : Abhishek Singh

191CS104 : Aditya Kumar

191CS105 : Aditya Santosh

191CS106 : Akanksha More

191CS107 : Amal Majunu Vidya

191CS108 : Anirudh Munnur Achal

191CS109 : Apurba Ranjan Mohapatra

191CS110 : Arnove Biswas

INDEX

Contribution of Members:

Roll Number	Member Name	Contribution
191CS101	Aaidan Ram Meghwal	FCFS and Round-Robin type of CPU Scheduling
191CS102	Aakarshee Jain	Disk Scheduling (FCFS), all the page Replacement algos (FIFO, LRU, MRU, OPTIMAL), Report Content, formatting, writing and output Formatting.
191CS103	Abhishek Singh	Preemptive and Nonpreemptive priority CPU scheduling, SCAN and CSCAN Disk Scheduling
191CS104	Aditya Kumar	Sleeping barber, Dining philosopher
191CS105	Aditya Santhosh	LJF, LRTF, SRTF CPU Scheduling, Bankers Algorithm.
191CS106	Akanksha More	Process Synchronization, CPU scheduling, Paging, SSTF, Report Formatting
191CS107	Amal Majunu Vidya	Process Synchronization, CPU Scheduling, Disk Scheduling
191CS108	Anirudh Munnur Achal	MFT Memory Management, MVT Memory Management, Bankers Algorithm, Code Merging and Resolving all compatibility issues, Output Formatting
191CS109	Apurba Ranjan Mohapatra	FCFS,SJF and Round Robin type CPU-scheduling,reader’s priority in process synchronization
191CS110	Arnove Biswas	Disk scheduling

Overview

The following report has been made as a part of course CS255- Operating Systems Lab Mini Project.

Goals

The significant goal of this simulator is to put together the different implementations related to various topics that we have learnt as part of the ***Operating Systems*** course throughout the duration of this semester and to learn more about the importance of operating systems.

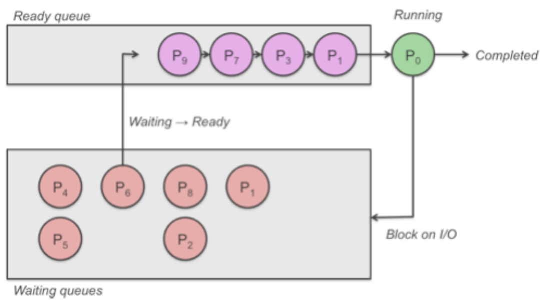
Specifications

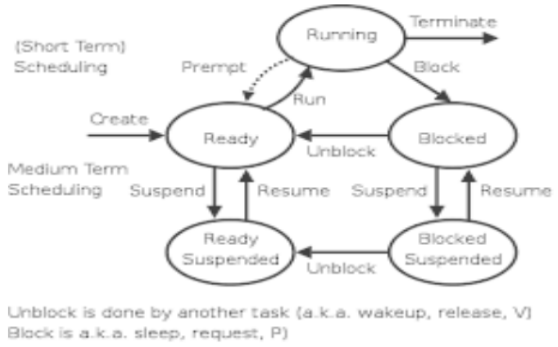
The user is presented with a menu displaying all the functions that can be implemented so as to delve deeper into the different topics of “***Operating Systems***”.

CPU Scheduling

Introduction:

CPU Scheduling is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least selects one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.The algorithms of CPU scheduling is given below.





<u>Different CPU Scheduling Algorithms:</u>

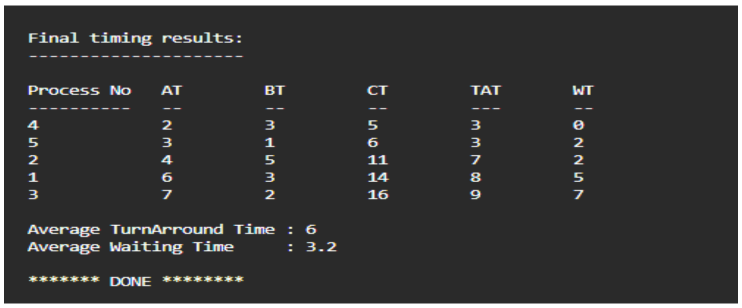
1] First Come First Serve (FCFS)

First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, processes which request the CPU first get the CPU allocation first. This is managed with a FIFO queue. The full form of FCFS is First Come First Serve.

Algorithm:

1. Processes are assigned to the processor in increasing order of arrival time.
2. Process with the smallest arrival time is allocated to the processor and its waiting time and turnaround time is calculated.
3. Above step is repeated until all processes are scheduled.

Output:



```
Final timing results:
---------------------

Process No    AT          BT          CT          TAT         WT
----------    --          --          --          ---         --
4             2           3           5           3           0
5             3           1           6           3           2
2             4           5           11          7           2
1             6           3           14          8           5
3             7           2           16          9           7

Average TurnArround Time : 6
Average Waiting Time     : 3.2

******* DONE ********
```

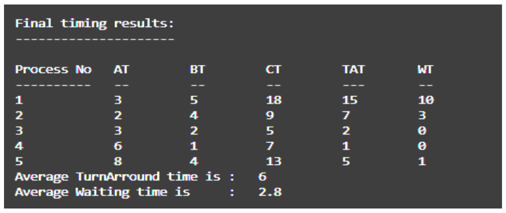
2] Shortest Job First (SJF)

Shortest Job First (SJF) is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method is non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution. The full form of SJF is Shortest Job First.SJF algorithms schedule processes in the order in which the shortest job is done first. It has a minimum average waiting time.

Algorithm:

1. Processes are assigned to the processor in increasing order of arrival time.
2. Process with the smallest arrival time is allocated to the processor and its waiting time and turnaround time is calculated.
3. Above step is repeated until all processes are scheduled.

Output:



```
Final timing results:
---------------------

Process No   AT        BT        CT        TAT       WT
----------   --        --        --        ---       --
1            3         5         18        15        10
2            2         4         9         7         3
3            3         2         5         2         0
4            6         1         7         1         0
5            8         4         13        5         1
Average TurnArround time is :   6
Average Waiting time is     :   2.8
```

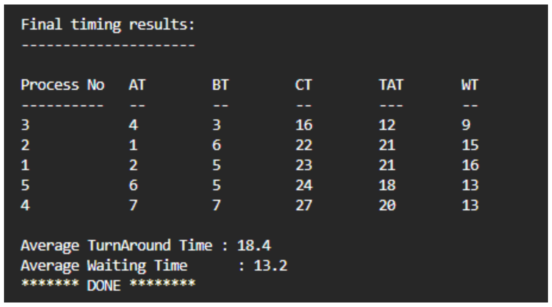
3] Round Robin

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. Round Robin is a preemptive scheduling algorithm where each process is assigned a time slot and after that it is preempted and CPU is assigned to the next process if it is in the ready queue.

Algorithm:

1. Sort the processes according to their arrival times.
2. Assign each process that is in the ready queue a certain time slot. If the time slot is greater than the process burst time, assign the process only its burst time and terminate that process.
3. Preempt the process after the time slot. Before adding it back to the queue, check if other processes arrived. If yes, then add those processes to the queue. If the current process has not finished its job, add it back to the queue.
4. Repeat the above steps until all the processes have been executed.

Output:



```
Final timing results:
---------------------

Process No   AT        BT        CT        TAT       WT
----------   --        --        --        ---       --
3            4         3         16        12        9
2            1         6         22        21        15
1            2         5         23        21        16
5            6         5         24        18        13
4            7         7         27        20        13

Average TurnAround Time : 18.4
Average Waiting Time      : 13.2
******* DONE ********
```

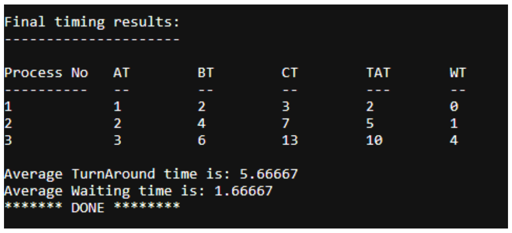
4] Shortest Remaining Time First (SRTF)

This is a preemptive version of Shortest Job First (SJF) scheduling algorithm. If a process with lower burst time arrives during execution of a process, running process is preempted.

Algorithm:

1. Sort the process according to their arrival times
2. Keep track of time that has passed with a counter variable. Increment it at every unit time.
3. Amongst the process that have arrived within our count of time, choose the process with lowest burst time, and update it’s remaining burst time. In every iteration, increment the count of time and choose appropriate processes based on burst time everytime.
4. Repeat above steps till all processes have been executed.

Output:



```
Final timing results:
--------------------

Process No   AT        BT        CT        TAT       WT
----------   --        --        --        ---       --
1            1         2         3         2         0
2            2         4         7         5         1
3            3         6         13        10        4

Average TurnAround time is: 5.66667
Average Waiting time is: 1.66667
******* DONE ********
```

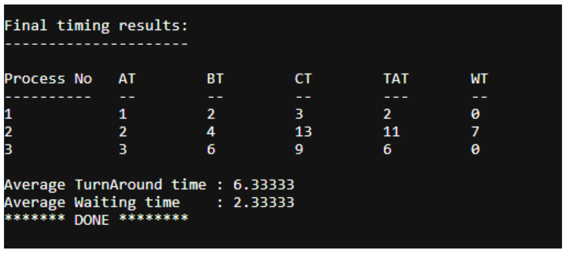
5] Longest Job First (LJF)

Longest Job First (LJF) is a non-preemptive scheduling algorithm. This algorithm is based upon the burst time of the processes. In this algorithm the process with the largest burst time is processed first.

Algorithm:

1. Sort the processes in increasing order of their Arrival Time.
2. Choose the process that has the highest Burst Time among all the processes that have arrived till that time.
3. Repeat above steps.

Output:



```
Final timing results:
--------------------

Process No   AT        BT        CT        TAT       WT
----------   --        --        --        ---       --
1            1         2         3         2         0
2            2         4         13        11        7
3            3         6         9         6         0

Average TurnAround time : 6.33333
Average Waiting time    : 2.33333
******* DONE ********
```

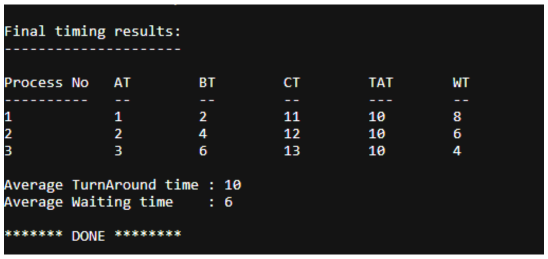
6] Longest Remaining Time First (LRTF)

This is a preemptive version of Longest Job First (LJF) scheduling algorithm. If a process with higher burst time arrives during execution of a process, the running process is preempted.

Algorithm:

1. Sort the process according to their arrival times
2. Keep track of time that has passed with a counter variable. Increment it at every unit time.
3. Amongst the process that have arrived within our count of time, choose the process with highest burst time, and update it's remaining burst time. In every iteration, increment the count of time and choose appropriate processes based on burst time everytime.
4. Repeat above steps till all processes have been executed.

Output:



```
Final timing results:
--------------------

Process No   AT       BT       CT       TAT      WT
----------   --       --       --       ---      --
1            1        2        11       10       8
2            2        4        12       10       6
3            3        6        13       10       4

Average TurnAround time : 10
Average Waiting time    : 6

******* DONE ********
```

7] Priority based (Non-Preemptive)

In Priority scheduling every process is assigned a priority beforehand according to certain factors such as time/memory/resource requirement and its importance and then these processes are executed according to the priority, processes with higher priority are executed first. If 2 or more processes have the same priority then they are executed on the first-come-first-serve basis.

In Non preemptive scheduling, a running process will only terminate or switch context after its full execution even if it is a lower priority. Arrival of a higher priority process while another process is running will not affect its execution.

Algorithm:

1. Find the maximum priority process among the processes in the ready queue to schedule.
2. Repeat the above step until all requests are handled.

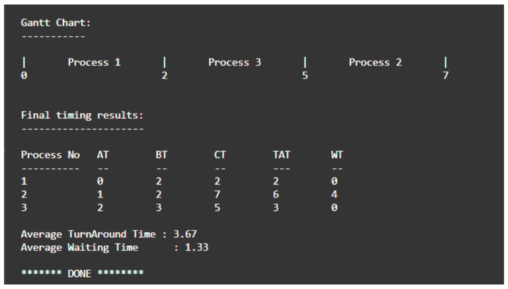
8]Priority based (Preemptive)

This is the preemptive version of priority scheduling which means if a process of higher priority comes in a ready queue then it will be executed first and the process currently running will be preempted.

Algorithm:

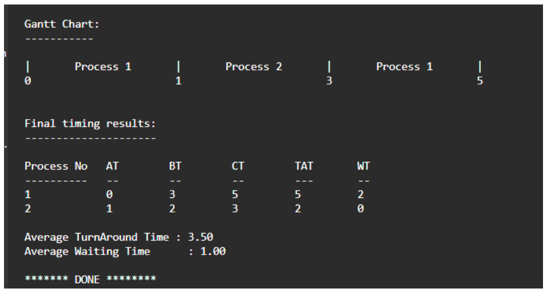
1. Find the maximum priority process among the processes in the ready queue to schedule.
2. Decrease the burst time of process by 1.
3. Increment the time variable by 1.
4. Repeat the above step until all requests are handled.

Output:



```
Gantt Chart:
-----------

|       Process 1       |       Process 3       |       Process 2       |
0                       2                       5                       7


Final timing results:
--------------------

Process No   AT       BT       CT       TAT      WT
----------   --       --       --       ---      --
1            0        2        2        2        0
2            1        2        7        6        4
3            2        3        5        3        0

Average TurnAround Time : 3.67
Average Waiting Time      : 1.33

******* DONE ********
```

Non preemptive priority scheduling



```
Gantt Chart:
-----------

|       Process 1       |       Process 2       |       Process 1       |
0                       1                       3                       5


Final timing results:
--------------------

Process No   AT       BT       CT       TAT      WT
----------   --       --       --       ---      --
1            0        3        5        5        2
2            1        2        3        2        0

Average TurnAround Time : 3.50
Average Waiting Time      : 1.00

******* DONE ********
```

Preemptive scheduling

9]Highest Response Ratio Next (HRRN) : Non-Preemptive

As the name suggests, the task is to schedule a job with the highest response ratio.

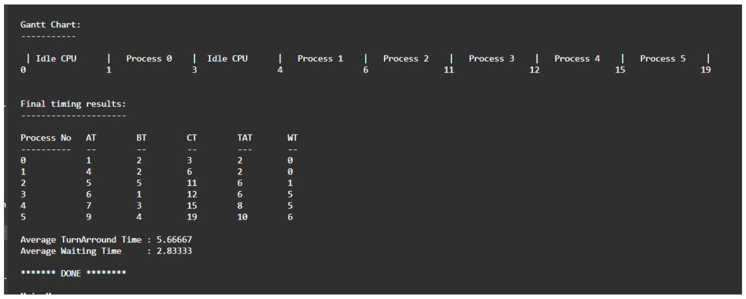
Response Ratio **=** $(WT + BT)/BT$

Processor is allocated to the process with the highest response ratio. This type scheduling favours shorter jobs/ processes. Aging increases the waiting time and hence the response ratio, thus this allows longer jobs to get past other shorter jobs.

Algorithm:

1. We schedule the process with the highest response ratio.
2. Processes that arrive when the processor is busy are added to the ready queue.
3. Above steps are repeated unless all processes are scheduled.
4. Finally output is average waiting time and average turnaround time

Output:



```
Gantt Chart:
-----------

 | Idle CPU    |   Process 0   |  Idle CPU    |   Process 1   |   Process 2   |   Process 3   |   Process 4   |   Process 5   |
0              1               3              4               6               11              12              15              19


Final timing results:
--------------------

Process No   AT       BT       CT       TAT      WT
----------   --       --       --       ---      --
0            1        2        3        2        0
1            4        2        6        2        0
2            5        5        11       6        1
3            6        1        12       6        5
4            7        3        15       8        5
5            9        4        19       10       6

Average TurnArround Time : 5.66667
Average Waiting Time     : 2.83333

******* DONE ********
```

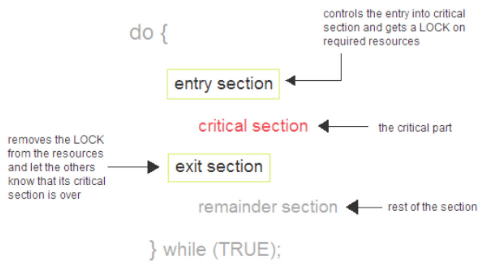
Process Synchronization Problems

Introduction:

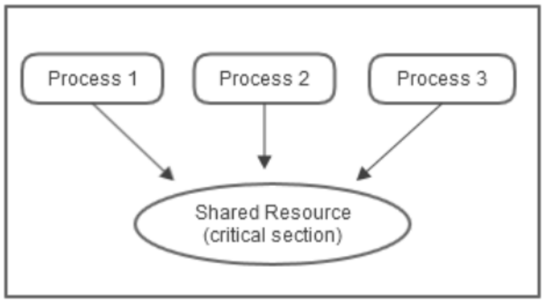
Process Synchronization means sharing system resources by processes in such a way that Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Critical Section:

Critical section is a code segment that can be accessed by only one process at a time. Critical section contains shared variables which need to be synchronized to maintain consistency of data variables.



do {
controls the entry into critical section and gets a LOCK on required resources
entry section
critical section
the critical part
removes the LOCK from the resources and let the others know that its critical section is over
exit section
remainder section
rest of the section
} while (TRUE);



Any solution to the critical section problem must satisfy three requirements:

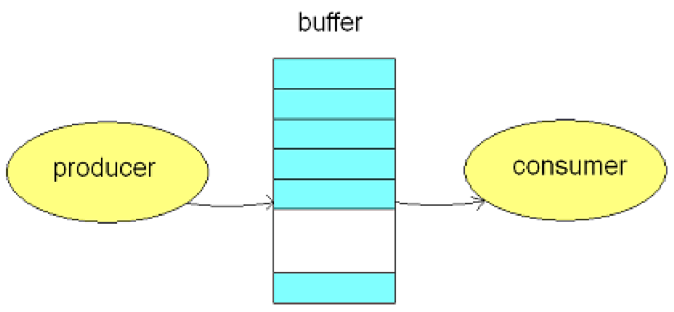
- **Mutual Exclusion :** If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
- **Progress :** If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter in the critical section next, and the selection can not be postponed indefinitely.
- **Bounded Waiting :** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

1]Producer-Consumer

Producer consumer problem is a classical synchronization problem. We consider a buffer of fixed size. A producer can produce an item and can place it in the buffer. A consumer can pick items and can consume them. It should also be ensured that while the producer is producing an item, consumers should not consume at the same time. Thus, only one of them can access the buffer at a time thus ensuring synchronization. The code uses an array for buffer and 2 variables - full,empty to check for overflow and underflow conditions.

Overflow condition: When the buffer is full, the producer cannot produce an item unless a consumer consumes an item: (full = 1, empty =0). If no producer exists, the program terminates.

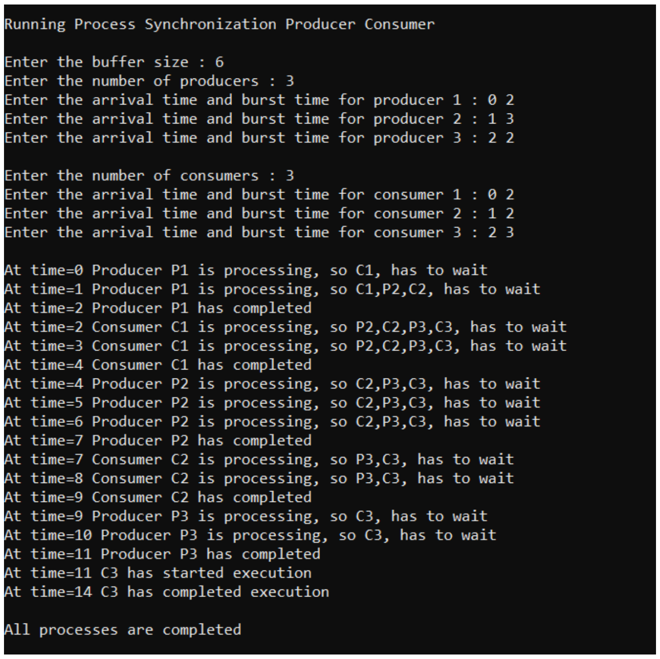
Underflow condition: When the buffer is empty, the consumers cannot consume unless a producer produces an item. (full = 0, empty =1). If no consumer exists, the program terminates.



Algorithm:

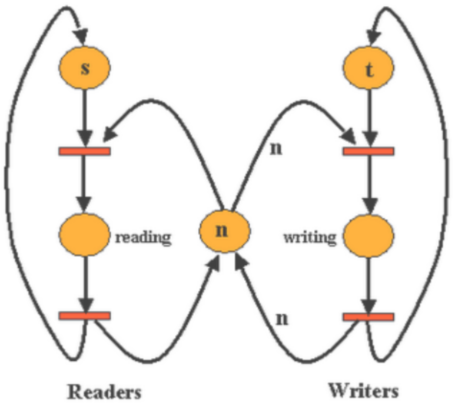
1. Merge both arrays of producer and consumer and sort them based on their arrival time.
2. If it is the producer's turn, check for overflow. If there is an overflow, shift its position to that after a consumer. If no overflow, increment the variable full and decrement variable empty. Allow the producer to produce for it's burst time. Assign the producer's completion time.
3. If it is the consumer's turn, check for underflow. If there is an underflow, shift its position to that after a producer. If no underflow, decrement the variable full and increment the variable empty. Allow the consumer to consume for it's burst time. Assign the consumer's completion time.
4. Repeat the above steps until no more producers can produce or no more consumers can consume.

Output



```
Running Process Synchronization Producer Consumer

Enter the buffer size : 6
Enter the number of producers : 3
Enter the arrival time and burst time for producer 1 : 0 2
Enter the arrival time and burst time for producer 2 : 1 3
Enter the arrival time and burst time for producer 3 : 2 2

Enter the number of consumers : 3
Enter the arrival time and burst time for consumer 1 : 0 2
Enter the arrival time and burst time for consumer 2 : 1 2
Enter the arrival time and burst time for consumer 3 : 2 3

At time=0 Producer P1 is processing, so C1, has to wait
At time=1 Producer P1 is processing, so C1,P2,C2, has to wait
At time=2 Producer P1 has completed
At time=2 Consumer C1 is processing, so P2,C2,P3,C3, has to wait
At time=3 Consumer C1 is processing, so P2,C2,P3,C3, has to wait
At time=4 Consumer C1 has completed
At time=4 Producer P2 is processing, so C2,P3,C3, has to wait
At time=5 Producer P2 is processing, so C2,P3,C3, has to wait
At time=6 Producer P2 is processing, so C2,P3,C3, has to wait
At time=7 Producer P2 has completed
At time=7 Consumer C2 is processing, so P3,C3, has to wait
At time=8 Consumer C2 is processing, so P3,C3, has to wait
At time=9 Consumer C2 has completed
At time=9 Producer P3 is processing, so C3, has to wait
At time=10 Producer P3 is processing, so C3, has to wait
At time=11 Producer P3 has completed
At time=11 C3 has started execution
At time=14 C3 has completed execution

All processes are completed
```

2]Reader-Writer



If we have a single shared file among multiple processes which has following conditions for editing or reading:

1. Only one process at a time can edit the file concurrently.
2. One or more processes can read the file at the same time but no cannot write the file.

Readers-Writers problem helps in dealing with such a situation. Similar problem can be demonstrated as follows:
There exists a shareable document which allows a user to read or write it. A reader reads the document whereas a writer writes the document.
Thus, above conditions can be applied as follows:

1. Only one writer can write a document at a time.

2. One or more readers can read the document at the same time.
3. When a writer is writing, no other reader/writer can read/write the document.
4. When one or more readers are reading, no writer can write the document.

Priority:
Reader’s Priority:
When a reader and a writer arrive at the same or are waiting, we will give more priority to the reader to next access the document. This may happen in following conditions:

1. If the document is not being read/written, and a reader and writer arrive at the same, then the reader is given priority to read the document.
2. If the document is being written, and a reader and writer are waiting, then next access to the document is given to the reader even if the writer arrived before the reader.

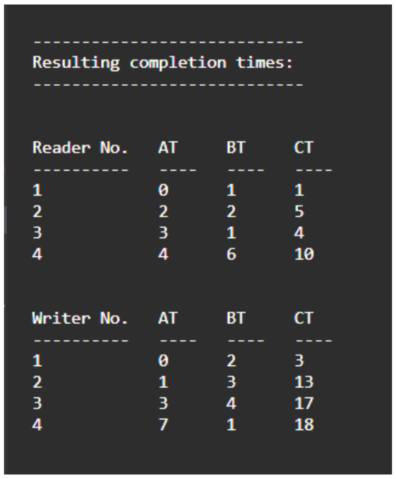
Writer’s Priority:
When a reader and a writer arrive at the same or are waiting, we will give more priority to the writer to next access the document. This may happen in following conditions:

1. If the document is not being read/written, and a reader and writer arrive at the same, then the writer is given priority to read the document.
2. If the document is being written, and a reader and writer are waiting, then next access to the document is given to the writer even if the reader arrived before the writer.

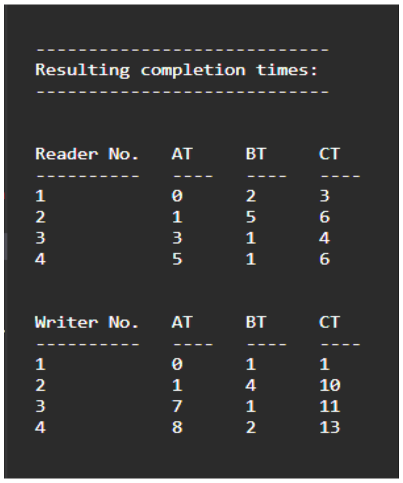
Algorithm:

1. Readers and writers are sorted according to their arrival time and merged in a single array. If their arrival times are the same then the one with higher priority is ordered first.
2. Reader/writer is allowed access based on priority.
3. If current user is writer:
    a. In case of writer’s priority, the writer is allowed access to the document.
    b. In case of reader’s priority, first we check if there are any readers waiting. In case of that, we allow access to the reader and the writer keeps waiting again.
4. If current user is reader:
    a. In case of reader’s priority, the reader is allowed access to the document. Also we check for other readers in the waiting queue and allow them to read as well.
    b. In case of writer’s priority, first we check if there are any writers waiting. In that case, we allow access to the writer who first arrived in waiting queue.
5. If the document is in read mode, then all the arriving readers are allowed to read while writers are kept in waiting.

6. If the document is in write mode, then all incoming readers and writers are kept in waiting.
7. Once the document is free, we again repeat the above steps unless all the readers are writers who have completed the task.

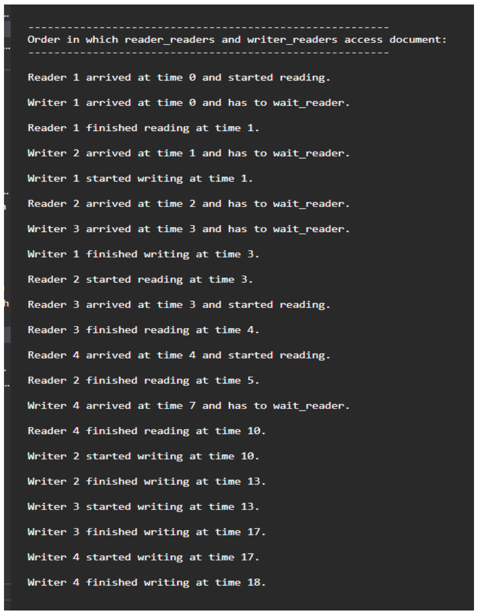


```
----------------------------
Resulting completion times:
----------------------------


Reader No.   AT    BT    CT
----------   ----  ----  ----
1            0     1     1
2            2     2     5
3            3     1     4
4            4     6     10


Writer No.   AT    BT    CT
----------   ----  ----  ----
1            0     2     3
2            1     3     13
3            3     4     17
4            7     1     18
```

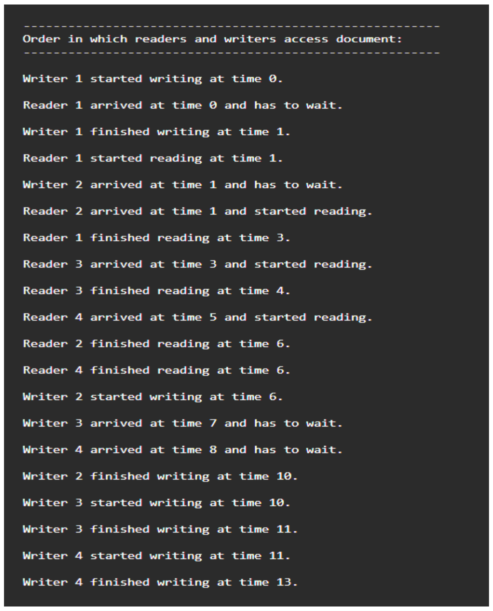


```
----------------------------
Resulting completion times:
----------------------------


Reader No.   AT    BT    CT
----------   ----  ----  ----
1            0     2     3
2            1     5     6
3            3     1     4
4            5     1     6


Writer No.   AT    BT    CT
----------   ----  ----  ----
1            0     1     1
2            1     4     10
3            7     1     11
4            8     2     13
```

Reader's Priority **Writer's Priority**



```
---------------------------------------------------------
Order in which reader_readers and writer_readers access document:
---------------------------------------------------------

Reader 1 arrived at time 0 and started reading.

Writer 1 arrived at time 0 and has to wait_reader.

Reader 1 finished reading at time 1.

Writer 2 arrived at time 1 and has to wait_reader.

Writer 1 started writing at time 1.

Reader 2 arrived at time 2 and has to wait_reader.

Writer 3 arrived at time 3 and has to wait_reader.

Writer 1 finished writing at time 3.

Reader 2 started reading at time 3.

Reader 3 arrived at time 3 and started reading.

Reader 3 finished reading at time 4.

Reader 4 arrived at time 4 and started reading.

Reader 2 finished reading at time 5.

Writer 4 arrived at time 7 and has to wait_reader.

Reader 4 finished reading at time 10.

Writer 2 started writing at time 10.

Writer 2 finished writing at time 13.

Writer 3 started writing at time 13.

Writer 3 finished writing at time 17.

Writer 4 started writing at time 17.

Writer 4 finished writing at time 18.
```

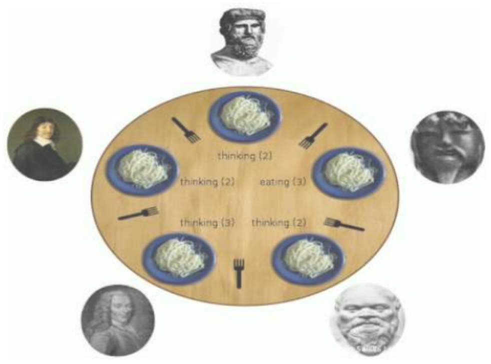


```
---------------------------------------------------------
Order in which readers and writers access document:
---------------------------------------------------------

Writer 1 started writing at time 0.

Reader 1 arrived at time 0 and has to wait.

Writer 1 finished writing at time 1.

Reader 1 started reading at time 1.

Writer 2 arrived at time 1 and has to wait.

Reader 2 arrived at time 1 and started reading.

Reader 1 finished reading at time 3.

Reader 3 arrived at time 3 and started reading.

Reader 3 finished reading at time 4.

Reader 4 arrived at time 5 and started reading.

Reader 2 finished reading at time 6.

Reader 4 finished reading at time 6.

Writer 2 started writing at time 6.

Writer 3 arrived at time 7 and has to wait.

Writer 4 arrived at time 8 and has to wait.

Writer 2 finished writing at time 10.

Writer 3 started writing at time 10.

Writer 3 finished writing at time 11.

Writer 4 started writing at time 11.

Writer 4 finished writing at time 13.
```

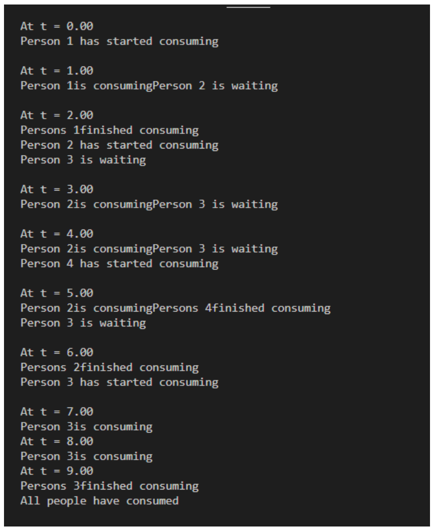
Reader's Priority **Writers's Priority**

3]Dining Philosophers problem

The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick-up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.



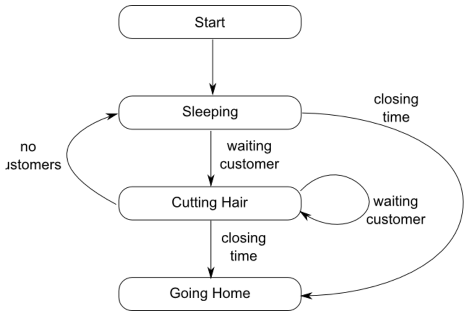
Output:



```
At t = 0.00
Person 1 has started consuming

At t = 1.00
Person 1is consumingPerson 2 is waiting

At t = 2.00
Persons 1finished consuming
Person 2 has started consuming
Person 3 is waiting

At t = 3.00
Person 2is consumingPerson 3 is waiting

At t = 4.00
Person 2is consumingPerson 3 is waiting
Person 4 has started consuming

At t = 5.00
Person 2is consumingPersons 4finished consuming
Person 3 is waiting

At t = 6.00
Persons 2finished consuming
Person 3 has started consuming

At t = 7.00
Person 3is consuming
At t = 8.00
Person 3is consuming
At t = 9.00
Persons 3finished consuming
All people have consumed
```

4]Sleeping Barbers problem

The barber shop has one barber and chairs for waiting customers, if any, to sit on. When a customer arrives, he has to wake up the sleeping barber. If additional customers arrive while the barber is cutting a customer's hair, they either sit down (if there are empty chairs) or leave the shop (if all chairs are full). The problem is to program the barber and the customers without getting into race conditions. This problem is similar to various queueing situations, such as a multi person helpdesk with a computerized call waiting system for holding a limited number of incoming calls.



Resource Allocation and Deadlock

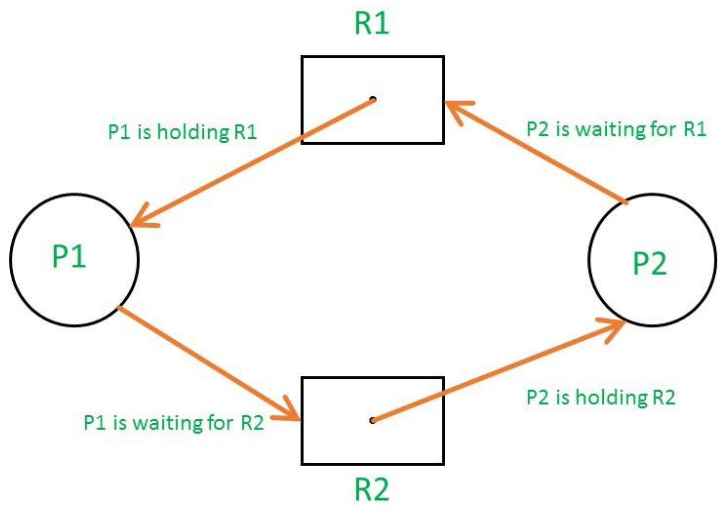
Introduction:

The Operating System allocates resources when a program needs them. When the program terminates, the resources are de-allocated, and allocated to other programs that need them.

Deadlock: It is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Causes of deadlock

- **Mutual Exclusion:** One or more than one resource are non-shareable (Only one process can use at a time)
- **Hold and Wait:** A process is holding at least one resource and waiting for resources.
- **No Preemption:** A resource cannot be taken from a process unless the process releases the resource.
- **Circular Wait:** A set of processes are waiting for each other in circular form.



Deadlock situation

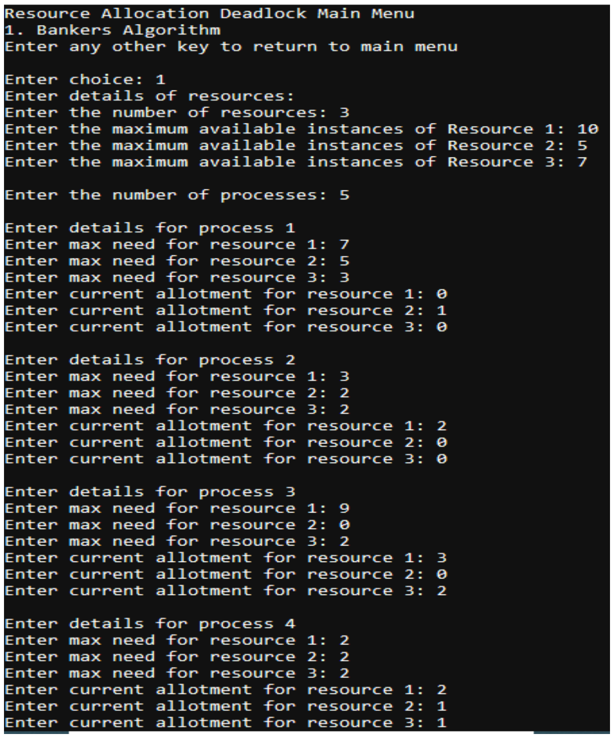
1]Banker's Algorithm

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

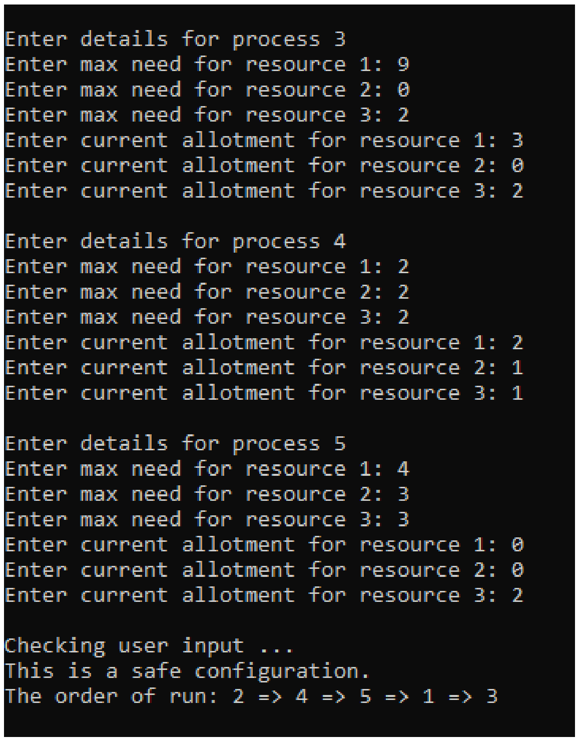
Algorithm:

1. We maintain a matrix such that we have information about each process and the number of each type of resource needed corresponding to each process. We also maintain the number of resources currently available.
2. In each iteration we check if we can meet the requirements of all resources needed by a process. Only if all requirements can be met, do we allocate resources. After doing so, the process is marked as completed and then all it’s resources are put back into the available resources array so that other processes can use these resources. This process is printed out in the safe sequence.
3. The above steps are repeated until all processes have completed execution.

Outputs



```
Resource Allocation Deadlock Main Menu
1. Bankers Algorithm
Enter any other key to return to main menu

Enter choice: 1
Enter details of resources:
Enter the number of resources: 3
Enter the maximum available instances of Resource 1: 10
Enter the maximum available instances of Resource 2: 5
Enter the maximum available instances of Resource 3: 7

Enter the number of processes: 5

Enter details for process 1
Enter max need for resource 1: 7
Enter max need for resource 2: 5
Enter max need for resource 3: 3
Enter current allotment for resource 1: 0
Enter current allotment for resource 2: 1
Enter current allotment for resource 3: 0

Enter details for process 2
Enter max need for resource 1: 3
Enter max need for resource 2: 2
Enter max need for resource 3: 2
Enter current allotment for resource 1: 2
Enter current allotment for resource 2: 0
Enter current allotment for resource 3: 0

Enter details for process 3
Enter max need for resource 1: 9
Enter max need for resource 2: 0
Enter max need for resource 3: 2
Enter current allotment for resource 1: 3
Enter current allotment for resource 2: 0
Enter current allotment for resource 3: 2

Enter details for process 4
Enter max need for resource 1: 2
Enter max need for resource 2: 2
Enter max need for resource 3: 2
Enter current allotment for resource 1: 2
Enter current allotment for resource 2: 1
Enter current allotment for resource 3: 1
```



```
Enter details for process 3
Enter max need for resource 1: 9
Enter max need for resource 2: 0
Enter max need for resource 3: 2
Enter current allotment for resource 1: 3
Enter current allotment for resource 2: 0
Enter current allotment for resource 3: 2

Enter details for process 4
Enter max need for resource 1: 2
Enter max need for resource 2: 2
Enter max need for resource 3: 2
Enter current allotment for resource 1: 2
Enter current allotment for resource 2: 1
Enter current allotment for resource 3: 1

Enter details for process 5
Enter max need for resource 1: 4
Enter max need for resource 2: 3
Enter max need for resource 3: 3
Enter current allotment for resource 1: 0
Enter current allotment for resource 2: 0
Enter current allotment for resource 3: 2

Checking user input ...
This is a safe configuration.
The order of run: 2 => 4 => 5 => 1 => 3
```

Memory Management

Introduction:

The operating system partitions the memory. Then allocates the processes in those partitions. Algorithms for creating partitions are given below. In the variable partitioning(MVT) algorithm, we partition the memory according to size of the process. In MFT, the partitions are already fixed.

1]First Fit
In this algorithm, we allocate the process in the first available partition that can fit the process. If the size of the partition is more than the size of the process, then another partition will be formed from the current partition equal to size of previous partition-size of process.

2]Best Fit
In this algorithm, the process will be allocated in that partition which has available space for process and lowest difference of size of partition and size of process. If the size of the partition is more than the size of the process, then another partition will be formed from the current partition equal to size of previous partition-size of process.

3]Worst Fit
In this algorithm, the process will be allocated in that partition which has available space for process and largest difference of size of partition and size of process. If the size of the partition is more than the size of the process, then another partition will be formed from the current partition equal to size of previous partition-size of process.

Variable partitioning:

Initially, there are no partitions in memory. Partitions are created as processes start occupying memory. It is a contiguous memory management technique in which the main memory is not divided into partitions and the process is allocated a chunk of free memory that is big enough for it to fit. The space which is left is considered as the free space which can be further used by other processes. It also provides the concept of compaction. In compaction the spaces that are free and the spaces which are not allocated to the process are combined and a single large memory space is made.
Two operations performed are:

1. Insert

    Algorithm:

    a. Process size and process id are taken as input.
    b. If the process id already exists in memory then we continue with further input.
    c. If process size is more than available space we continue with further input.

   d. A new partition is created for the process.
   e. Further we find the partition in memory:
      i. First : first partition large enough to accommodate the process
      ii. Best : partition that is large enough as well minimum to accommodate the process.
      iii. Worst : partition that is large enough as well maximum to accommodate the process.
   f. If no such partition is found then the new partition is discarded and the process cannot be allocated.
   g. If there are no partitions in the memory, this means memory is empty and hence we create a partition and allocate the process if its size does not exceed memory size.
   h. If an empty partition is found then we allocate the process to that partition, if the size of partition is larger than the process size then we split it to obtain an empty partition of leftover size.

2. Delete

   Algorithm:

   a. Process id is taken as input.
   b. We find the partition to which the given process id belongs.
   c. If no such partition is found we continue with further input.
   d. If the partition is found then we remove the process from that partition and we merge the partition with neighbouring partitions in case they are empty.

Fixed partitioning:

Multiprogramming with fixed partitioning is a contiguous memory management technique in which the main memory is divided into fixed sized partitions which can be of equal or unequal size. Whenever we have to allocate a process memory then a free partition that is big enough to hold the process is found. Then the memory is allocated to the process.If there is no free space available then the process waits in the queue to be allocated memory. It is one of the oldest memory management techniques which is easy to implement.

Two operations performed are:

1. Insert

   Algorithm:

   a. Process size and process id are taken as input.
   b. If the process id already exists in memory then we continue with further input.
   c. If process size is more than partition size or if it is more than available space we continue with further input.
   d. Further we find the partition in memory:
      i. First : first partition large enough to accommodate the process

ii. Best : partition that is large enough as well minimum to accommodate the process.
iii. Worst : partition that is large enough as well maximum to accommodate the process.

e. If no such partition is found then the new partition is discarded and the process cannot be allocated.

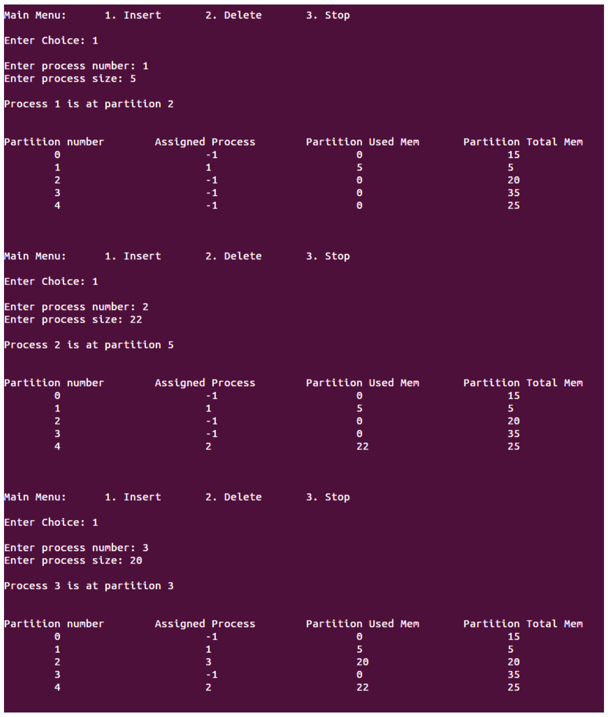
2. Delete

Algorithm:

a. Process id is taken as input.
b. We find the partition to which the given process id belongs.
c. If no such partition is found we continue with further input.
d. If the partition is found then we remove the process from that partition and we merge the partition with neighbouring partitions in case they are empty

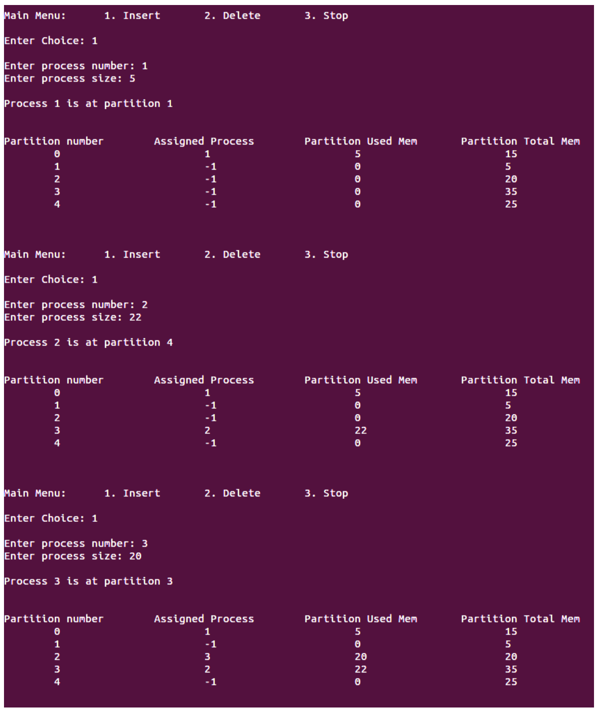
Outputs:

MFT - Best



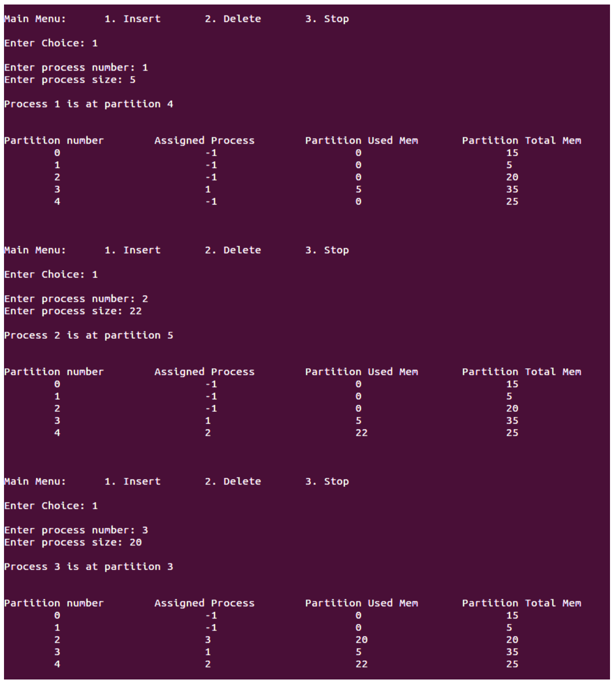
```
Main Menu:      1. Insert       2. Delete       3. Stop

Enter Choice: 1

Enter process number: 1
Enter process size: 5

Process 1 is at partition 2


Partition number        Assigned Process        Partition Used Mem       Partition Total Mem
        0                       -1                      0                       15
        1                       1                       5                       5
        2                       -1                      0                       20
        3                       -1                      0                       35
        4                       -1                      0                       25



Main Menu:      1. Insert       2. Delete       3. Stop

Enter Choice: 1

Enter process number: 2
Enter process size: 22

Process 2 is at partition 5


Partition number        Assigned Process        Partition Used Mem       Partition Total Mem
        0                       -1                      0                       15
        1                       1                       5                       5
        2                       -1                      0                       20
        3                       -1                      0                       35
        4                       2                       22                      25



Main Menu:      1. Insert       2. Delete       3. Stop

Enter Choice: 1

Enter process number: 3
Enter process size: 20

Process 3 is at partition 3


Partition number        Assigned Process        Partition Used Mem       Partition Total Mem
        0                       -1                      0                       15
        1                       1                       5                       5
        2                       3                       20                      20
        3                       -1                      0                       35
        4                       2                       22                      25
```

MFT-First



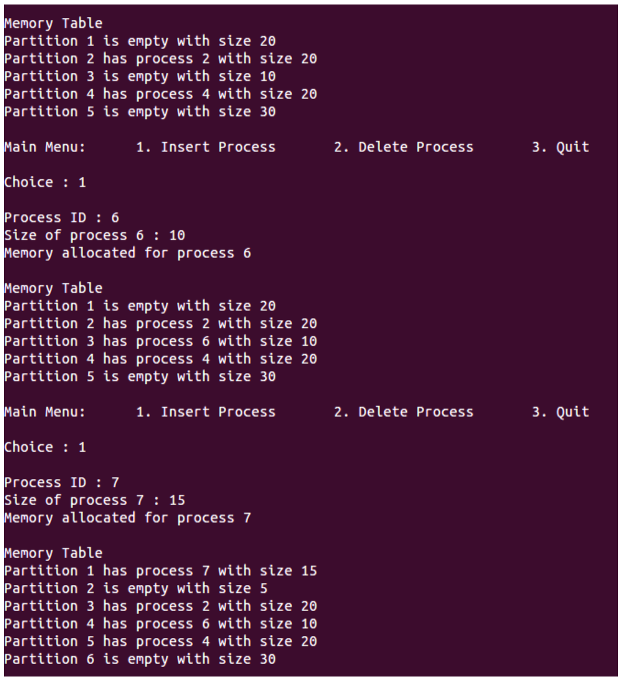
```
Main Menu:      1. Insert       2. Delete       3. Stop

Enter Choice: 1

Enter process number: 1
Enter process size: 5

Process 1 is at partition 1


Partition number        Assigned Process        Partition Used Mem       Partition Total Mem
        0                       1                       5                       15
        1                       -1                      0                       5
        2                       -1                      0                       20
        3                       -1                      0                       35
        4                       -1                      0                       25



Main Menu:      1. Insert       2. Delete       3. Stop

Enter Choice: 1

Enter process number: 2
Enter process size: 22

Process 2 is at partition 4


Partition number        Assigned Process        Partition Used Mem       Partition Total Mem
        0                       1                       5                       15
        1                       -1                      0                       5
        2                       -1                      0                       20
        3                       2                       22                      35
        4                       -1                      0                       25



Main Menu:      1. Insert       2. Delete       3. Stop

Enter Choice: 1

Enter process number: 3
Enter process size: 20

Process 3 is at partition 3


Partition number        Assigned Process        Partition Used Mem       Partition Total Mem
        0                       1                       5                       15
        1                       -1                      0                       5
        2                       3                       20                      20
        3                       2                       22                      35
        4                       -1                      0                       25
```

MFT - Worst



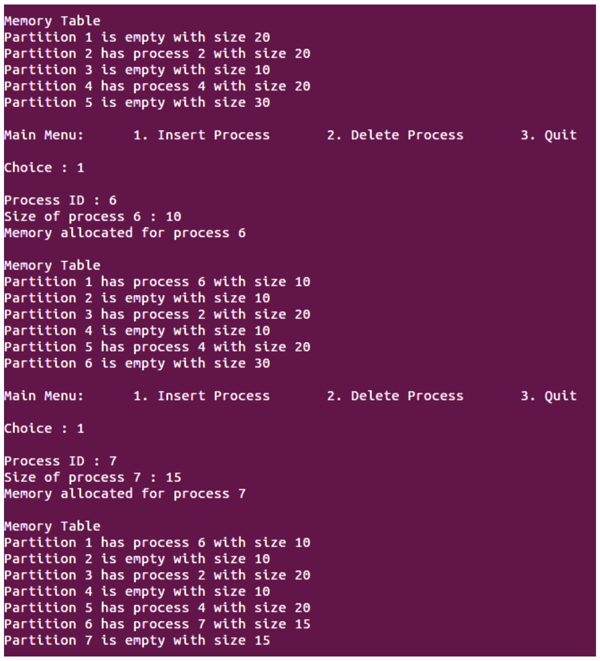
```
Main Menu:      1. Insert       2. Delete       3. Stop

Enter Choice: 1

Enter process number: 1
Enter process size: 5

Process 1 is at partition 4


Partition number        Assigned Process        Partition Used Mem        Partition Total Mem
        0                       -1                      0                       15
        1                       -1                      0                       5
        2                       -1                      0                       20
        3                       1                       5                       35
        4                       -1                      0                       25



Main Menu:      1. Insert       2. Delete       3. Stop

Enter Choice: 1

Enter process number: 2
Enter process size: 22

Process 2 is at partition 5


Partition number        Assigned Process        Partition Used Mem        Partition Total Mem
        0                       -1                      0                       15
        1                       -1                      0                       5
        2                       -1                      0                       20
        3                       1                       5                       35
        4                       2                       22                      25



Main Menu:      1. Insert       2. Delete       3. Stop

Enter Choice: 1

Enter process number: 3
Enter process size: 20

Process 3 is at partition 3


Partition number        Assigned Process        Partition Used Mem        Partition Total Mem
        0                       -1                      0                       15
        1                       -1                      0                       5
        2                       3                       20                      20
        3                       1                       5                       35
        4                       2                       22                      25
```

MVT - Best



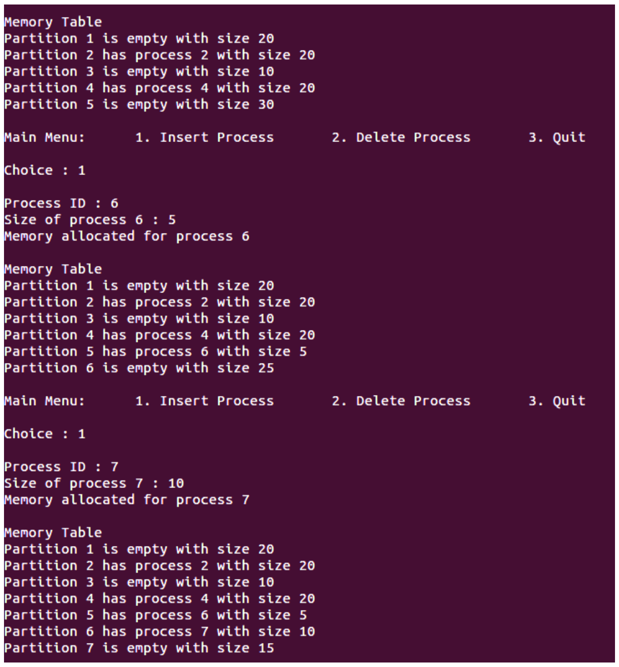
```
Memory Table
Partition 1 is empty with size 20
Partition 2 has process 2 with size 20
Partition 3 is empty with size 10
Partition 4 has process 4 with size 20
Partition 5 is empty with size 30

Main Menu:      1. Insert Process       2. Delete Process       3. Quit

Choice : 1

Process ID : 6
Size of process 6 : 10
Memory allocated for process 6

Memory Table
Partition 1 is empty with size 20
Partition 2 has process 2 with size 20
Partition 3 has process 6 with size 10
Partition 4 has process 4 with size 20
Partition 5 is empty with size 30

Main Menu:      1. Insert Process       2. Delete Process       3. Quit

Choice : 1

Process ID : 7
Size of process 7 : 15
Memory allocated for process 7

Memory Table
Partition 1 has process 7 with size 15
Partition 2 is empty with size 5
Partition 3 has process 2 with size 20
Partition 4 has process 6 with size 10
Partition 5 has process 4 with size 20
Partition 6 is empty with size 30
```

MVT First



```
Memory Table
Partition 1 is empty with size 20
Partition 2 has process 2 with size 20
Partition 3 is empty with size 10
Partition 4 has process 4 with size 20
Partition 5 is empty with size 30

Main Menu:      1. Insert Process       2. Delete Process       3. Quit

Choice : 1

Process ID : 6
Size of process 6 : 10
Memory allocated for process 6

Memory Table
Partition 1 has process 6 with size 10
Partition 2 is empty with size 10
Partition 3 has process 2 with size 20
Partition 4 is empty with size 10
Partition 5 has process 4 with size 20
Partition 6 is empty with size 30

Main Menu:      1. Insert Process       2. Delete Process       3. Quit

Choice : 1

Process ID : 7
Size of process 7 : 15
Memory allocated for process 7

Memory Table
Partition 1 has process 6 with size 10
Partition 2 is empty with size 10
Partition 3 has process 2 with size 20
Partition 4 is empty with size 10
Partition 5 has process 4 with size 20
Partition 6 has process 7 with size 15
Partition 7 is empty with size 15
```

MVT - Worst



```
Memory Table
Partition 1 is empty with size 20
Partition 2 has process 2 with size 20
Partition 3 is empty with size 10
Partition 4 has process 4 with size 20
Partition 5 is empty with size 30

Main Menu:      1. Insert Process       2. Delete Process       3. Quit

Choice : 1

Process ID : 6
Size of process 6 : 5
Memory allocated for process 6

Memory Table
Partition 1 is empty with size 20
Partition 2 has process 2 with size 20
Partition 3 is empty with size 10
Partition 4 has process 4 with size 20
Partition 5 has process 6 with size 5
Partition 6 is empty with size 25

Main Menu:      1. Insert Process       2. Delete Process       3. Quit

Choice : 1

Process ID : 7
Size of process 7 : 10
Memory allocated for process 7

Memory Table
Partition 1 is empty with size 20
Partition 2 has process 2 with size 20
Partition 3 is empty with size 10
Partition 4 has process 4 with size 20
Partition 5 has process 6 with size 5
Partition 6 has process 7 with size 10
Partition 7 is empty with size 15
```

Page Replacement

Introduction:

When a page fault occurs, the OS has to remove a page from the memory so that it can fit in another page in the memory. These page replacement algorithms are used in operating systems that support virtual memory management.

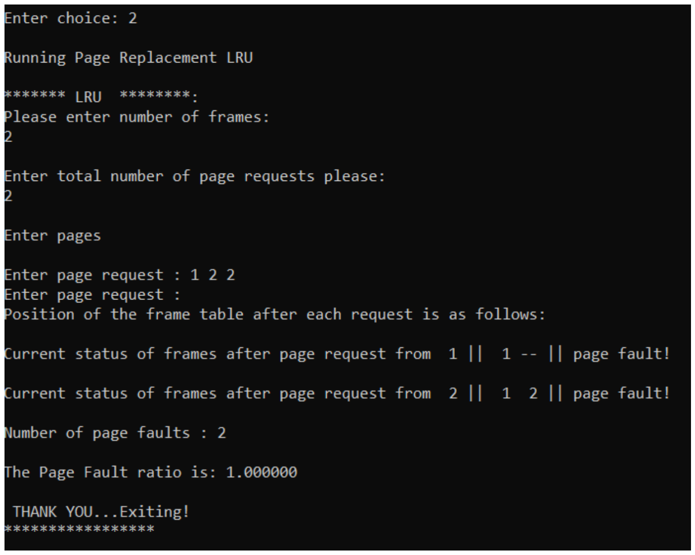
1]Least Recently Used

In Least Recently Used (LRU) algorithm is a Greedy algorithm where the page to be replaced is least recently used. The idea is based on locality of reference, the least recently used page is not likely

Algorithm:

1. Requests are handled sequentially hence we handle the current request.
2. If the page already exists in some frame then we continue else we allocate it to a free frame.
3. If no free frames exist then we replace it with the least recently used page in the frames.
4. Above steps are repeated until all requests are handled.

Output:



```
Enter choice: 2

Running Page Replacement LRU

******* LRU  ********:
Please enter number of frames:
2

Enter total number of page requests please:
2

Enter pages

Enter page request : 1 2 2
Enter page request :
Position of the frame table after each request is as follows:

Current status of frames after page request from  1 ||  1 -- || page fault!

Current status of frames after page request from  2 ||  1  2 || page fault!

Number of page faults : 2

The Page Fault ratio is: 1.000000

 THANK YOU...Exiting!
*****************
```

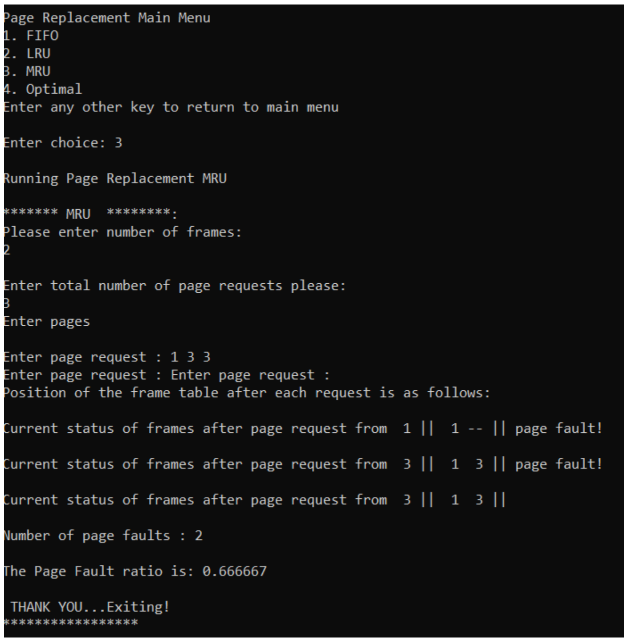
2]Most Recently Used

Most Recently Used (MRU) algorithm assumes that the page which was used most frequently will not be needed immediately so it will replace the most recently used page.

Algorithm:

1. Requests are handled sequentially hence we handle the current request.
2. If the page already exists in some frame then we continue else we allocate it to a free frame.
3. If no free frames exist then we replace it with the most recently used page in the frames.
4. Above steps are repeated until all requests are handled.

Output:



```
Page Replacement Main Menu
1. FIFO
2. LRU
3. MRU
4. Optimal
Enter any other key to return to main menu

Enter choice: 3

Running Page Replacement MRU

******* MRU  ********:
Please enter number of frames:
2

Enter total number of page requests please:
3
Enter pages

Enter page request : 1 3 3
Enter page request : Enter page request :
Position of the frame table after each request is as follows:

Current status of frames after page request from  1 ||  1 -- || page fault!

Current status of frames after page request from  3 ||  1  3 || page fault!

Current status of frames after page request from  3 ||  1  3 ||

Number of page faults : 2

The Page Fault ratio is: 0.666667

 THANK YOU...Exiting!
*****************
```

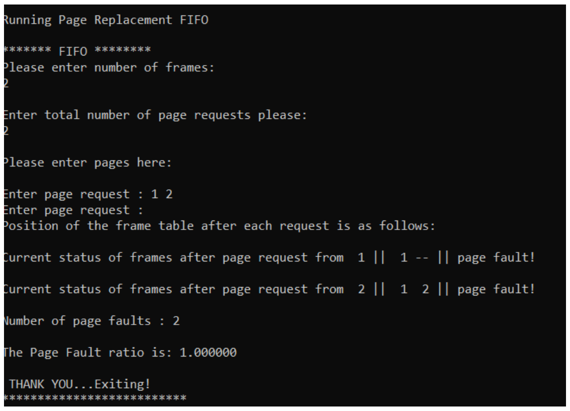
3]First in First out

FIFO Page Replacement technique is one of the simplest one to implement amongst other page replacement algorithms. It is a conservative algorithm.It is a low-overhead algorithm that maintains a queue to keep a track of all the pages in a memory.When a page needs to be replaced, the page at the FRONT of the Queue will be replaced. The FIFO page replacement technique is not implemented in operating systems nowadays.

Algorithm:

1. Requests are handled sequentially hence we handle the current request.
2. If the page already exists in some frame then we continue else we allocate it to a free frame.
3. If no free frames exist then we replace it with the oldest page in the frames.

Output:

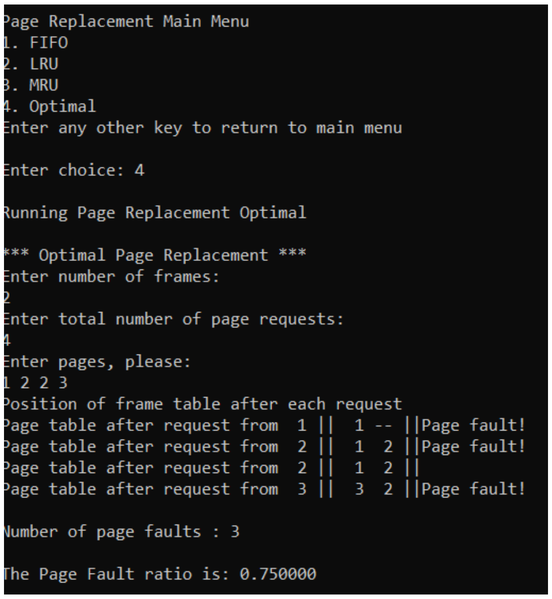


```
Running Page Replacement FIFO

******* FIFO ********
Please enter number of frames:
2

Enter total number of page requests please:
2

Please enter pages here:

Enter page request : 1 2
Enter page request :
Position of the frame table after each request is as follows:

Current status of frames after page request from  1 ||  1 -- || page fault!

Current status of frames after page request from  2 ||  1  2 || page fault!

Number of page faults : 2

The Page Fault ratio is: 1.000000

 THANK YOU...Exiting!
************************
```

4]Optimal Page Replacement Algorithm

In operating systems, whenever a new page is referred to and not present in memory, page fault occurs and the Operating System replaces one of the existing pages with a newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.In this algorithm, OS replaces the page that will not be used for the longest period of time in future. The given example demonstrates the following algorithm.

Output:



```
Page Replacement Main Menu
1. FIFO
2. LRU
3. MRU
4. Optimal
Enter any other key to return to main menu

Enter choice: 4

Running Page Replacement Optimal

*** Optimal Page Replacement ***
Enter number of frames:
2
Enter total number of page requests:
4
Enter pages, please:
1 2 2 3
Position of frame table after each request
Page table after request from  1 ||  1 -- ||Page fault!
Page table after request from  2 ||  1  2 ||Page fault!
Page table after request from  2 ||  1  2 ||
Page table after request from  3 ||  3  2 ||Page fault!

Number of page faults : 3

The Page Fault ratio is: 0.750000
```

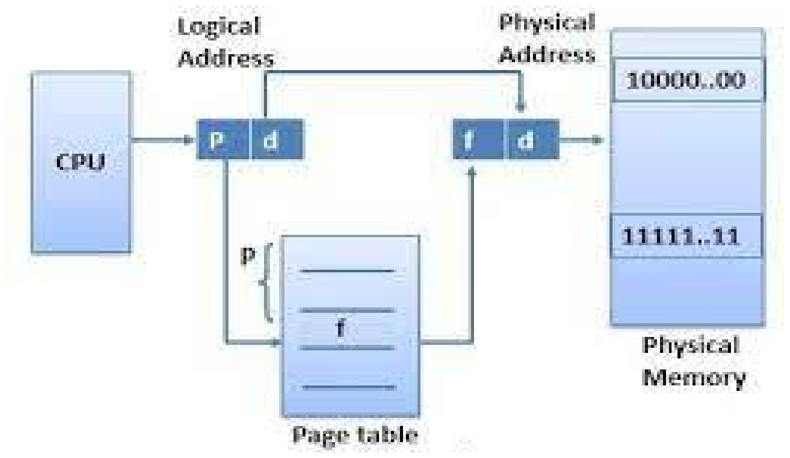
Paging

Introduction:

In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.

Normal Paging

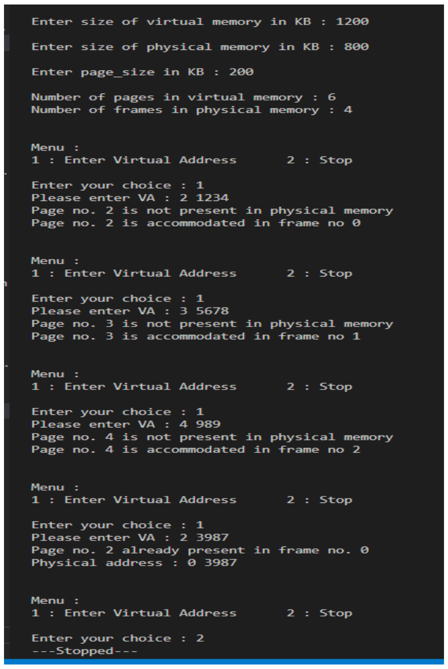
Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. Main memory is divided into frames. Size of the frame is the same as that of a page. Task is to bring the requested page from secondary memory into the main memory and allocate a frame. This scheme permits the physical address space of a process to be noncontiguous.



Implementation:

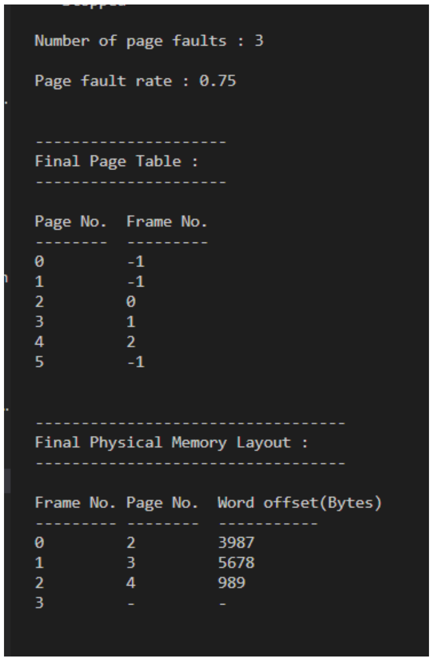
1. Page table stores the information of frames allocated to each page. If no frame is allocated to a page then frame number is stored as -1.
2. No. of pages = VM_Size / page_size, No. of frames = PM_Size / page_size
3. If a requested page is not present in the main memory, then it is brought into main memory and allocated an empty frame. It's logical address is converted to a physical address.
4. If the requested page is not present in the main memory and no frame is empty, then it is allocated a frame using some page replacement algorithm as discussed in the above module.
5. If a page is already present into the main memory, then its physical address is printed.
6. Offset of the requested page must be less than the permitted word offset. Page number request should not exceed the number of pages in Page Table.

Output:



```
Enter size of virtual memory in KB : 1200

Enter size of physical memory in KB : 800

Enter page_size in KB : 200

Number of pages in virtual memory : 6
Number of frames in physical memory : 4


Menu :
1 : Enter Virtual Address       2 : Stop

Enter your choice : 1
Please enter VA : 2 1234
Page no. 2 is not present in physical memory
Page no. 2 is accommodated in frame no 0


Menu :
1 : Enter Virtual Address       2 : Stop

Enter your choice : 1
Please enter VA : 3 5678
Page no. 3 is not present in physical memory
Page no. 3 is accommodated in frame no 1


Menu :
1 : Enter Virtual Address       2 : Stop

Enter your choice : 1
Please enter VA : 4 989
Page no. 4 is not present in physical memory
Page no. 4 is accommodated in frame no 2


Menu :
1 : Enter Virtual Address       2 : Stop

Enter your choice : 1
Please enter VA : 2 3987
Page no. 2 already present in frame no. 0
Physical address : 0 3987


Menu :
1 : Enter Virtual Address       2 : Stop

Enter your choice : 2
---Stopped---
```

Menu



```
Number of page faults : 3

Page fault rate : 0.75


---------------------
Final Page Table :
---------------------

Page No.  Frame No.
--------  ---------
0         -1
1         -1
2         0
3         1
4         2
5         -1


----------------------------------
Final Physical Memory Layout :
----------------------------------

Frame No. Page No.  Word offset(Bytes)
--------- --------  -----------
0         2         3987
1         3         5678
2         4         989
3         -         -
```

Final table outputs

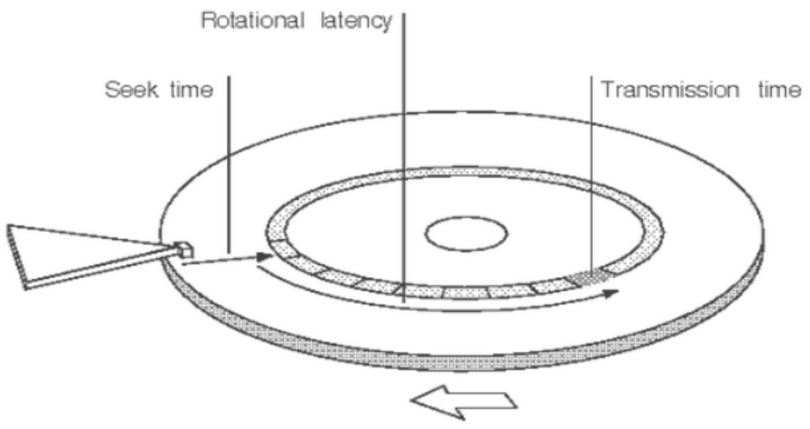
Disk Scheduling

Introduction:

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

1. Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
2. Two or more requests may be far from each other so can result in greater disk arm movement.
3. Hard drives are one of the slowest parts of computer systems and thus need to be accessed in an efficient manner.



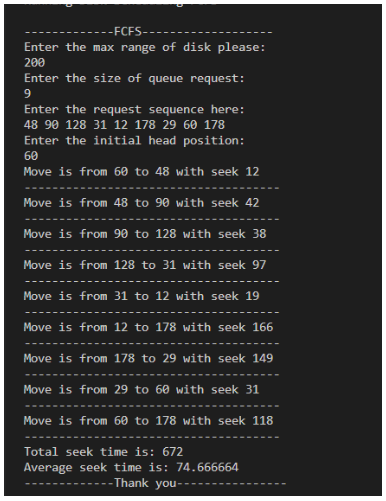
1]First Come First Serve:

FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

Algorithm:

1. Seek time = |curr_request - read_write_head|
2. Read_write head is updated to curr_request position.
3. Above steps are repeated until all requests are handled.

Output:



```
------------FCFS------------------
Enter the max range of disk please:
200
Enter the size of queue request:
9
Enter the request sequence here:
48 90 128 31 12 178 29 60 178
Enter the initial head position:
60
Move is from 60 to 48 with seek 12
-------------------------------------
Move is from 48 to 90 with seek 42
-------------------------------------
Move is from 90 to 128 with seek 38
-------------------------------------
Move is from 128 to 31 with seek 97
-------------------------------------
Move is from 31 to 12 with seek 19
-------------------------------------
Move is from 12 to 178 with seek 166
-------------------------------------
Move is from 178 to 29 with seek 149
-------------------------------------
Move is from 29 to 60 with seek 31
-------------------------------------
Move is from 60 to 178 with seek 118
-------------------------------------
Total seek time is: 672
Average seek time is: 74.666664
------------Thank you---------------
```

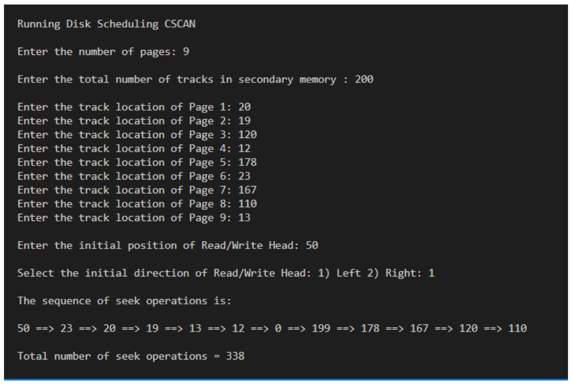
2]SCAN

In SCAN algorithm, the R/W head moves into a particular direction till the extreme end, satisfying all the requests in between, and then it reverses its direction again satisfying requests in its path. The working is similar to that of a traditional elevator hence it is also known as Elevator Algorithm.

Algorithm:

1. Initial read_write head position and direction are taken as input.
2. Next we populate left and right vectors with track requests less than and greater than head respectively.
3. Left and right arrays are sorted in increasing order of distance from read_write head.
4. First all the requests in the given direction are taken sequentially.
5. Once we reach the end of the disk in that direction then we reverse the direction.
6. Then again all the requests are handled sequentially in the given direction until.
7. Final output is the total seek time.

Output:



```
Running Disk Scheduling CSCAN

Enter the number of pages: 9

Enter the total number of tracks in secondary memory : 200

Enter the track location of Page 1: 20
Enter the track location of Page 2: 19
Enter the track location of Page 3: 120
Enter the track location of Page 4: 12
Enter the track location of Page 5: 178
Enter the track location of Page 6: 23
Enter the track location of Page 7: 167
Enter the track location of Page 8: 110
Enter the track location of Page 9: 13

Enter the initial position of Read/Write Head: 50

Select the initial direction of Read/Write Head: 1) Left 2) Right: 1

The sequence of seek operations is:

50 ==> 23 ==> 20 ==> 19 ==> 13 ==> 12 ==> 0 ==> 199 ==> 178 ==> 167 ==> 120 ==> 110

Total number of seek operations = 338
```

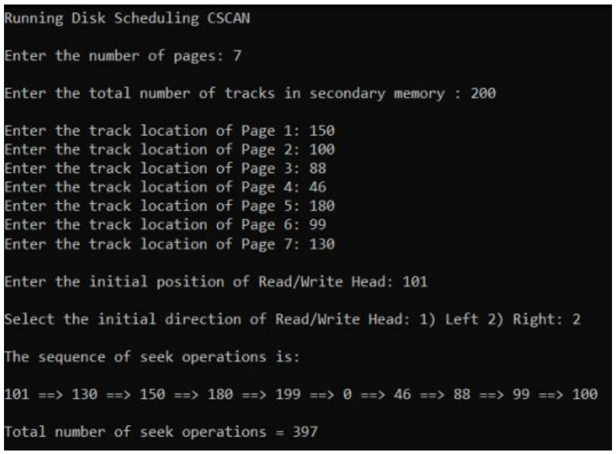
3] Circular SCAN (CSCAN)

C-SCAN is a modified SCAN algorithm that deals with the disadvantages of the SCAN algorithm by executing the requests more uniformly. C-SCAN moves the head from one end servicing all the requests to the other extreme end. Then it reverses its direction and reaches the initial position without servicing any requests in between and then it starts servicing the remaining services again.

Algorithm:

1. Initial read_write head position and direction are taken as input.
2. Next we populate left and right vectors with track requests less than and greater than head respectively.
3. Left and right arrays are sorted in increasing order of distance from read_write head.
4. First all the requests in the given direction are taken sequentially.
5. Once we reach the end of tracks in that direction then we reverse the direction.
6. Read_write head is now taken to the other end of the disk.
7. Then again all the requests are handled sequentially in the given direction until.
8. Final output is the total seek time.

Output:



```
Running Disk Scheduling CSCAN

Enter the number of pages: 7

Enter the total number of tracks in secondary memory : 200

Enter the track location of Page 1: 150
Enter the track location of Page 2: 100
Enter the track location of Page 3: 88
Enter the track location of Page 4: 46
Enter the track location of Page 5: 180
Enter the track location of Page 6: 99
Enter the track location of Page 7: 130

Enter the initial position of Read/Write Head: 101

Select the initial direction of Read/Write Head: 1) Left 2) Right: 2

The sequence of seek operations is:

101 ==> 130 ==> 150 ==> 180 ==> 199 ==> 0 ==> 46 ==> 88 ==> 99 ==> 100

Total number of seek operations = 397
```

4]LOOK

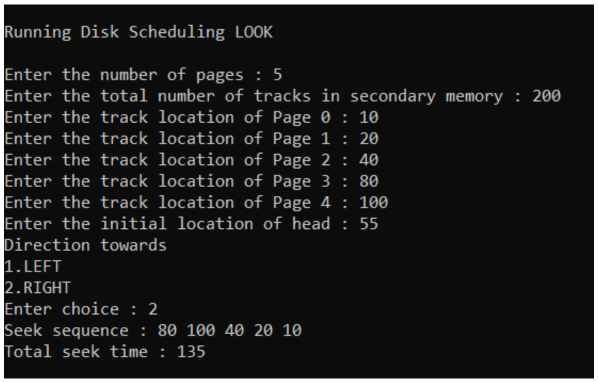
LOOK is the advanced version of SCAN (elevator) disk scheduling algorithm which gives slightly better seek time than any other algorithm. The LOOK algorithm services request similarly as SCAN algorithm meanwhile it also looks ahead as if there are more tracks that are needed to be serviced in the same direction. If there are no pending requests in the moving direction the head reverses the direction and starts servicing requests in the opposite direction.

Algorithm:

1. Sort the pages based on their track location.

2. If the track direction is left, find the first track location which is greater than the initial track location. If the track direction is right, find the first track location which is greater than or equal to the initial track location.
3. Find the seek time which is the absolute difference between the next track location (depending on the track direction) and initial track location.
4. Once we service all the requests in this direction, the direction is reversed and remaining requests are serviced in the opposite direction.

Output:



```
Running Disk Scheduling LOOK

Enter the number of pages : 5
Enter the total number of tracks in secondary memory : 200
Enter the track location of Page 0 : 10
Enter the track location of Page 1 : 20
Enter the track location of Page 2 : 40
Enter the track location of Page 3 : 80
Enter the track location of Page 4 : 100
Enter the initial location of head : 55
Direction towards
1.LEFT
2.RIGHT
Enter choice : 2
Seek sequence : 80 100 40 20 10
Total seek time : 135
```

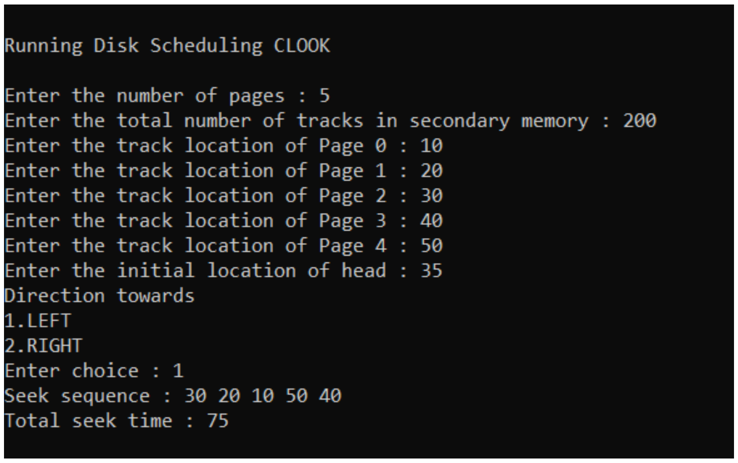
5]Circular LOOK (CLOOK)

C-LOOK is an enhanced version of both SCAN as well as LOOK disk scheduling algorithms. This algorithm also uses the idea of wrapping the tracks as a circular cylinder as C-SCAN algorithm but the seek time is better than the C-SCAN algorithm. In this algorithm, the head services requests only in one direction(either left or right) until all the requests in this direction are not serviced and then jumps back to the farthest request on the other direction and service the remaining requests which gives a better uniform servicing as well as avoids wasting seek time for going till the end of the disk.

Algorithm:

1. Sort the pages based on their track location.
2. If the track direction is left, find the first track location which is greater than the initial track location. If the track direction is right, find the first track location which is greater than or equal to the initial track location.
3. Find the seek time which is the absolute difference between the next track location (depending on the track direction) and initial track location.
4. Once we service all the requests in this direction, the farthest request in the other direction should be serviced next. Again in the same direction, find the seek time from this location and continue until all requests are serviced.
5. Final output is the total seek time i.e., the summation of all the seek times.

Output:



```
Running Disk Scheduling CLOOK

Enter the number of pages : 5
Enter the total number of tracks in secondary memory : 200
Enter the track location of Page 0 : 10
Enter the track location of Page 1 : 20
Enter the track location of Page 2 : 30
Enter the track location of Page 3 : 40
Enter the track location of Page 4 : 50
Enter the initial location of head : 35
Direction towards
1.LEFT
2.RIGHT
Enter choice : 1
Seek sequence : 30 20 10 50 40
Total seek time : 75
```

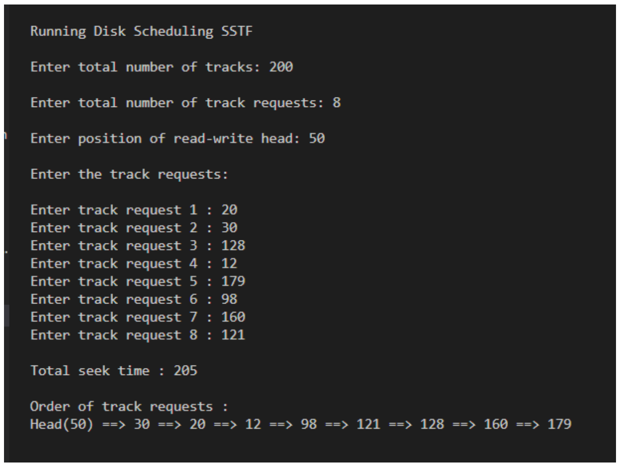
6]Shortest Seek Time First (SSTF)

In SSTF (Shortest Seek Time First), requests having the shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of the system.

Algorithm:

1. Find the distance of all track requests from the read-write head.
2. Pick the request with the shortest distance from the read-write head.
3. If two requests have the same shortest distance but on either side of head, then select the request in the direction in which the read-write head was moving.
4. Update the read-write head and seek time.
5. Repeat above steps until all requests are handled.
6. Final output is the total seek time.

Output:



```
Running Disk Scheduling SSTF

Enter total number of tracks: 200

Enter total number of track requests: 8

Enter position of read-write head: 50

Enter the track requests:

Enter track request 1 : 20
Enter track request 2 : 30
Enter track request 3 : 128
Enter track request 4 : 12
Enter track request 5 : 179
Enter track request 6 : 98
Enter track request 7 : 160
Enter track request 8 : 121

Total seek time : 205

Order of track requests :
Head(50) ==> 30 ==> 20 ==> 12 ==> 98 ==> 121 ==> 128 ==> 160 ==> 179
```

Novelty

As an attempt to add novelty to our mini-project, we have implemented maximum synchronisation problems without the use of external libraries like ***p thread*** and without the use of "semaphores" and at the same time making sure that the logic of the program remains correct and precise. This can be considered as a novelty as most of the solutions on the net use "***semaphores***" to code these following problems, but we have tried implementing them in a slightly different many, making them easier to understand and code! Also, extra care has been taken while implementing the algorithms (The algorithms have been coded from scratch, without using much of the standard library functions) which makes them uniquely coded! Thus, as a part of our project, we have tried to come up with new approaches for the problems that already existed.

Conclusion

OS algorithms have been implemented which involve solutions to some common problems. The entire mini project was a great learning experience for each one of us and has not only helped us in making our basics stronger but has also given us a chance to practically simulate everything that we have learnt so far as part of the course, "***Operating Systems***"!

Github Link for main combined code:

https://github.com/AnirudhAchal/OS-Simulator

Thank you!