G.S. Mandal's

Maharashtra Institute of Technology, Aurangabad

Department of Computer Science and Engineering

LAB MANUAL

CSE 322: Operating System

(2019-20 Part 1)

Maharashtra Institute of Technology, Aurangabad
NH-211, MIT Campus, Satara Village Road, Aurangabad- 431 010 (M.S.); India.
Phone: (0240) 2375222; Fax: (0240) 2376618, E-mail: principalmitt@mit.asia
Website: www.btech.mit.asia

Department of Computer Science and Engineering

Vision

To develop the department as a center of excellence in the field of computer science and engineering by imparting knowledge & training to the students for meeting growing needs of the industry & society.

Mission

Providing quality education through a well-designed curriculum in tune with the challenging needs of software industry by providing state of the art facilities and to impart knowledge in the thrust areas of computer science and engineering.

Department of Computer Science and Engineering

Program Educational Objectives

PEO1: To prepare the students to achieve success in Computing Domain to create individual careers, innovations or to work as a key contributor to the private or Government sector and society.

PEO2: To develop the ability among the students to understand Computing and mathematical fundamentals and apply the principles of Computer Science for analyzing, designing and testing software for solving problems.

PEO3: To empower the students with ability to quickly reflect the changes in the new technologies in the area of computer software, hardware, networking and database management.

PEO4: To promote the students with awareness for lifelong learning, introduce them to professional practice, ethics and code of professionalism to remain continuous in their profession and leaders in technological society.

Program Specific Objectives

PSO1: Identify appropriate data structures and algorithms for a given contextual problem and develop programs to design and implement applications.

PSO3: Design and manage the large databases and develop their own databases to solve real world problems and to design, build, manage networks and apply wireless techniques in mobile based applications.

PSO3: Design a variety of computer-based components and systems using computer hardware, system software, systems integration process and use standard testing tools for assuring the software quality.

Program Outcomes

PO1: Apply knowledge of mathematics, science, and engineering fundamentals to solve problems in Computer science and Engineering.

PO2: Identify, formulate and analyze complex problems.

PO3: Design system components or processes to meet the desired needs within realistic constraints for the public health and safety, cultural, societal and environmental considerations.

PO4: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data for valid conclusions.

PO5: Select and apply modern engineering tools to solve the complex engineering problem.

PO6: Apply knowledge to assess contemporary issues.

PO7: Understand the impact of engineering solutions in a global, economic, environmental, and societal context.

PO8: Apply ethical principles and commit to professional ethics and responsibilities.

PO9: Work effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.

PO10: Communicate effectively in both verbal and written form.

PO11: Demonstrate knowledge and apply engineering and management principles to manage projects and in multi-disciplinary environment.

PO12: To engage in life-long learning to adopt to the technological changes.

Department of Computer Science and Engineering

Course: CSE 301: Operating System

Course Outcomes:

After completing the course students will be able to

CO1: To understand the role of OS and Learn UNIX Commands

CO2: To analyze and Compare various algorithms used for memory management, CPU scheduling, File Handling and I/O Operations.

CO3: To apply various concepts related to deadlock to solve problems related to resource allocation.

CO4: Analyze role of process synchronization in increasing system performance.

CO5: Apply various process scheduling algorithms.

CO6: To use various memory allocation techniques.

Mapping

Experiment No.	Blooms Level	Mapping To CO	Mapping To PO
1	Apply	2	1
2	Apply	1	1
3	Apply	5	2
4	Apply	5	2
5	Apply	5	2
6	Apply	5	2
7	Apply	3	1,2
8	Apply	3	1,2
9	Apply	6	1,3
10	Apply	6	1,3

QUEST FOR EXCELLENCE	**MAHARASHTRA INSTITUTE OF TECHNOLOGY AURANGABAD**	**LABORATORY MANUAL**
	PRACTICAL EXPERIMENT INSTRUCTION SHEET	
	DEPARTMENT: COMPUTER SCIENCE AND ENGINEERING	

LABORATORY : A.Y.: 2019-20		
Class: TY CSE	PART: I	SUBJECT: Operating System

LIST OF EXPERIMENT

EXPERIMENT NO.	EXPERIMENT DESCRIPTION	Co Mapping
1	To perform various primitive operations of Files: Open a file, Read text from a file, Append to a file, Write Text into a file, Copy contents of one file to other.	2
2	To study and implement various basic Linux System Calls Fork(), Exec(), Delay(),Kill(),Alarm()	1
3	To study and implement First Come First Serve(FCFS) process scheduling algorithm	5
4	To study and implement shortest job first(SJF) process scheduling algorithm.	5
5	To study and implement Round Robin(RR) process scheduling algorithm. Use Quantum very low and Very high and compare results	5
6	To study and implement Priority Scheduling Algorithm.	5
7	To study and implement Deadlock Detection Algorithm.	3
8	To study and implement Banker’s Algorithm.	3
9	To study and implement memory allocation technique: First Fit Algorithm, Best Fit Algorithm, Worst Fit Algorithm	6
10	To study and implement page replacement policy: First In First Out(FIFO)	6

PREPARED BY : Mr. K.R. Khandarkar	APPROVED BY : HCSED

Experiment No. 1

Aim:- To perform various primitive operations of Files: Open a file, Read text from a file, Append to a file, Write Text into a file, Copy contents of one file to other.

Theory:-

To get the input we may read to specify the file or device to use as the source and the address and length of the memory buffer into which the input should be read.

The general method should be used to pass the parameter in register. The arguments that we pass on to main() at the command prompt are called command line arguments. The function main( ) can have two arguments, traditional named as argc and argv is a array of pointers to strings and argc is an int whose value is equal to the number f strings to which arg v points. When the program is executed, the string on the command line are passed to main(). More precisely we pass, the string but the command line are stored in argv[1] and so on. The argument argc is set the number of strings given on the command line, For example in the program, if at the command prompt we give

```
$ gcc file.c
$ ./a.out   abc1.txt      abc2.txt
```

Then

Argc contains 3

Argv[0] would contain base of “file”

Argv[1] would contain base of “abc1.txt”

Argv[2] would contain base of “abc2.txt”

Whenever   we pass arguments to main(), it is a good habit whether the correct number of arguments have been passed on the main() or not, In our program this has been done through

```
if(arg c! = 3)

{

printf (“improper number of arguments “);

exit();

}
```

1] There is no need to compile the program every time we want to use this utility. It can be executed at command prompt.

2] We are able to pass source file name and target file name to main() and utility them in main().

One final comment…..The while loop that we have used in our program can be written in a more compact from as shown below:

```
While (ch=f get c (fs) != EOF)

F put   (ch, ft);
```

This avoid the usage of indefinite loop and a break statement to come out of this loop. Here, first fget c (fs) gets the character from the file assign it to the variable ch, and the ch is compared against EOF. Remember that is necessary to put the expression

Ch=fget(fs)

Within a pair of parentheses, so that the character reads is assigned to variable ch and then it is compared with EOF . There is one more way of writing the while loop, It is shown below:

```
While ( ! feof (fs) )

{

Ch=fgetc (fs);

Fput c(ch,ft);

}
```

Here, feof() is macro which returns a 0 if end if file in not reahed, Hence the ! operator to negate this 0 to the truth value. When the end of file is reached frof() returns non-zero value, 1! Operator makes it 0 and since the while loop gets terminated.

Note that in each of one of them the following three method for opening a file are same , since in each one of them essentially a base of the string is being passed to fopen()

```
Fs=fopen (“abc1.c,”r”)

Fs=fopen (“file.exe,”r”)

Fs=fopen (“arg[1], ”r”);
```

Logic of the Program:-

```
if ( argc != 3 )

{

puts ( "Improper number of arguments" ) ;
exit( ) ;
}
fs = fopen ( argv[1], "r" ) ;
if ( fs == NULL )
{
puts ( "Cannot open source file" ) ;
exit( ) ;
}
ft = fopen ( argv[2], "w" ) ;
if ( ft == NULL )
{
puts ( "Cannot open target file" ) ;
fclose ( fs ) ;
exit( ) ;
}
```

Expected Output

First file.c file

```
$ gcc file.c
$ ./a.out   abc1.txt      abc2.txt
File copied
```

Conclusion:-

Hence we have implemented a program to read and write data from one file and copy it into another file using command line arguments.

Experiment No. 2

Aim:- To study and implement various basic Linux System Calls Fork(), Exec(), Delay(),Kill(),Alarm()

Theory:-

System Calls:

A system call is a service request made by the kernel by a program. Generally speaking, the service is something that only the kernel has the right to do, like I / O. Programmers don't usually have to think about system calls because there are functions in the GNU C library to do nearly all that system calls do. By making machine calls themselves, these functions work. For instance, there is a system call that changes a file's permissions, but you don't have to learn about it because you can only use the chmod feature of the GNU C Library. System calls are sometimes called kernel calls.

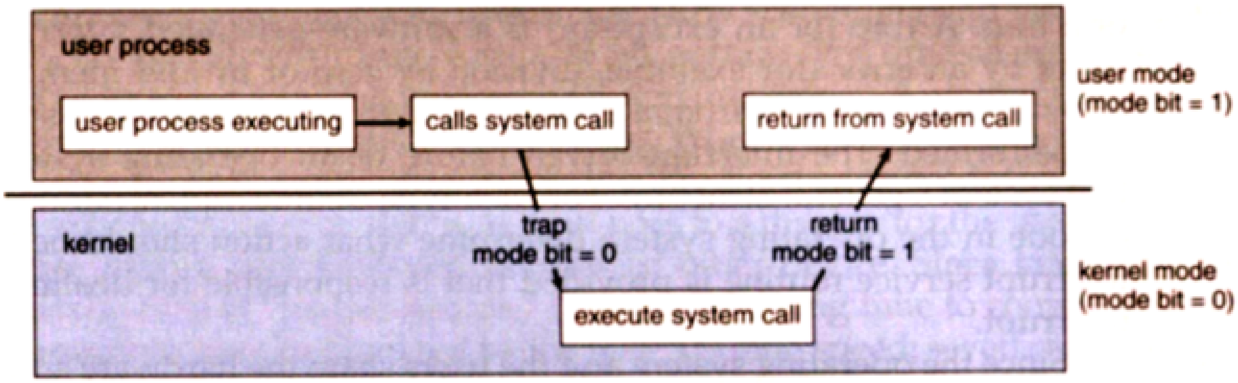


Fig. 2.1 System call Execution

The machine call provides the operating system resources with an interface. Application developers often do not have direct access to device calls, but can use an application programming interface (API) to access them. The functions in the API invoke the actual system calls. Several advantages can be achieved by using the API: portability: as long as an API is supported by a framework, any application that uses the API will compile and run.

Ease of use: it can be considerably easier to use the API than to use the actual program call.

There are 5 different categories of system calls: process control, file manipulation, device manipulation, information maintenance and communication.

Steps for “fork” system call demo:

STEP 1: Start the program.

STEP 2: Declare pid as integer.

STEP 3: Create the process using Fork command.

STEP 4: Check pid is less than 0 then print error else if pid is equal to 0 then execute command else parent process wait for child process.

STEP 5: Stop the program.

SAMPLE OUTPUT:

```
$ gcc pc.c
$ a.out
parent process $ child process
$ps
 PID   CLS   PRI    TTY   TIME         COMD
5913   TS     70     pts   022 0:00     ksh
```

Conclusion:-

Thus the program was executed and verified successfully for various basic Linux System Calls.

Experiment No.3

Aim: To study and implement First Come First Serve(FCFS) process scheduling algorithm.

Theory :

CPU scheduling is a process that allows one process to use the CPU while the execution of another process is on hold due to the unavailability of any resource such as I / O and so on, making full use of the CPU. CPU scheduling is intended to make the system effective, fast and reasonable.

When the CPU is idle, one of the processes in the ready queue to be run must be selected by the operating system. The short-term scheduler (or CPU scheduler) conducts the selection process. The scheduler selects and allocates the CPU to one of the processes in memory that are ready to execute.

The Dispatcher is another factor involved in the scheduling role of the CPU. The dispatcher is the module that enables the CPU to be managed by the short-term scheduler selected.

This function involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program from where it left last time.

Because it is called during every process turn, the dispatcher should be as fast as possible. The time taken by the dispatcher is known as the Dispatch Latency to interrupt one process and resume another process. Use the figure below to illustrate the dispatch latency.

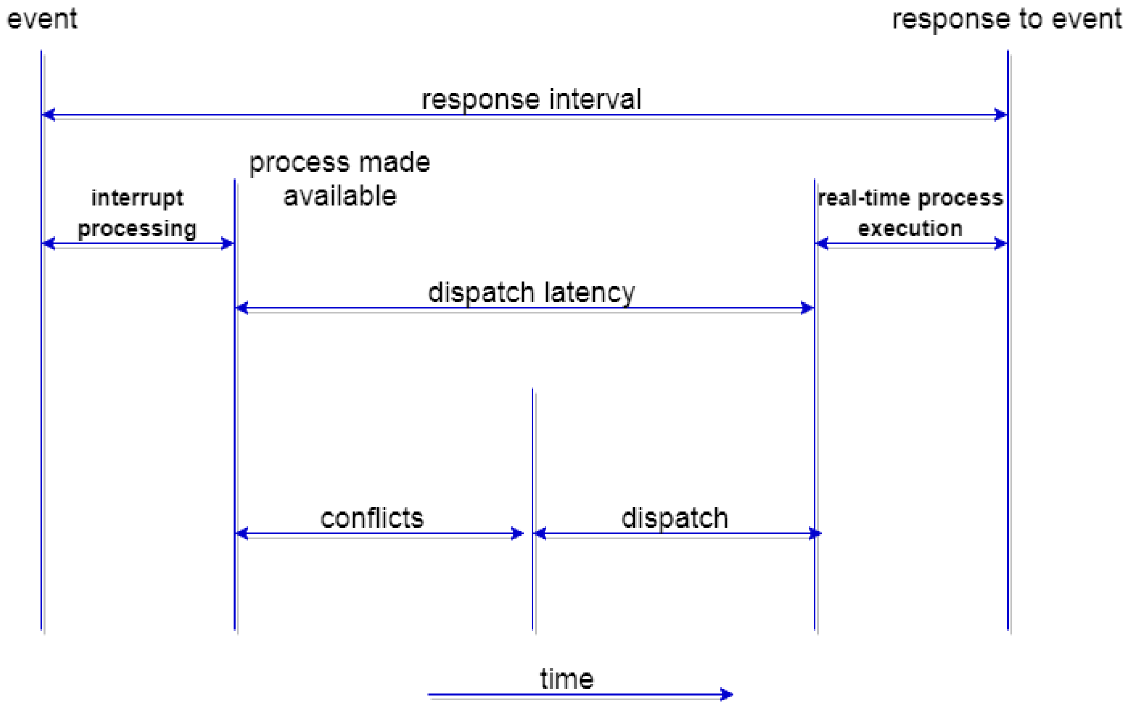


Fig. 3.1 dispatch latency

CPU scheduling decisions will take place under the following four circumstances:

1. when a process moves from the running state to the waiting state (for I / O requests or requests to wait for one of the child processes to be terminated).
2. When a process switches from the operating state to the ready state (e.g. when there is an interrupt).

3. When a phase transitions from the waiting state to the ready state (e.g. I / O completion).

4. When a cycle is completed.

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process(if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3.When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **non-preemptive**; otherwise the scheduling scheme is **preemptive**.

Non-Preemptive Scheduling:
Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.
It is the only method that can be used on certain hardware platforms, because It does not require the special hardware (for example: a timer) needed for preemptive scheduling.
Preemptive Scheduling:
In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

First Come First Serve Scheduling: In the scheduling algorithm "First come first serve," as the name suggests, the process that comes first is executed first, or we can tell the process that first asks the CPU gets the CPU allocated first.
First Come First Serve is like FIFO(First in First out) Queue data structure, where the data item that is first added to the queue is the one that first leaves the queue. For Batch Systems, this is used. Using a queue data structure, where a new process enters through the queue's tail, it is easy to understand and execute programmatically and the scheduler selects process from the queue's head. Purchasing tickets at the ticket counter is a perfect example of FCFS scheduling in real life.

Calculating Average Waiting Time:
Average waiting time is a key parameter for any scheduling algorithm to assess its performance.
The average waiting time of the processes in the queue is AWT or Average waiting time, waiting for the scheduler to select them for execution. Lower the Average Waiting Time, better the scheduling algorithm.
Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with **Arrival Time** 0, and given **Burst Time**, let's find the average waiting time using the FCFS scheduling algorithm.

Process Number	Burst time (ms)
P1	21
P2	3
P3	6
P4	2

The average waiting time = (0+21+24+30) / 4 = 18.75 ms.

P1	P2	P3	P4

0 21 24 30 32

GANNT chart for above process.

Here we have simple formulae for calculating various times for given processes:

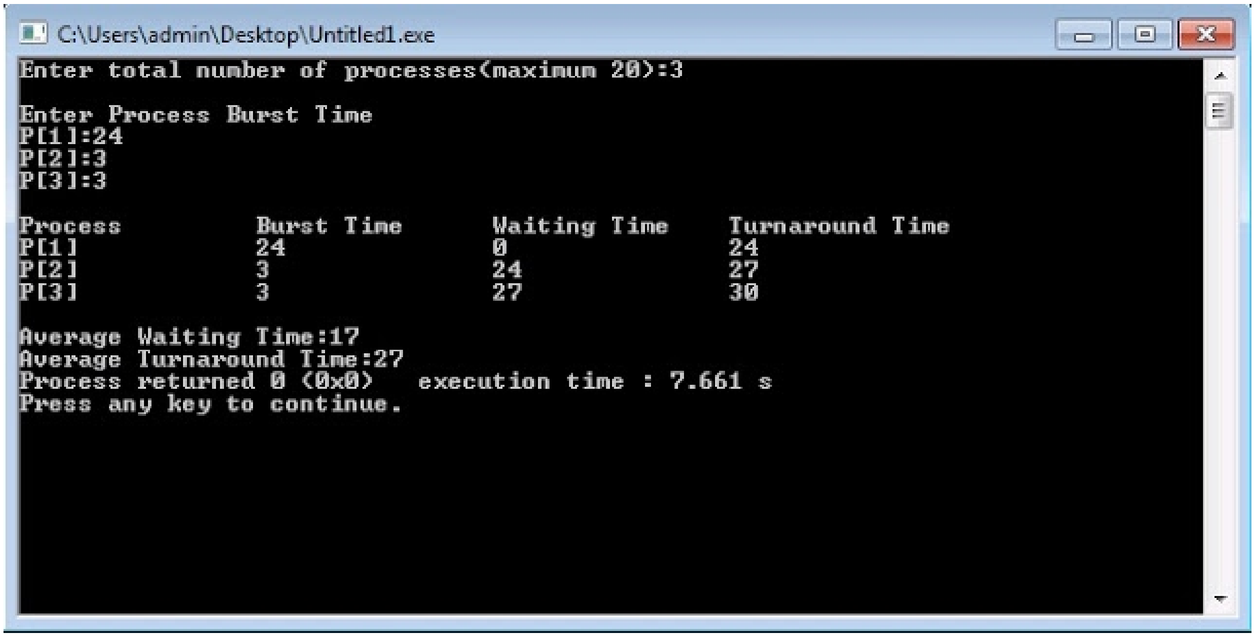
Completion Time: Time taken for the execution to complete, starting from arrival time.

Turn Around Time: Time taken to complete after arrival. In simple words, it is the difference between the Completion time and the Arrival time.

Waiting Time: Total time the process has to wait before it's execution begins. It is the difference between the Turn Around time and the Burst time of the process.

For the program above, we have considered the **arrival time** to be 0 for all the processes, try to implement a program with variable arrival times.

Expected Output:



```
Enter total number of processes(maximum 20):3

Enter Process Burst Time
P[1]:24
P[2]:3
P[3]:3

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            24              0               24
P[2]            3               24              27
P[3]            3               27              30

Average Waiting Time:17
Average Turnaround Time:27
Process returned 0 (0x0)   execution time : 7.661 s
Press any key to continue.
```

Conclusion: Thus we have studied and implemented FCFS process scheduling algorithm.

Experiment No.4

Aim : To study and implement shortest job first(SJF) process scheduling algorithm.

Theory :

- Shortest Job First scheduling operates first on the process with the shortest time of burst or length. This is the best approach to minimize waiting time.
- This is used in Batch Systems.
- It is of two types:
  1. Non Pre-emptive
  2. Pre-emptive
- To order to implement it effectively, the processor should be aware of the burst time / duration time of the processes to advance, which is technically not always feasible.
- If all the jobs / processes are available at the same time, this scheduling algorithm is optimal. (Either time of arrival is 0 for everyone, or time of arrival is the same for everyone)

Consider the below processes available in the ready queue for execution, with **arrival time** as 0 for all and given **burst times**.

Process Number	Burst time (ms)
P1	21
P2	3
P3	6
P4	2

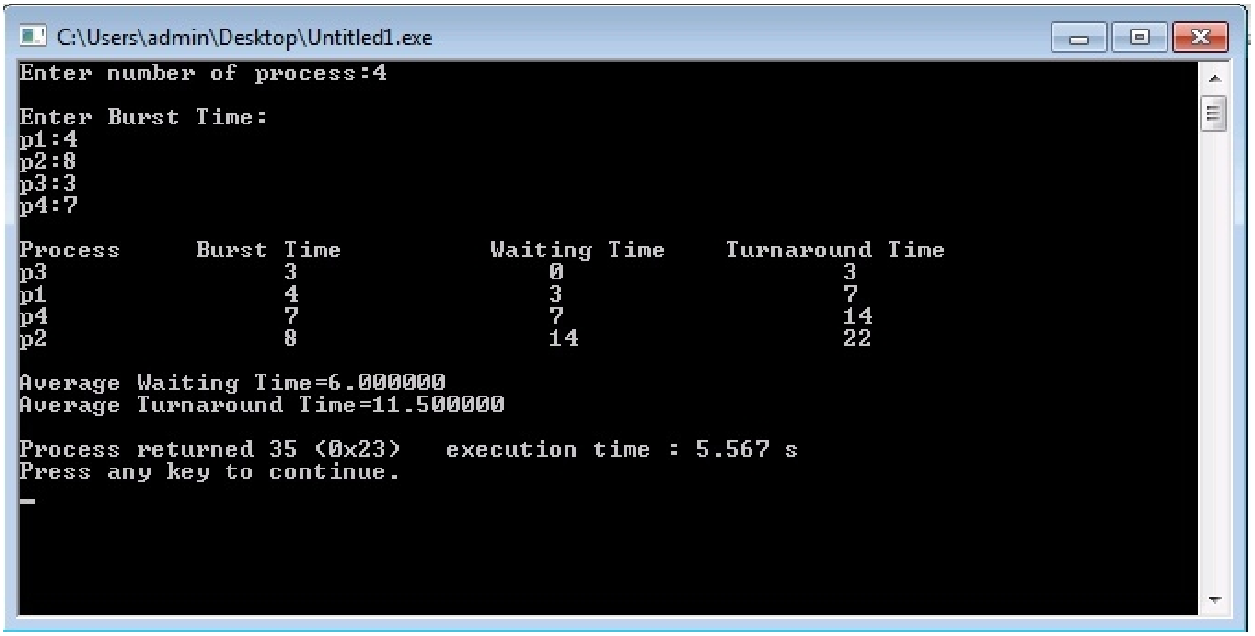
The average waiting time = (0+2+5+11) / 4 = 4.5 ms.

P4	P2	P3	P1

0 2 5 11 32

GANNT chart for above process.

Expected sample output:



```
Enter number of process:4

Enter Burst Time:
p1:4
p2:8
p3:3
p4:7

Process     Burst Time          Waiting Time    Turnaround Time
p3                3                   0                   3
p1                4                   3                   7
p4                7                   7                   14
p2                8                   14                  22

Average Waiting Time=6.000000
Average Turnaround Time=11.500000

Process returned 35 (0x23)   execution time : 5.567 s
Press any key to continue.
```

Conclusion: Thus we have studied and implemented FCFS process scheduling algorithm.

Experiment No.5

Aim : To study and implement Round Robin(RR) process scheduling algorithm. Use Quantum very low and Very high and compare results.

Theory:

Round Robin is a CPU scheduling algorithm in which a fixed time slot is cyclically allocated to each operation. It is quick, easy to implement and free from hunger as all processes receive a fair share of the CPU. One of the most widely used CPU scheduling strategies as a heart. It is preventive because processes are allocated to the CPU for at most only a fixed slice of time.
The downside of this is that background switching is more overhead.

Example: Consider following Three processes with their arrival time,

Process Number	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is 1 unit,

P1	P2	P3	P1	P2	P3	P1	P2	P3	P2

0 10

P1 waiting time = 4

P2 waiting time = 6

P3 waiting time = 6

1. Completion Time: Time at which process completes its execution.
2. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
3. Waiting Time(W.T): Time Difference between turnaround time and burst time. Waiting Time = Turn Around Time – Burst Time

Steps to find waiting times of all processes:

```
1- Create an array rem_bt[] to keep track of remaining
   burst time of processes. This array is initially a
   copy of bt[] (burst times array)
2- Create another array wt[] to store waiting times
   of processes. Initialize this array as 0.
3- Initialize time : t = 0
4- Keep traversing the all processes while all processes
   are not done. Do following for i'th process if it is
   not done yet.
    a- If rem_bt[i] > quantum
       (i)  t = t + quantum
       (ii) bt_rem[i] -= quantum;
    c- Else // Last cycle for this process
       (i)  t = t + bt_rem[i];
       (ii) wt[i] = t - bt[i]
       (ii) bt_rem[i] = 0;
```

Expected Sample Output:

```
Processes  Burst time  Waiting time  Turn around time

 1              10          13            23

 2               5          10            15

 3               8          13            21


Average waiting time = 12

Average turnaround time = 19.6667
```

Conclusion: Thus we have studied and implemented Round robin Process scheduling algorithm.

Experiment No. 6

Aim: To study and implement Priority Scheduling Algorithm.

Theory: The priority of a process is typically the opposite of the CPU burst time in the Shortest Job First scheduling algorithm, i.e. the higher the burst time the lower is the priority of that process.

In the case of priority scheduling, the priority is not always set as the reverse of the CPU burst time, but it can be set internally or externally, but yes the scheduling is done on the basis of the process's priority where the most urgent process is handled first, followed by the lesser priority processes in order.

FCFS performs operations with the same priority. The priority of process, when internally defined, can be decided based on **memory requirements**, **time limits** ,**number of open files**, **ratio of I/O burst to CPU burst** etc.

Whereas, external priorities are set based on criteria outside the operating system, like the importance of the process, funds paid for the computer resource use, market factor etc.

Priority Scheduling can be of two types,

Preemptive Priority Scheduling: if the new process arrived at the ready queue has a higher priority than the current process, the CPU is preempted, which means the processing of the current process is stopped and the higher priority incoming new process is allocated to the CPU for execution.

Non-Preemptive Priority Scheduling: In the case of a non-preemptive priority scheduling algorithm, if a new process arrives with a higher priority than the current running process, the incoming process is positioned at the head of the ready queue, indicating that it will be processed after the running of the current process.

Consider the below table fo processes with their respective CPU burst times and the priorities.

Process Number	Burst time (ms)	Priority
P1	21	2
P2	3	1
P3	6	4
P4	2	3

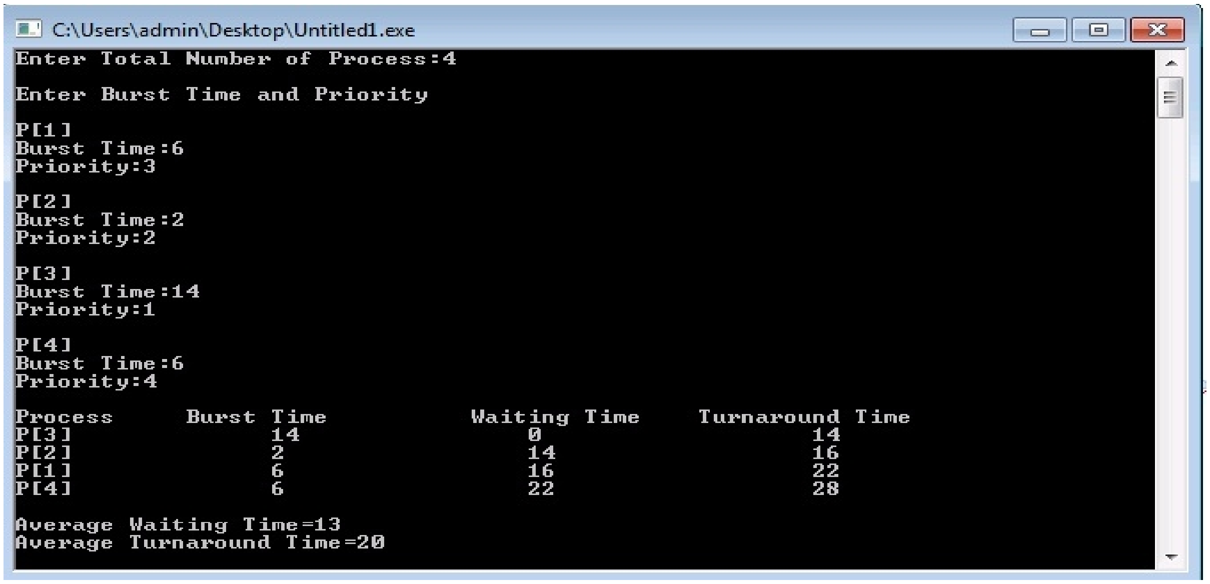
P2	P1	P4	P1

0 3 24 26 32

GANNT chart for above process.

The average waiting time = (0+3+24+26) / 4 = 13.25 ms.

Expected Sample output:



```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:6
Priority:3

P[2]
Burst Time:2
Priority:2

P[3]
Burst Time:14
Priority:1

P[4]
Burst Time:6
Priority:4

Process     Burst Time          Waiting Time    Turnaround Time
P[3]                14                  0                       14
P[2]                2                   14                      16
P[1]                6                   16                      22
P[4]                6                   22                      28

Average Waiting Time=13
Average Turnaround Time=20
```

Conclusion : Thus we have studied and implemented priority scheduling algorithm.

Experiment No 7.

Aim : To study and implement Deadlock Detection Algorithm.

Theory:

Deadlock is a condition where a series of processes are delayed because each process retains a resource and is waiting for another resource that some other process has acquired.

Imagine an example when two trains come to each other on the same track and there is only one track, once they are in front of each other, none of the trains can pass. Similar situation exists in operating systems when there are two or more processes that have some resources and are waiting for other resources. For example, Process 1 holds Resource 1 in the diagram below and is waiting for Resource 2 obtained via Process 2, and Process 2 is waiting for Resource 1.

Deadlock can arise if four conditions hold simultaneously:

- Mutual exclusion: only one process at a time can use a resource
- Hold and wait: holding at least one resource and is waiting to acquire additional resources held by others
- No preemption: a resource can be released only voluntarily by the process holding it.
- Circular wait: there exists a set {P0, P1, …, Pn} of waiting processes such that:

P0 is waiting for a resource that is held by P1,

P1 is waiting for a resource that is held by P2, …,

Pn–1 is waiting for a resource that is held by Pn, and

Pn is waiting for a resource that is held by P0.

Deadlock Prevention:

- Restrain the ways request can be made
- Hold and Wait –Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.
- No Preemption – If a process holding some resources requests another resource that cannot be immediately allocated to it, all resources currently being held are released. Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
- Circular Wait – impose a total ordering of all resource types, and require that each process requests resources in an *increasing order* of enumeration.

Deadlock Avoidance:

Requires that the system has a priori information available.

- Simplest model requires that each process declare the maximum number of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.

-

Deadlock Detection Algorithm in Operating System:

If a system does not employ either a deadlock prevention or deadlock avoidance algorithm then a deadlock situation may occur. In this case,

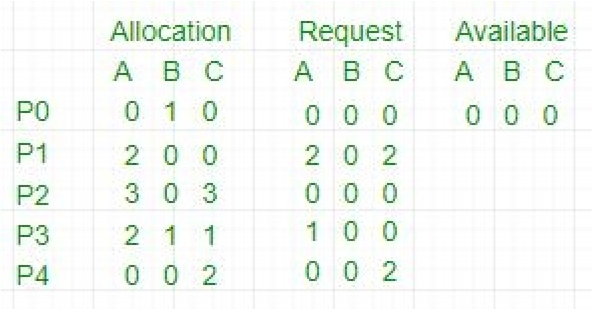
- Apply an algorithm to examine state of system to determine whether deadlock has has occurred or not.
- Apply an algorithm to recover from the deadlock.

Deadlock Detection Algorithm :

The algorithm employs several time varying data structures:

- **Available-** A vector of length m indicates the number of available resources of each type.
- **Allocation-** An n*m matrix defines the number of resources of each type currently allocated to a process. Column represents resource and resource represent process.
- **Request-** An n*m matrix indicates the current request of each process. If request[i][j] equals k then process P_i is requesting k more instances of resource type R_j.

We treat rows in the matrices Allocation and Request as vectors, we refer them as Allocation_i and Request_i.

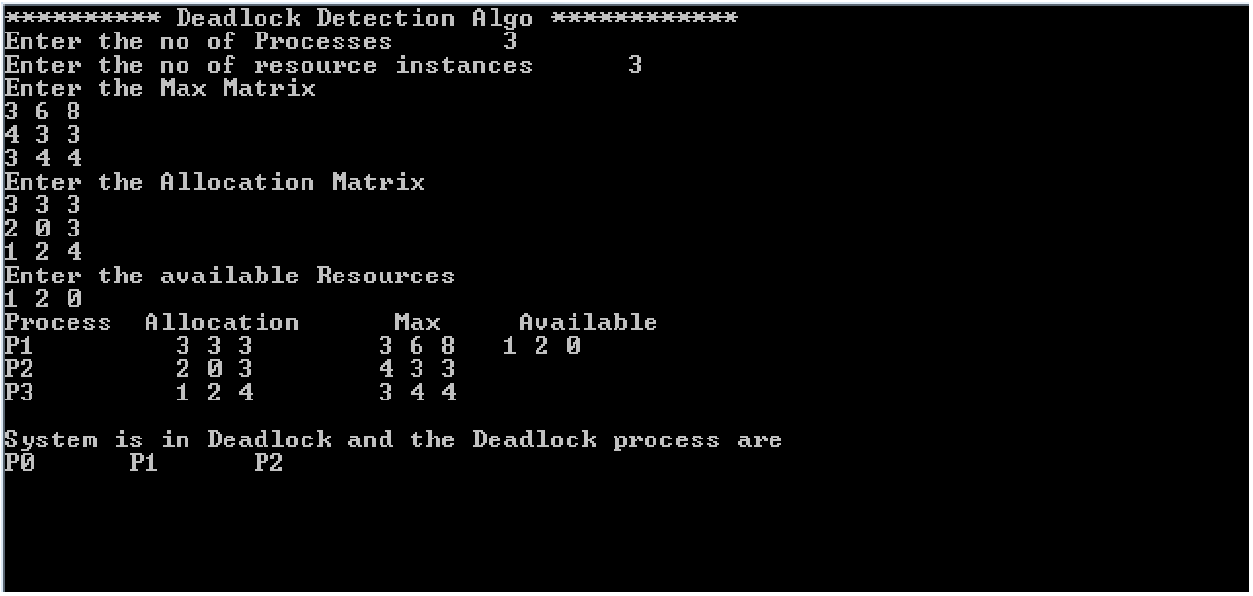


	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

1. n this, Work = [0, 0, 0] &
   Finish = [false, false, false, false, false]
2. *i=0* is selected as both Finish[0] = false and [0, 0, 0]<=[0, 0, 0].
3. Work =[0, 0, 0]+[0, 1, 0] =>[0, 1, 0] &
   Finish = [true, false, false, false, false].
4. *i=2* is selected as both Finish[2] = false and [0, 0, 0]<=[0, 1, 0].
5. Work =[0, 1, 0]+[3, 0, 3] =>[3, 1, 3] &
   Finish = [true, false, true, false, false].

6. *i=1* is selected as both Finish[1] = false and [2, 0, 2]<=[3, 1, 3].
7. Work =[3, 1, 3]+[2, 0, 2] =>[5, 1, 5] &
   Finish = [true, true, true, false, false].
8. *i=3* is selected as both Finish[3] = false and [1, 0, 0]<=[5, 1, 5].
9. Work =[5, 1, 5]+[2, 1, 1] =>[7, 2, 6] &
   Finish = [true, true, true, true, false].
10. *i=4* is selected as both Finish[4] = false and [0, 0, 2]<=[7, 2, 6].
11. Work =[7, 2, 6]+[0, 0, 2] =>[7, 2, 8] &
    Finish = [true, true, true, true, true].
12. Since Finish is a vector of all true it means **there is no deadlock** in this example.

Expected Sample output:



```
********** Deadlock Detection Algo ************
Enter the no of Processes       3
Enter the no of resource instances      3
Enter the Max Matrix
3 6 8
4 3 3
3 4 4
Enter the Allocation Matrix
3 3 3
2 0 3
1 2 4
Enter the available Resources
1 2 0
Process  Allocation      Max     Available
P1         3 3 3        3 6 8   1 2 0
P2         2 0 3        4 3 3
P3         1 2 4        3 4 4

System is in Deadlock and the Deadlock process are
P0      P1      P2
```

Conclusion: Thus, we have studied and implemented deadlock detection algorithm.

Experiment No 8.

Aim : To study and implement Banker’s Algorithm.

Theory:

Banker's algorithm is an algorithm of impasse avoidance. It is called so because in banking systems, this algorithm is used to assess whether or not a loan can be issued. Remember that in a bank there are n account holders and the sum of money in all their accounts is S. Each time the bank has to issue a loan, it subtracts the amount of the loan from the bank's total money. Then it will test whether that disparity is greater than S. It's done because, only then, even if all the account holders draw all their money at once, the bank would have enough money. Banker's algorithm operates in computers similarly.

Banker's algorithm is an allocation of resources and an algorithm for the prevention of deadlock. This safety algorithm test simulates the allocation for fixed maximum possible quantities of all resources, then conducts a "s-state" check to monitor for potential activities before determining whether to allow the allocation to proceed.
Simply put, it tests if allocation of any resource would result in a deadlock or not, or whether it is possible to assign a resource to a process and if not, resource will not be allocated to that process.
Determining a stable sequence (even if there is only 1) should ensure that the program is not blocked.
Banker’s algorithm is generally used to find if a safe sequence exist or not. But here we will determine the total number of safe sequences and print all safe sequences.
The data structure used are,

- Available vector
- Max Matrix
- Allocation Matrix
- Need Matrix

Resource Request Algorithm:

This describes the behavior of the system when a process makes a resource request in the form of a request matrix.

The steps are:

1. If number of requested instances of each resource is less than the need (which was declared previously by the process), go to step 2.
2. If number of requested instances of each resource type is less than the available resources of each type, go to step 3. If not, the process has to wait because sufficient resources are not available yet.
3. Now, assume that the resources have been allocated. Accordingly do,

Available = Available – Request i

Allocation(i) = Allocation(i) + Request(i)

Need(i) = Need(i) - Request(i)

This step is done because the system needs to assume that resources have been allocated. So there will be fewer resources available after allocation. The number of allocated instances will increase. The need of the resources by the process will reduce. That's what is represented by the above three operations. After completing the above three steps, check if the system is in safe state by applying the safety algorithm. If it is in safe state, proceed to allocate the requested resources. Else, the process has to wait longer.

Safety Algorithm:

1. Let Work and Finish be vectors of length m and n, respectively. Initially,

Work = Available

Finish[i] =false for i = 0, 1, ... , n - 1.

This means, initially, no process has finished and the number of available resources is represented by the Available array.

2. Find an index i such that both

Finish[i] ==false

Needi <= Work

If there is no such i present, then proceed to step 4.

It means, we need to find an unfinished process whose need can be satisfied by the available resources. If no such process exists, just go to step 4.

3. Perform the following:

Work = Work + Allocation;

Finish[i] = true;

Go to step 2.

When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

4. If Finish[i] == true for all i, then the system is in a safe state.

That means if all processes are finished, then the system is in safe state.

Example:

Resource number	R1	R2	R3
Instances available	10	5	7

Process	Allocation			Max		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3
P2	2	0	0	3	2	2
P3	3	0	2	9	0	2
P4	2	1	1	2	2	2

Explanation:

Total resources are R1 = 10, R2 = 5, R3 = 7 and allocated resources are R1 = (0+2+3+2 =) 7, R2 = (1+0+0+1 =) 2, R3 = (0+0+2+1 =) 3. Therefore, remaining resources are R1 = (10 – 7 =) 3, R2 = (5 – 2 =) 3, R3 = (7 – 3 =) 4.

Remaining available = Total resources – allocated resources

and

Remaining need = max – allocated

So, we can start from either P2 or P4. We can not satisfy remaining need from available resources of either P1 or P3 in first or second attempt step of Banker's algorithm. There are only four possible safe sequences.

These are:

P2–> P4–> P1–> P3

P2–> P4–> P3–> P1

P4–> P2–> P1–> P3

P4–> P2–> P3–> P1

Conclusion: Thus we have studied and implemented Banker's algorithm.

Experiment No 9

Aim: To study and implement memory allocation technique: First Fit Algorithm, Best Fit Algorithm, and Worst Fit Algorithm.

Theory :

The allocation policy that always allocates from the smallest suitable free block. Suitable allocation mechanisms include sequential fit searching for a perfect fit, first fit on a size-ordered free block chain, segregated fits, and indexed fits. Many good fit allocators are also described as best fit. In theory, best fit may exhibit bad fragmentation, but in practice this is not commonly observed.

First fit simply searches the free list from the beginning, and uses the first free block large enough to satisfy the request. If the block is larger than necessary, it is split and the remainder is put on the free list.

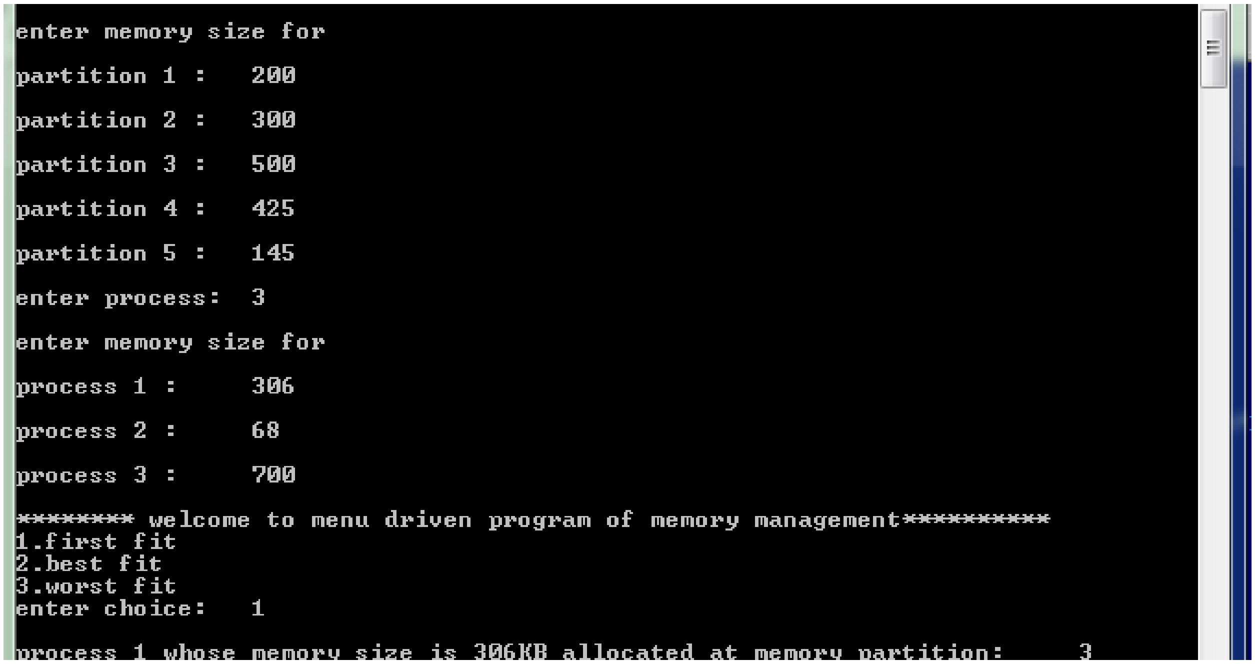
Logic of the Program:-

```
Take one array to store total number of partition:
Say arrr[]
Store element in arr[]
Sort it
Ask for  the Requirement of Memory
Store it as choice if choice is less than arr[i]
  if(arr[i]>=ch)
  {
   printf("\nBest Fit=%d",arr[i]);
   printf("\nFirst Fit=%d",arr[i+1]);
   printf("\nWorst Fit=%d",arr[n-1]);
   x=1;

   break;

  }
  }
  if(x==0)
{
printf("\nMemory Requirement is Large...");
}
 getch();
}
```

Sample Output:



```
enter memory size for

partition 1 :   200

partition 2 :   300

partition 3 :   500

partition 4 :   425

partition 5 :   145

enter process:  3

enter memory size for

process 1 :     306

process 2 :     68

process 3 :     700

********* welcome to menu driven program of memory management***********
1.first fit
2.best fit
3.worst fit
enter choice:   1

process 1 whose memory size is 306KB allocated at memory partition:     3
```

Conclusion: Hence we have studied and implemented memory management algorithms.

Experiment no. 10

Aim: To study and implement page replacement policy: First In First Out(FIFO).

Theory:

A page replacement algorithm is needed in an operating system that uses paging for memory management to assess which page needs to be replaced when a new page is entered.
Page Fault–A page fault occurs when a running program accesses a memory page mapped into the space of the virtual address, but not loaded into the physical memory.
Because the actual physical memory is much smaller than the virtual memory, there are page faults. Operating System may need to replace one of the current pages with the newly needed page in the event of a page fault. Different algorithms for page replacement suggest different ways of selecting which page to replace. The goal is to reduce the number of page faults for all algorithms.

Page Replacement Algorithms:

First In First Out (FIFO) : This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Steps for Program:
1. Start the process

2. Declare the size with respect to page length

3. Check the need of replacement from the page to memory

4. Check the need of replacement from old page to new page in memory

5. Form a queue to hold all pages

6. Insert the page require memory into the queue

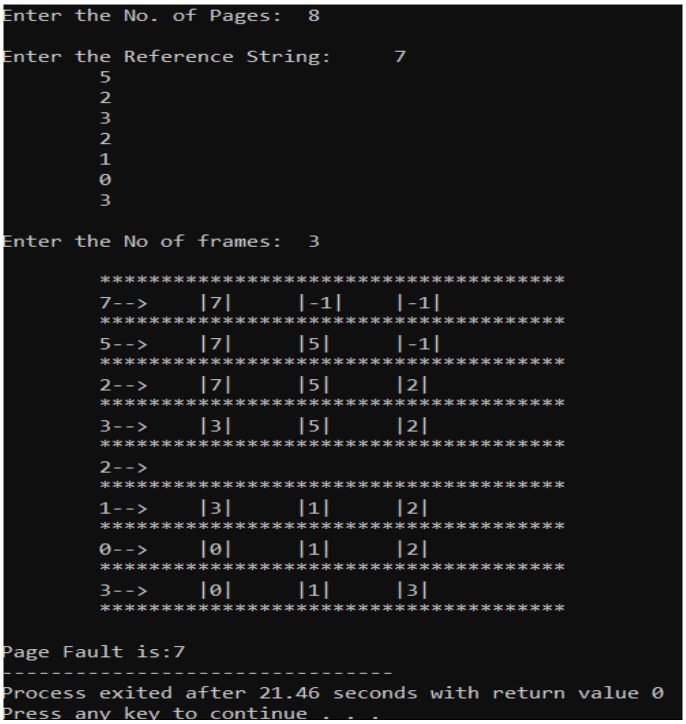
7. Check for bad replacement and page fault

8. Get the number of processes to be inserted

9. Display the values

10. Stop the process

Sample Expected Output:



```
Enter the No. of Pages:  8

Enter the Reference String:     7
        5
        2
        3
        2
        1
        0
        3

Enter the No of frames:  3

        **************************************
        7-->    |7|     |-1|    |-1|
        **************************************
        5-->    |7|     |5|     |-1|
        **************************************
        2-->    |7|     |5|     |2|
        **************************************
        3-->    |3|     |5|     |2|
        **************************************
        2-->
        **************************************
        1-->    |3|     |1|     |2|
        **************************************
        0-->    |0|     |1|     |2|
        **************************************
        3-->    |0|     |1|     |3|
        **************************************

Page Fault is:7
--------------------------------
Process exited after 21.46 seconds with return value 0
Press any key to continue . . .
```

Conclusion : Thus we have studied and implemented FIFO page replacement algorithm.