SRM INSTITUTE OF SCIENCE & TECHNOLOGY FACULTY OF SCIENCE AND HUMANITIES

Department of Computer Science and Applications (B.Sc. Computer Science)

PRACTICAL RECORD

NAME :

REGISTER NO :

COURSE : B.Sc. - Computer Science

SEMESTER / YEAR : IV / II

SUBJECT CODE : USA23402J

SUBJECT NAME : Operating System

APRIL 2025

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
Ramapuram, Chennai
FACULTY OF SCIENCE AND HUMANITIES

Department of Computer Science & Applications (B.Sc. - Computer Science)

REGISTER NUMBER:

BONAFIDE CERTIFICATE

This is to certify that the bonafide work done by _____in the subject **OPERATING SYSTEM [USA23402J]** at SRM Institute of Science and Technology, Ramapuram, Chennai in April 2025.

STAFF IN CHARGE **HEAD OF THE DEPARTMENT**

Submitted for the University Practical Examination held at SRM Institute of Science and Technology, Ramapuram, Chennai on_____.

INTERNAL EXAMINER 1 **INTERNAL EXAMINER 2**

INDEX

Ex.No:1 **Name:**

Date : **Reg.No:**

COMPARISON BETWEEN VARIOUS OPERATING SYSTEMS

Different Operating Systems

1.MS-DOS:

MS-DOS which is short for Microsoft Disk Operating System is a non-graphical command line operating system developed for IBM compatible computers with x86 microprocessor. The operating system used a command line interface for the user to input commands to navigate, open and manipulate files on their computer.

Features:

- It is a single user operating system meaning only one user can operate at a time.
- It is a light weight operating system allowing users to have direct access to the BIOS and its underlying hardware.
- Loads data and programs from external sources and bring them into the internal memory so they can be used on the computer.
- Enables the computer to perform input and output operations such as taking commands from keyboard, printing information on the screen.
- It is very helpful in making file management like creating, editing, deleting files, etc.
- It also controls and manages other external devices such as the printer, keyboard or external hard drive using various drive utilities.

Drawbacks:

- It does not allow multiple users to operate on the system.
- It does not support graphical interface hence mouse cannot be used to operate it.
- It does not support multiprogramming meaning it can only have one process in the ram.
- It lacked memory protection which meant no security, and less stability.
- It has difficulty in memory access when addressing more than 640 MB of RAM.

2.WindowsOperatingSystem:

Windows is an operating system designed by Microsoft to be used on a standard x86 Intel and AMD processors. It provides an interface, known as a graphical user interface(GUI) which eliminates the need to memorize commands for the command line by using a mouse to navigate through menus, dialog boxes, buttons, tabs, and icons. The operating system was named windows since the programs are displayed in the shape of a square. This Windows operating system has been designed for both a novice user just using at home as well as for professionals who are into development.

Features:

- It is designed to run on any standard x86 Intel and AMD hence most of

the hardware vendors make drivers for windows like Dell, HP, etc.
- It supports enhanced performance by utilizing multi-core processors.

It comes preloaded with many productivity tools which helps to complete all types of everyday tasks on your computer.

- Windows has a very large user base so there is a much larger selection of available software programs, utilities.
- Windows is backward compatible meaning old programs can run on newer versions.
- Hardware is automatically detected eliminating need of manually installing any device drivers.

Drawbacks:

- Windows can be expensive since the OS is paid license and majority of its applications are paid products.
- Windows has high computer resource requirement like it should have high ram capacity, a lot of hard drive space and good graphics card.
- Windows slows and hangs up if the user loads up many programs at the same time.
- Windows includes network sharing that can be useful if user has a network with many PCs.
- Windows is vulnerable to virus attacks since it has a huge user base and users have to update OS to keep up-to-date with security patches.

3.LINUX:

The Linux OS is an open source operating system project that is a freely distributed, cross- platform operating system developed based on UNIX. This operating system is developed by Linus Torvalds. The name Linux comes from the Linux kernel. It is basically the system software on a computer that allows apps and users to perform some specific task on the computer. The development of Linux operating system pioneered the open source development and became the symbol of software collaboration.

Features:

- Linux is free can be downloaded from the Internet or redistribute it under GNU licenses and has the best community support.
- Linux OS is easily portable which means it can be installed on various types of devices like mobile, tablet computers.
- It is a multi-user, multitasking operating system.
- BASH is the Linux interpreter program which can be used to execute commands.
- Linux provides multiple levels of file structures i.e. hierarchical structure in which all the files required by the system and those that are created by the user are arranged.
- Linux provides user security using authentication features and also threat detection and solution is very fast because Linux is mainly community driven.

Drawbacks:

- There's no standard edition of Linux hence confusing for users and also

becoming familiar with the Linux may be a problem for new users.

- More difficult to find applications to support user needs since Linux does not dominate the market.
- Since some applications are developed specifically for Windows and Mac, those might not be compatible with linux and sometimes users might not have much of a choice to choose between different applications like in Windows or Mac since most apps are developed for operating systems that have a huge user base.
- Some hardware may not be incompatible with Linux since it has patchier support for drivers which may result in malfunction.
- There are plenty of forums to resolve Linux issues, but it may not always match the user's own level of technical understanding.
-

4.SolarisOperatingSystem:

Solaris or SunOS is the name of the Sun company's Unix variant operating system that was originally developed for its family of Scalable Processor Architecture-based processors (SPARC) as well as for Intel-based processors. The UNIX workstation market had been largely dominated by this operating system during its time. As the Internet grew Sun's Solaris systems became the most widely installed servers for Web sites. Oracle purchased Sun and later renamed to Oracle Solaris.

Features:

- Solaris is known for its scalability. It can handle a large workload and still delivers indisputable performance advantages for database, Web, and Java technology-based services.
- Solaris systems were known to their availability meaning that these operating systems hardly crashes at anytime and because of its internet networking oriented design and broad scope of features it makes the job of adding new features or fixing any problems easy.
- It is built for network computing as it provides optimized network stack and support for advanced network computing protocols that delivers high-performance networking to most applications.
- Solaris has advanced, unique security capabilities which includes some of the world's most advanced security features, such as user rights management, cryptographic Framework and secure by default networking that allows users to safely deliver new solutions.
- Provides tools to enable seamless interoperability, test new software and efficiently consolidate application workloads.

Drawbacks:

- Solaris is quite expensive since it's an enterprise operating system. Also, Solaris doesn't provide updates for free.
- Solaris lacks a good graphical user interface support and is not user friendly.
- Hardware support is not nearly as good as many other operating systems.
- Performance would degrade considerably since Solaris cannot make use of different hardware that efficiently.
- Solaris sometimes becomes unstable and crashes due to total consumption of CPU and memory.

Result:

Based on comparative studies of various operating systems, the result highlights that the choice of an OS depends on its intended application, with Windows excelling in user-friendliness, Linux offering flexibility and performance, and macOS prioritizing seamless integration and design.

Ex.No:2 **Name:**

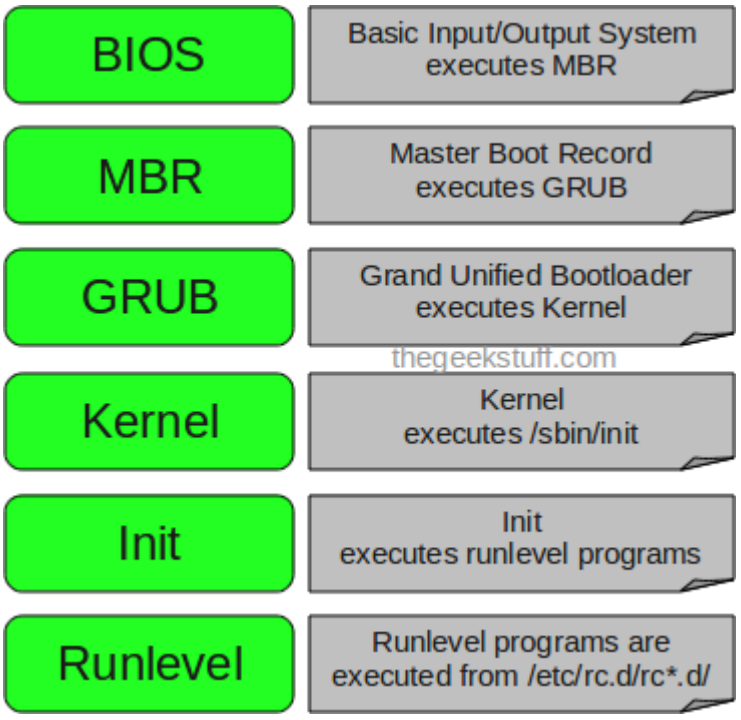
Date : **Reg.No:**

BOOTING PROCESS IN LINUX OS

SIX Stages of Linux Boot Process (Startup Sequence)

When you press the power button, a fascinating sequence unfolds: the system's firmware (BIOS or UEFI) initializes hardware, checks for issues, and locates the bootloader. The bootloader then loads the Linux kernel into memory, which initializes drivers and system processes. Finally, the init system (like systemd) sets up user space, launching services and bringing you to the login prompt—all in a matter of seconds! It's a symphony of software and hardware coordination.

The following are the 6 high level stages of a typical Linux boot process.



1. BIOS

- BIOS stands for Basic Input/Output System
- Performs some system integrity checks
- Searches, loads, and executes the boot loader program.
- It looks for boot loader in floppy, cd-rom, or hard drive. You can press a key (typically F12 of F2, but it depends on your system) during the BIOS startup to change the boot sequence.
- Once the boot loader program is detected and loaded into the memory, BIOS gives the control to it.
- So, in simple terms BIOS loads and executes the MBR boot loader.

2. MBR

- MBR stands for Master Boot Record.
- It is located in the 1st sector of the bootable disk. Typically /dev/hda, or /dev/sda

MBR is less than 512 bytes in size. This has three components 1) primary boot loader info in 1st 446 bytes 2) partition table info in next 64 bytes 3) mbr validation check in last 2 bytes.

- It contains information about GRUB (or LILO in old systems).
- So, in simple terms MBR loads and executes the GRUB boot loader.

3. GRUB

- GRUB stands for Grand Unified Bootloader.
- If you have multiple kernel images installed on your system, you can choose which one to be executed.
- GRUB displays a splash screen, waits for few seconds, if you don't enter anything, it loads the default kernel image as specified in the grub configuration file.
- GRUB has the knowledge of the filesystem (the older Linux loader LILO didn't understand filesystem).
- Grub configuration file is /boot/grub/grub.conf (/etc/grub.conf is a link to this). The following is sample grub.conf of CentOS.

```
#boot=/dev/sda

default=0

timeout=5

splashimage=(hd0,0)/boot/grub/splash.xpm.gz

hiddenmenu

title CentOS (2.6.18-194.el5PAE)

root (hd0,0)

kernel  /boot/vmlinuz-2.6.18-194.el5PAE  ro  root=LABEL=/

initrd  /boot/initrd-2.6.18-194.el5PAE.img
```

- As you notice from the above info, it contains kernel and initrd image.
- So, in simple terms GRUB just loads and executes Kernel and initrd images.

4. Kernel

- Mounts the root file system as specified in the "root=" in grub.conf
- Kernel executes the /sbin/init program
- Since init was the 1st program to be executed by Linux Kernel, it has the process id (PID) of 1. Do a 'ps -ef | grep init' and check the pid.
- initrd stands for Initial RAM Disk.

initrd is used by kernel as temporary root file system until kernel is booted and the real root file system is mounted. It also contains necessary drivers compiled inside, which helps it to access the hard drive partitions, and other hardware.

5. Init

- Looks at the /etc/inittab file to decide the Linux run level.
- Following are the available run levels
  - 0 – halt
  - 1 – Single user mode
  - 2 – Multiuser, without NFS
  - 3 – Full multiuser mode
  - 4 – unused
  - 5 – X11
  - 6 – reboot
- Init identifies the default initlevel from /etc/inittab and uses that to load all appropriate program.
- Execute 'grep initdefault /etc/inittab' on your system to identify the default run level
- If you want to get into trouble, you can set the default run level to 0 or 6. Since you know what 0 and 6 means, probably you might not do that.
- Typically you would set the default run level to either 3 or 5.

6. Runlevel programs

- When the Linux system is booting up, you might see various services getting started. For example, it might say "starting sendmail …. OK". Those are the runlevel programs, executed from the run level directory as defined by your run level.
- Depending on your default init level setting, the system will execute the programs from one of the following directories.
  - Run level 0 – /etc/rc.d/rc0.d/
  - Run level 1 – /etc/rc.d/rc1.d/
  - Run level 2 – /etc/rc.d/rc2.d/
  - Run level 3 – /etc/rc.d/rc3.d/
  - Run level 4 – /etc/rc.d/rc4.d/
  - Run level 5 – /etc/rc.d/rc5.d/
  - Run level 6 – /etc/rc.d/rc6.d/

- Please note that there are also symbolic links available for these directory under /etc directly. So, /etc/rc0.d is linked to /etc/rc.d/rc0.d.
- Under the /etc/rc.d/rc*.d/ directories, you would see programs that start with S and K.
- Programs starts with S are used during startup. S for startup.
- Programs starts with K are used during shutdown. K for kill.

- There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.
- For example, S12syslog is to start the syslog deamon, which has the sequence number of 12. S80sendmail is to start the sendmail daemon, which has the sequence number of 80. So, syslog program will be started before sendmail.

Result: The Linux booting process involves hardware initialization by BIOS/UEFI, bootloader execution to load the kernel, kernel initialization, and system setup by the init system, culminating in the login prompt.

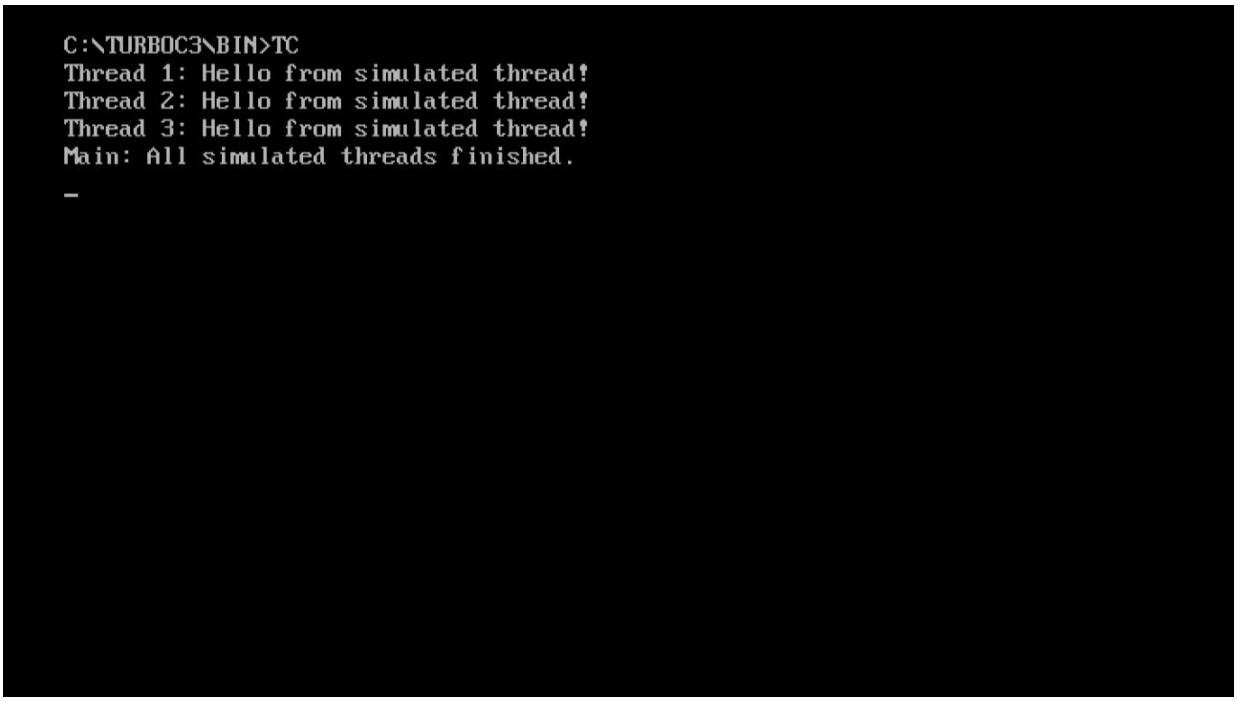
Ex.No:3 **Name:**

Date : **Reg.No:**

MULTI-THREAD PROGRAMMING

```
#include<stdi.h>
#include<conio.h>
#include <dos.h>
void threadFunction(int id) {
   printf("Thread %d: Hello from simulated thread!\n",
   id); delay(500); // Simulate thread execution delay
}

int main() {
   for (int i = 1; i <= 3; i++) {
      threadFunction(i); // Simulate thread-like function calls
   }
   printf("Main: All simulated threads
finished.\n"); getch();
   return 0;
}
```

Output:



Result: Multi-thread programming enables concurrent execution of multiple threads within a process, thus enhancing performance and efficiency by utilizing system resources effectively.

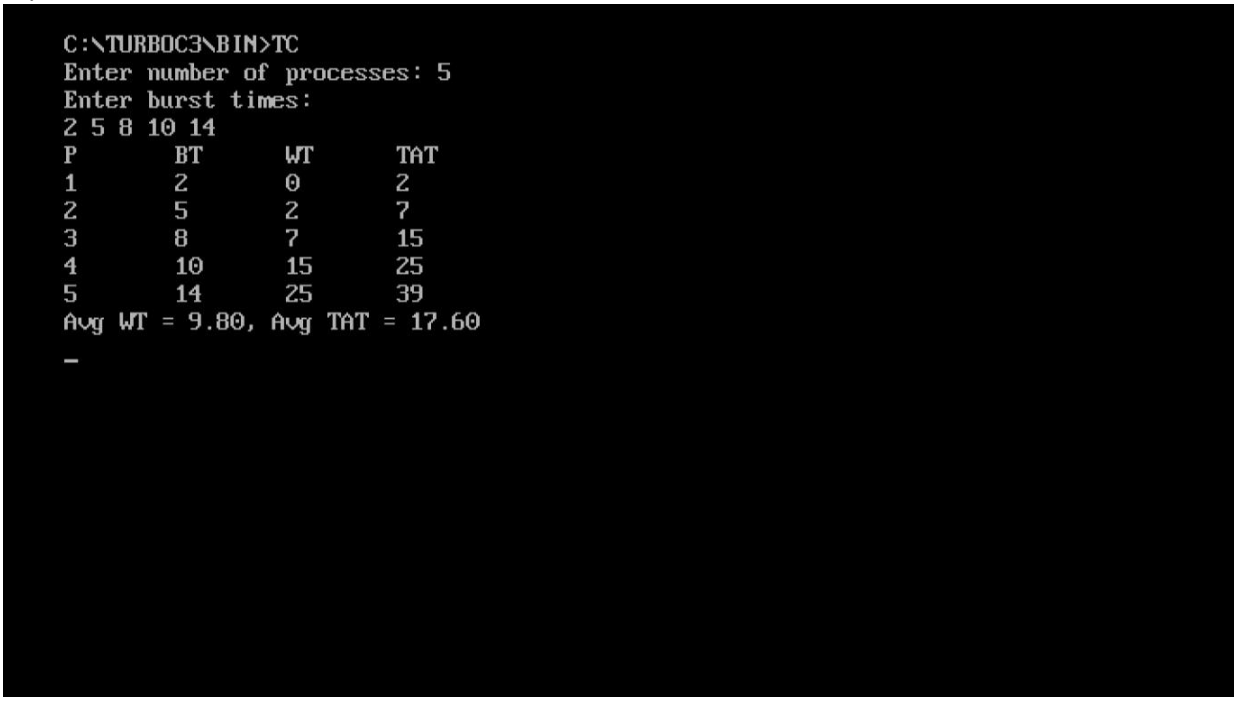
Ex.No:4 **Name:**

Date : **Reg.No:**

Simulation of FCFS CPU scheduling algorithm

```
#include<stdio.h>
#include<conio.h>
int main() {
   int n, bt[10], wt[10] = {0}, tat[10], i;
   float total_wt = 0, total_tat = 0;
   printf("Enter number of processes: ");
scanf("%d", &n);
   printf("Enter burst times:\n");
for (i = 0; i < n; i++) scanf("%d",
   &bt[i]); for (i = 1; i < n; i++)
wt[i] = wt[i - 1] + bt[i - 1];
   for (i = 0; i < n; i++)
{
 tat[i] = wt[i] +
 bt[i]; total_wt +=
 wt[i]; total_tat +=
 tat[i];
 }
     printf("P\tBT\tWT\tTAT
   \n"); for (i = 0; i < n; i++)
 printf("%d\t%d\t%d\t%d\n", i + 1, bt[i], wt[i], tat[i]);
   printf("Avg WT = %.2f, Avg TAT = %.2f\n", total_wt / n,
   total_tat / n); getch();   return 0;
}
```

Output:



Result: Thus, the simulation of the FCFS (First-Come, First-Served) CPU scheduling algorithm processes jobs sequentially based on their arrival time, ensuring simplicity while risking inefficiencies like the convoy effect.

Ex.No:5 **Name:**

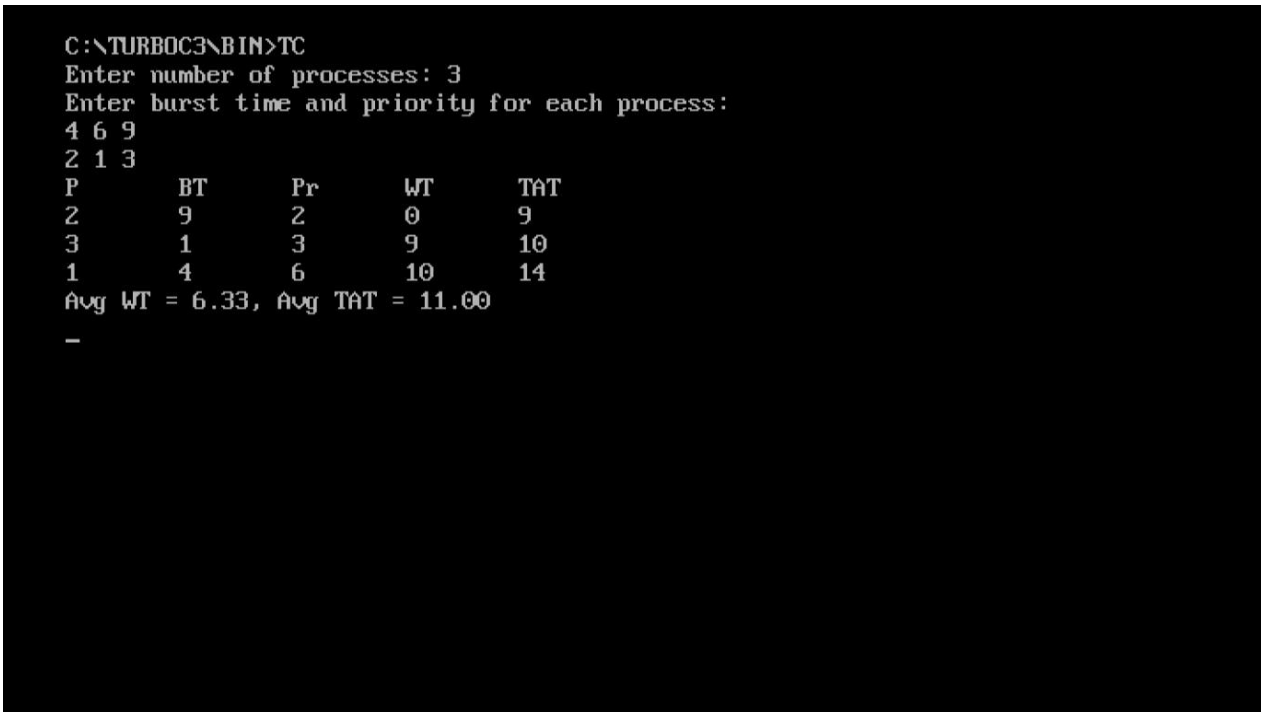
Date : **Reg.No:**

Priority CPU scheduling algorithm

```
#include <stdio.h>
#include<conio.h>

int main() {
   int n, i, j, temp;
   int p[10], bt[10], pr[10], wt[10] = {0}, tat[10]; float
   total_wt = 0, total_tat = 0;

   printf("Enter number of processes: ");
scanf("%d", &n);
   printf("Enter burst time and priority for each
   process:\n"); for (i = 0; i < n; i++) { p[i] = i + 1;
scanf("%d%d", &bt[i],
   &pr[i]); } for (i = 0; i < n
   - 1; i++)
for (j = i + 1; j < n;
j++) if (pr[i] >
pr[j]) {
      temp = pr[i]; pr[i] =
pr[j]; pr[j] = temp;
      temp = bt[i];
bt[i] = bt[j]; bt[j] =
      temp; temp =
      p[i];
p[i] = p[j]; p[j] = temp;
   }
   for (i = 1; i < n;
i++) wt[i] = wt[i - 1]
```

```
+ bt[i - 1];
   for (i = 0; i < n; i++)
 {
 tat[i] = wt[i] +
bt[i]; total_wt +=
wt[i]; total_tat +=
tat[i]; }
   printf("P\tBT\tPr\tWT\tTAT\n");
                  for (i = 0; i < n; i++)
printf("%d\t%d\t%d\t%d\t%d\n", p[i], bt[i], pr[i], wt[i], tat[i]);
   printf("Avg WT = %.2f, Avg TAT = %.2f\n", total_wt / n,
   total_tat / n);
getch();
   return
   0;
}
```

Output:



Result: Thus, the simulation of the Priority CPU scheduling algorithm executes processes based on their priority levels, ensuring critical tasks are handled first, but may lead to starvation of lower-priority processes.

Ex.No:6 **Name:**

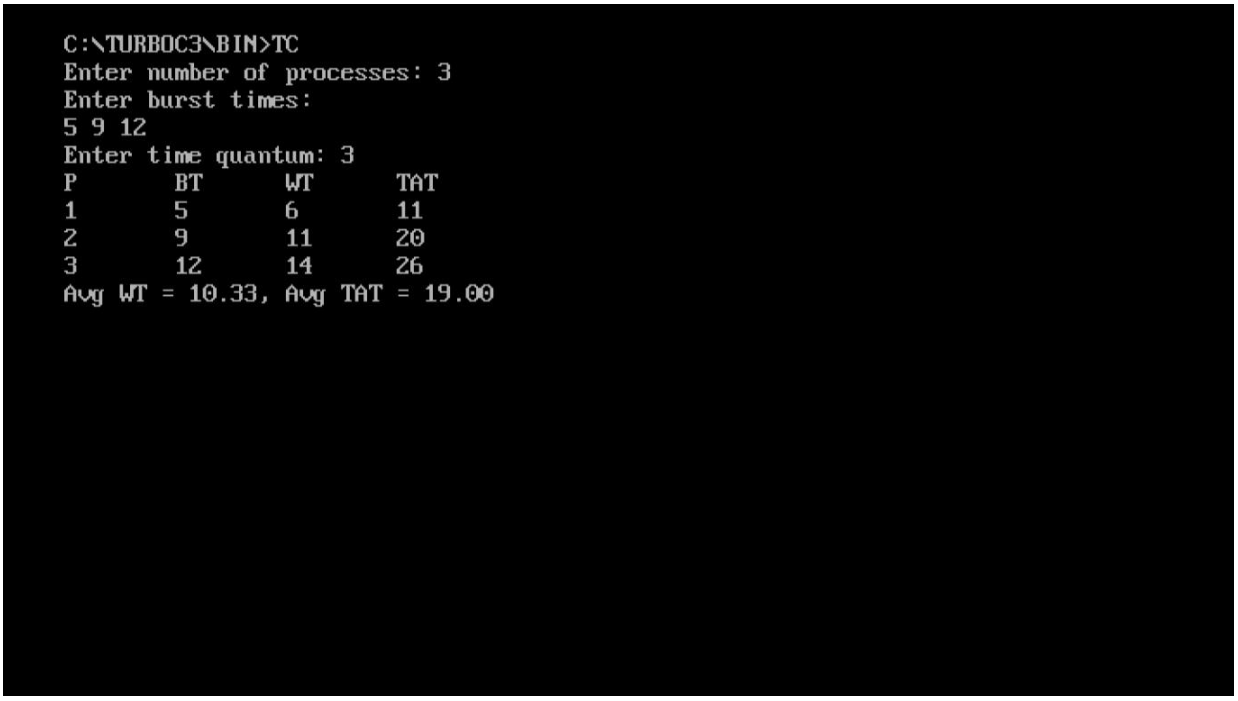
Date : **Reg.No:**

Simulation of Round Robin CPU scheduling algorithm

```
#include<stdio.h>
#include<conio.h>
int main() {
   int n, tq, bt[10], wt[10] = {0}, tat[10], rem_bt[10], time = 0,
   done; float total_wt = 0, total_tat = 0;
   printf("Enter number of processes: ");
 scanf("%d", &n);
   printf("Enter burst
times:\n"); for (int i = 0; i < n;
i++)
{
scanf("%d", &bt[i]); rem_bt[i] = bt[i];
 }
   printf("Enter time quantum: "); scanf("%d", &tq);
   do {
      done = 1;
      for (int i = 0; i < n; i++)
 {
         if  (rem_bt[i]  >
            0) { done = 0;
            if (rem_bt[i] > tq)
{
time += tq; rem_bt[i] -= tq;
 }
            else
{
```

```
time+=
rem_bt[i]; wt[i]
= time - bt[i];
rem_bt[i] = 0; }
        }
      }
   } while (!done);
   for (int i = 0; i < n; i++)
{
tat[i] = wt[i] + bt[i];
total_wt += wt[i];
total_tat += tat[i];
}
   printf("P\tBT\tWT\tTAT\
   n"); for (int i = 0; i < n;
   i++)
printf("%d\t%d\t%d\t%d\n", i + 1, bt[i], wt[i], tat[i]);
   printf("Avg WT = %.2f, Avg TAT = %.2f\n", total_wt / n,
total_tat / n); getch();
   return 0;
}
```

Output:



Result: Thus, the simulation of the Round Robin CPU scheduling algorithm processes tasks in a cyclic order with a fixed time quantum, ensuring fairness but possibly increasing context switching overhead.

Ex.No:7 **Name:**

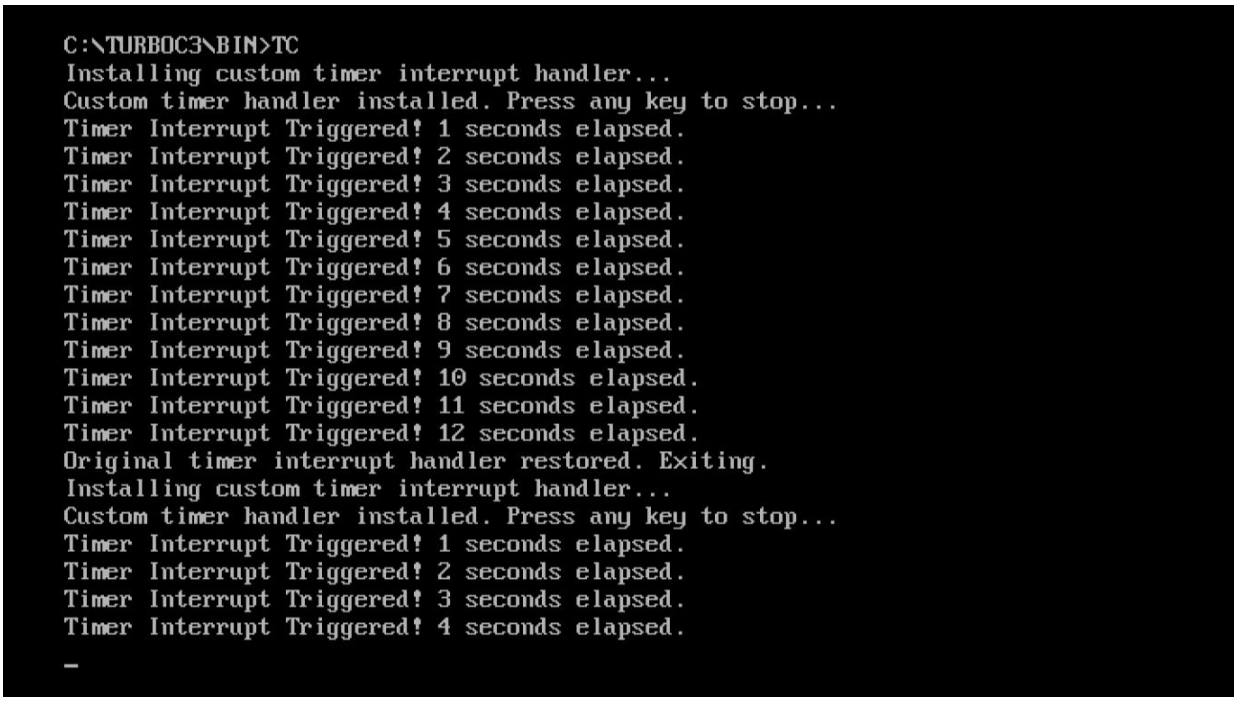
Date : **Reg.No:**

WRITE A PROCEDURE FOR TIMER INTERRUPT HANDLER

```
#include <dos.h>
#include <stdio.h>
 #include <conio.h>

void interrupt (*oldTimerISR)();  // Pointer to store the old interrupt service routine

void                    interrupt
   newTimerISR() {  static
   unsigned  int  ticks  =  0;
   ticks++;
   if (ticks % 18 == 0) {  // Approximately 1-second interval (18.2 ticks/sec)
      printf("Timer Interrupt Triggered! %u seconds elapsed.\n", ticks / 18);
   }
   (*oldTimerISR)();  // Call the original ISR to maintain timer functionality
}
void setupTimerInterrupt() {
   oldTimerISR  =  getvect(0x08);   // Save the old ISR for interrupt 0x08 (timer
   interrupt) setvect(0x08, newTimerISR);  // Set our custom ISR
}
void restoreTimerInterrupt() {
   setvect(0x08, oldTimerISR);  // Restore the original ISR
}
int main() {
   printf("Installing custom timer interrupt handler...\n");
   setupTimerInterrupt();
```

```
    printf("Custom timer handler installed. Press any key to
    stop...\n"); while (!kbhit());  // Wait for key press
    restoreTimerInterrupt();
    printf("Original timer interrupt handler restored.
Exiting.\n"); getch();
    return 0;
}
```

Output:



Result: Successfully implemented a timer interrupt handler that executes a predefined task upon timer expiration.

Ex.No:8 **Name:**

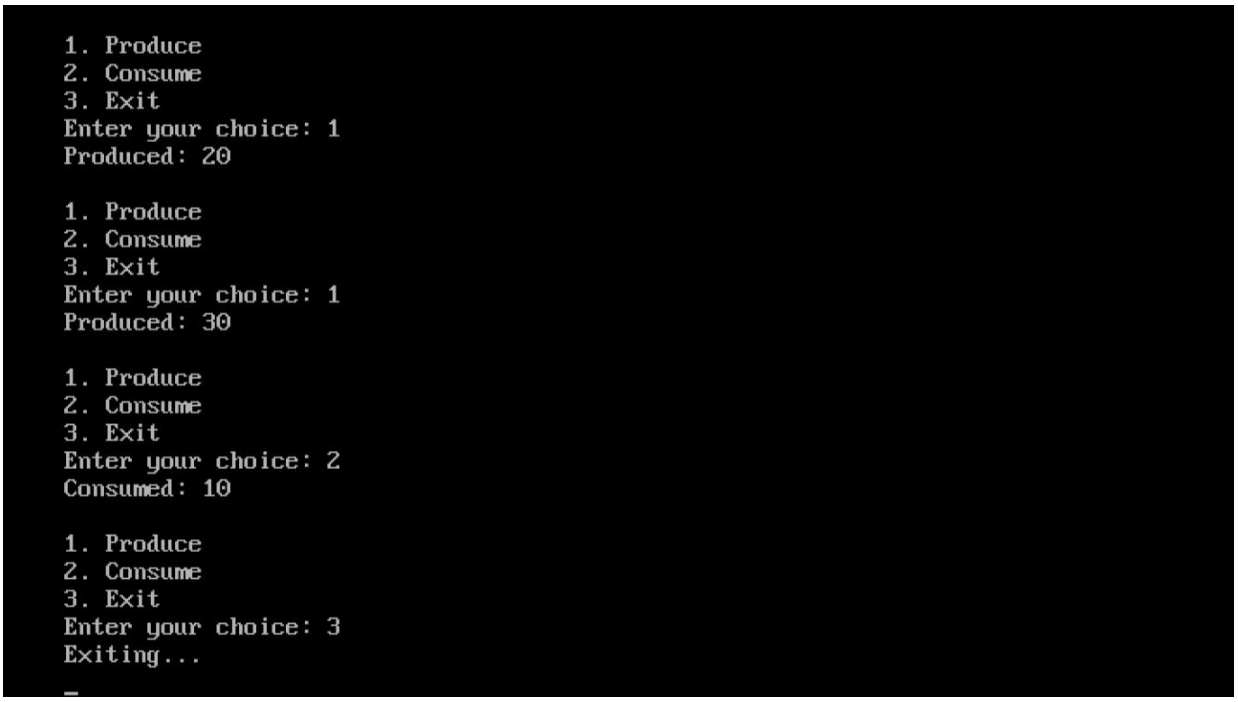
Date : **Reg.No:**

CLASSICAL INTER PROCESS COMMUNICATION PROBLEM (PRODUCER CONSUMER)

```
#include <stdio.h>
#include <conio.h>
#define
BUFFER_SIZE 5
int buffer[BUFFER_SIZE], item; // Shared buffer
int in = 0, out = 0, count = 0; // Indices and
counter void producer() {
   if (count == BUFFER_SIZE) {
      printf("Buffer full! Producer
      waiting...\n"); return;
   }
   item = (in + 1) * 10; // Produce an item
   buffer[in] = item;
   in = (in + 1) % BUFFER_SIZE;
   count++;
   printf("Produced: %d\n", item);
}
void
   consumer() {
   if (count ==
   0) {
      printf("Buffer empty! Consumer waiting...\n");
      return;
   }
   item = buffer[out];
   out = (out + 1) % BUFFER_SIZE;
   count--;
```

```
    printf("Consumed: %d\n", item);
  }
  int main()
    { int
    choice;
    do {
      printf("\n1. Produce\n2. Consume\n3. Exit\n");

printf("Enter your choice: "); scanf("%d", &choice);  switch (choice) {
      case 1:
        producer();
        break;
      case 2:
        consumer
        (); break;
      case 3:
        printf("Exiting...\n");
        break;
      default:
        printf("Invalid choice!\n");
      }
    } while (choice !=
    3); getch();
    return 0;
  }
```

Output:



Result: This Program implemented the Producer-Consumer problem using inter-process communication, ensuring synchronized data exchange between processes.

Ex.No:9 **Name:**

Date : **Reg.No:**

WRITE A PROCEDURE TO MAKE MESSAGE PASSING IN INTER PROCESS COMMUNICATION

```
#include<stdio.h>
#include<conio.h>
#include <string.h>
#define BUFFER_SIZE 100

char buffer[BUFFER_SIZE]; // Shared memory buffer
int message_flag = 0;  // Message availability flag (0 = empty, 1 = full)

// Function to simulate the sender
process void sender() {
   if (message_flag == 1) {
      printf("Buffer full! Sender
      waiting...\n"); return;
   }

   printf("Enter a message to send: ");
   fflush(stdin); // Clear input buffer
   gets(buffer);  // Read message into
   buffer message_flag = 1; // Mark buffer
   as full printf("Message sent
   successfully!\n");
}

// Function to simulate the receiver
process void receiver() {
   if (message_flag == 0) {
      printf("Buffer empty! Receiver
      waiting...\n"); return;
```

```
}
```

```
    printf("Received message: %s\n", buffer);
    strcpy(buffer, ""); // Clear the buffer
    message_flag = 0;  // Mark buffer as empty
    printf("Message received successfully!\n");
}

int main()
    { int
    choice;
    do {
        printf("\n1. Send Message\n2. Receive Message\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) { case 1:
            sender
            ();
            break;
        case 2:
            receiver();
            break;
        case 3:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 3);

    getch(); // Wait for key press before exiting
    return 0;}
```

Output

```
C:\TURBOC3\BIN>TC

1. Send Message
2. Receive Message
3. Exit
Enter your choice: 1
Enter a message to send: Hello SRM
Message sent successfully!

1. Send Message
2. Receive Message
3. Exit
Enter your choice: 2
Received message: Hello SRM
Message received successfully!

1. Send Message
2. Receive Message
3. Exit
Enter your choice: 3
Exiting...
```

Result: Thus the program, implemented message passing in IPC, enabling efficient and synchronized communication between processes.

Ex.No:10 **Name:**

Date : **Reg.No:**

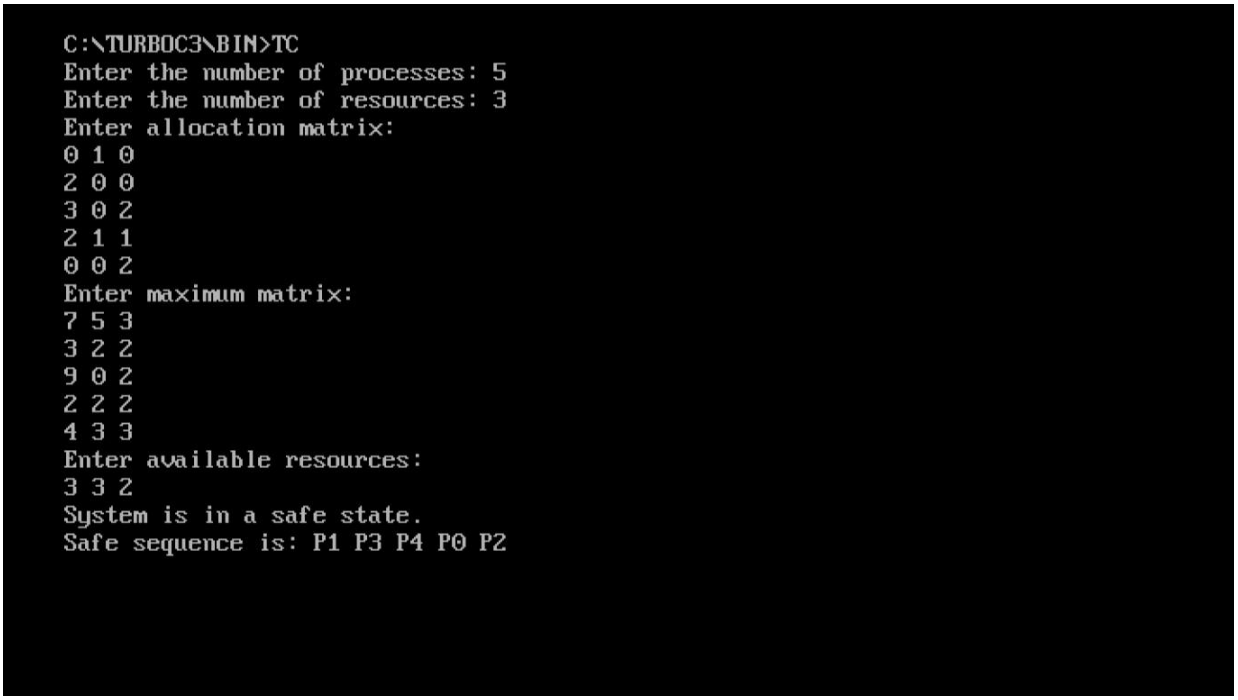
PROGRAM TO IMPLEMENT BANKERS' ALGORITHM

```
#include <stdio.h>
 #include <conio.h>

#define MAX 10

int allocation[MAX][MAX], max[MAX][MAX], need[MAX][MAX],
available[MAX]; int n, m; // Number of processes (n) and resources (m)

void input()
   { int i, j;
   printf("Enter the number of processes: ");
   scanf("%d", &n);
   printf("Enter the number of resources: ");
   scanf("%d", &m);

   printf("Enter allocation
   matrix:\n"); for (i = 0; i < n; i++)
     for (j = 0; j < m; j++)
        scanf("%d",
        &allocation[i][j]);

   printf("Enter maximum
   matrix:\n"); for (i = 0; i < n; i++)
     for (j = 0; j < m; j++)
        scanf("%d",
        &max[i][j]);

   printf("Enteravailable resources:\n"); for (i = 0; i < m; i++)
```

```
    scanf("%d", &available[i]);

            // Calculate the need matrix for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
      need[i][j] = max[i][j] - allocation[i][j];
}

int is_safe() {
  int work[MAX], finish[MAX] = {0}, safe_seq[MAX],
  count = 0; int i, j;

  for (i = 0; i < m; i++)
    work[i]          =
    available[i];

  while (count < n)
    { int found = 0;
    for (i = 0; i < n;
      i++)    {    if
      (!finish[i]) {
        for (j = 0; j < m; j++)
          if    (need[i][j]    >
            work[j]) break;

        if (j == m) {
          for (j = 0; j < m; j++)
            work[j]              +=
          allocation[i][j];
          safe_seq[count++]  =  i;
          finish[i] = 1;
          found = 1;
        }
      }
```

```
        }
  if (!found) {
            printf("System is in an unsafe state.\n");
            return 0;
        }
    }
    printf("System is in a safe state.\nSafe sequence is:
    "); for (i = 0; i < n; i++)
        printf("P%d                    ",
    safe_seq[i]); printf("\n");
    return 1;
}
int main()
{
input();
is_safe();
getch();
return 0;
}
```

Output:



Result: Successfully implemented the Banker's Algorithm, ensuring safe resource allocation and deadlock avoidance.

Ex.No:11 **Name:**

Date : **Reg.No:**

PROGRAM TO IMPLEMENT MEMORY ALLOCATION WITH PAGES

```
#include <stdio.h>
 #include <conio.h>
#define MAX 10 // Maximum number of pages or frames


int main() {
   int page_table[MAX], frames[MAX], n_pages, n_frames, i, page, frame;

   printf("Enter the number of pages (max %d): ",
   MAX); scanf("%d", &n_pages);
   printf("Enter the number of frames (max %d): ",
   MAX); scanf("%d", &n_frames);
   if (n_pages > MAX || n_frames > MAX) {
      printf("Number of pages or frames exceeds
      limit!\n");
      return 1;
   }
   // Initialize all frames to -1
   (empty) for (i = 0; i <
   n_frames; i++)
      frames[i] = -1;

   // Simulate page table mapping
   printf("Enter the frame numbers for each page (0 to %d):\n", n_frames
   - 1); for (i = 0; i < n_pages; i++) {
      printf("Page %d -> Frame: ", i);
      scanf("%d", &frame);
      if (frame < 0 || frame >= n_frames) {
         printf("Invalid frame number!\n");
```

```
            return 1;
        }
        page_table[i] = frame;
        frames[frame] = i; // Map frame to page
    }

    // Display page table
    printf("\nPage
    Table:\n");
    printf("Page\tFrame\n")
    ; for (i = 0; i < n_pages;
    i++) {
        printf("%d\t%d\n", i, page_table[i]);
    }

    // Simulate memory access
    printf("\nEnter a page number to access (-1 to
    exit): "); while (1) {
        scanf("%d",
        &page); if (page
        == -1) {
            printf("Exiting...\n");
            break;
        }
        if (page < 0 || page >= n_pages) {
            printf("Invalid page number!\n");
        } else {
            printf("Page %d is in Frame %d.\n", page, page_table[page]);
        }
    }
    getch();
    return 0;
}
```

Output:

```
Enter the number of pages (max 10): 4
Enter the number of frames (max 10): 3
Enter the frame numbers for each page (0 to 2):
Page 0 -> Frame: 1
Page 1 -> Frame: 2
Page 2 -> Frame: 0
Page 3 -> Frame: 1

Page Table:
Page    Frame
0       1
1       2
2       0
3       1

Enter a page number to access (-1 to exit): 2
Page 2 is in Frame 0.
3
Page 3 is in Frame 1.
9
Invalid page number!
-1
Exiting...
```

Result: Successfully implemented memory allocation using paging, ensuring efficient memory management and address translation.

Ex.No:12 **Name:**

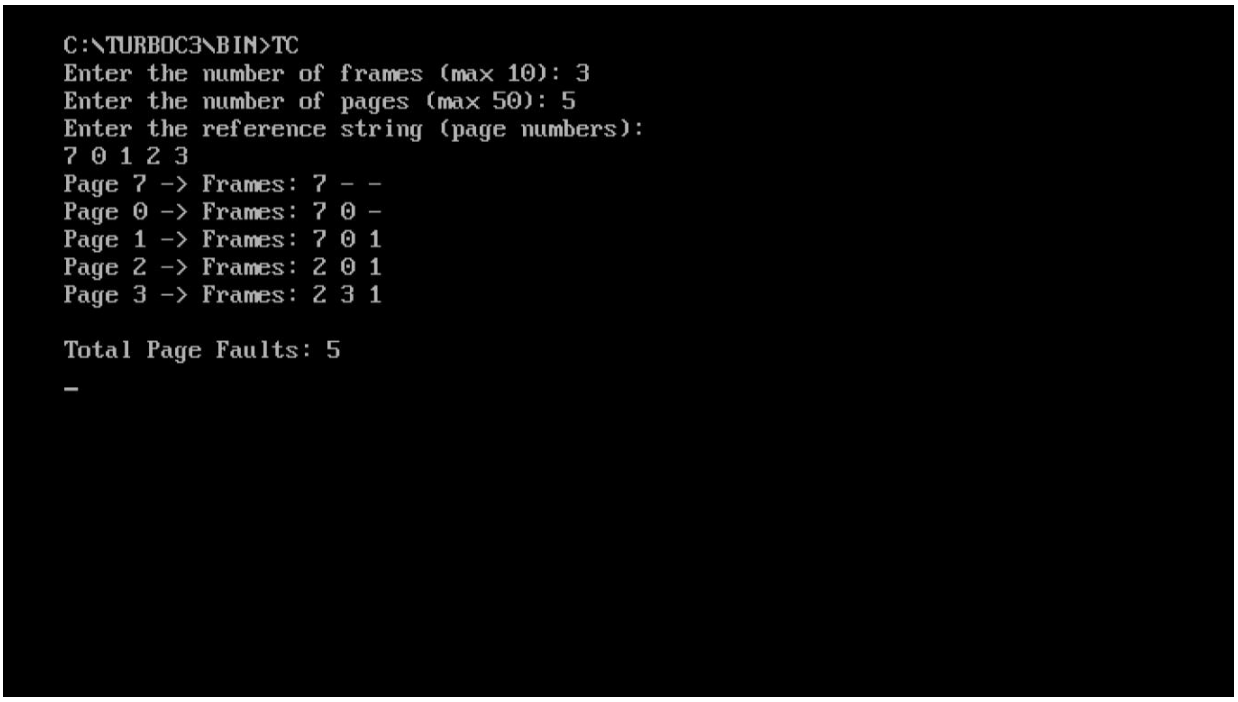
Date : **Reg.No:**

SIMULATION OF FIFO PAGE REPLACEMENT ALGORITHM

```
#include <stdio.h>
 #include <conio.h>
 #define MAX_FRAMES 10
#define MAX_PAGES 50
 int main() {
     int frames[MAX_FRAMES],
     pages[MAX_PAGES]; int n_frames,
     n_pages, i, j, k, page_faults = 0;
     int index = 0; // Points to the oldest frame for replacement
     printf("Enter the number of frames (max %d): ",
     MAX_FRAMES); scanf("%d", &n_frames);
     printf("Enter the number of pages (max %d): ",
     MAX_PAGES); scanf("%d", &n_pages);
     printf("Enter the reference string (page
     numbers):\n"); for (i = 0; i < n_pages; i++) {
        scanf("%d", &pages[i]);
     }
     // Initialize frames to -1 (indicating
     empty) for (i = 0; i < n_frames; i++) {
        frames[i] = -1;
     }
     // Simulate FIFO Page Replacement for (i = 0; i < n_pages; i++) {
        int found = 0;

        // Check if the page is already in one of the
        frames for (j = 0; j < n_frames; j++) {
           if (frames[j] ==
```

```
            pages[i]) { found = 1;
            break;
          }
        }
        if (!found) { // Page fault occurs
          frames[index] = pages[i]; // Replace the oldest page
          index = (index + 1) % n_frames; // Update the index to point to the next
          frame page_faults++;
        }
        // Display current frame contents
         printf("Page %d -> Frames: ", pages[i]); for (k = 0; k < n_frames; k++) {
          if (frames[k] != -1)
            printf("%d ",
          frames[k]); else
            printf("- ");
        }
        printf("\n");
    }
    printf("\nTotal Page Faults: %d\n",
    page_faults); getch();
    return 0;
}
```

Output:



Result: Thus, the simulated the FIFO page replacement algorithm, demonstrating page faults and efficient memory management.

Ex.No:13 **Name:**

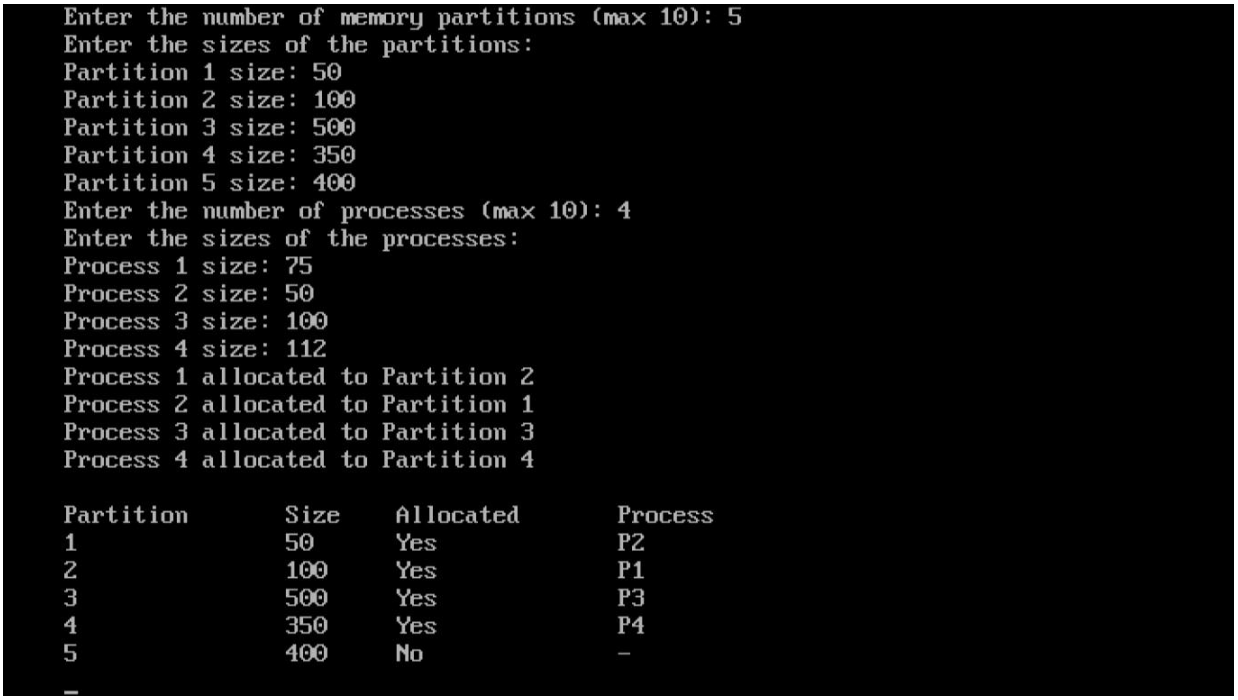
Date : **Reg.No:**

MULTIPLE PARTITION (DYNAMIC MEMORY ALLOCATION METHOD)

```
#include <stdio.h>
 #include <conio.h>
#define MAX 10

typedef struct
{
 int size;
   int allocated; int process;
} Partition;

int main() {
   Partition memory[MAX];
   int n_partitions, n_processes, process[MAX], i, j;

   printf("Enter the number of memory partitions (max %d): ",
   MAX); scanf("%d", &n_partitions);
   printf("Enter the sizes of the
   partitions:\n"); for (i = 0; i <
   n_partitions; i++) {
      printf("Partition %d size: ", i +
      1); scanf("%d",
      &memory[i].size);
      memory[i].allocated = 0;
      memory[i].process = -1;
   }
   printf("Enter the number of processes (max %d): ",
   MAX); scanf("%d", &n_processes);
   printf("Enter the sizes of the processes:\n");
```

```
  for (i = 0; i < n_processes; i++) {
    printf("Process %d size: ", i +
    1); scanf("%d", &process[i]);
  }
  // Allocate processes using First
  Fit for (i = 0; i < n_processes;
  i++) {
    for (j = 0; j < n_partitions; j++) {
      if (!memory[j].allocated && memory[j].size >= process[i])
        { memory[j].allocated = 1;
        memory[j].process = i + 1;
        printf("Process %d allocated to Partition %d\n", i + 1, j + 1);
        break;
      }
    }
    if (j == n_partitions) {
      printf("Process %d cannot be allocated (Insufficient memory).\n", i + 1);
    }
  }

  // Display memory allocation
  printf("\nPartition\tSize\tAllocated\tProcess\
  n"); for (i = 0; i < n_partitions; i++) {
    printf("%d\t\t%d\t%s\t\t", i + 1, memory[i].size, memory[i].allocated ? "Yes"
    : "No"); if (memory[i].allocated) {
      printf("P%d\n", memory[i].process);
    } else {
      printf("-\n");
} }
  getch();
  return
  0;
}
```

Output:



Result: Successfully implemented multiple partition dynamic memory allocation, thus optimizing memory utilization and reducing fragmentation.

Ex.No:14 **Name:**

Date : **Reg.No:**

SIMULATION OF LRU PAGE REPLACEMENT ALGORITHM

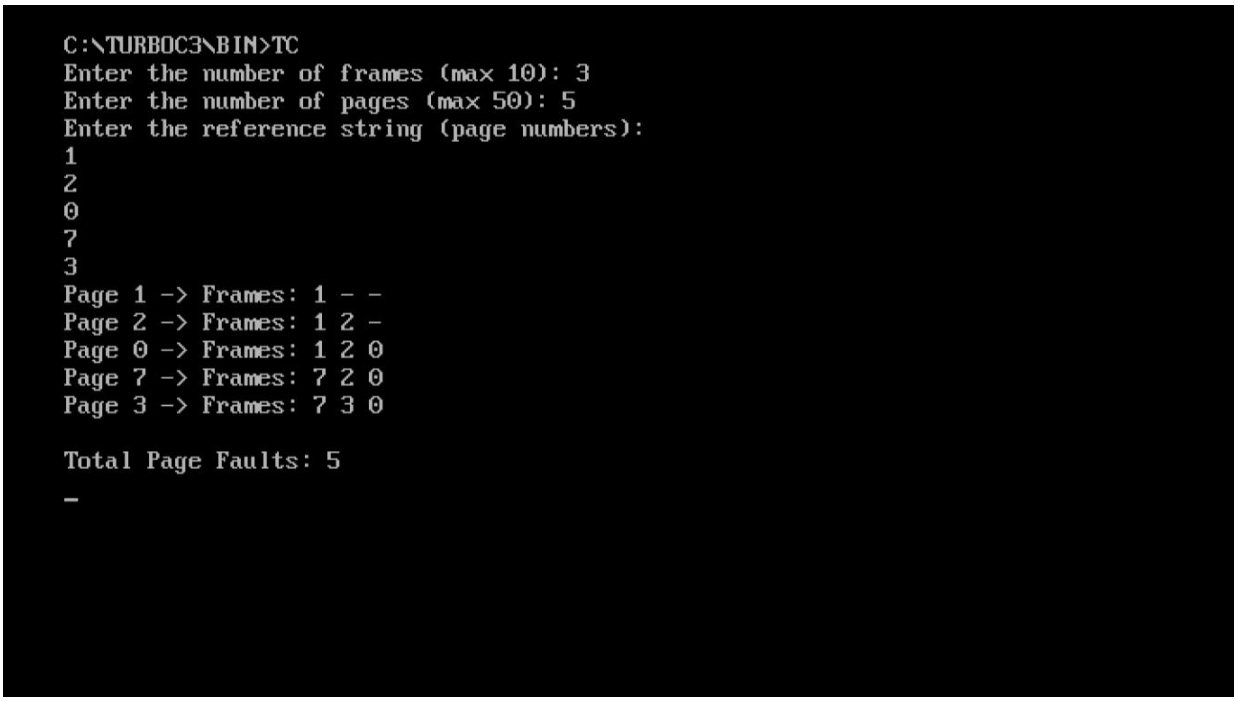
```
#include <stdio.h>
#include <conio.h>

#define MAX_FRAMES 10
#define MAX_PAGES 50

int findLRU(int time[], int
   n) { int i, min = time[0],
   pos = 0; for (i = 1; i < n;
   i++) {
      if (time[i] <
         min) { min =
         time[i]; pos
         = i;
      }
   }
   return pos;
}

int main() {
   int frames[MAX_FRAMES], pages[MAX_PAGES],
   time[MAX_FRAMES];  int n_frames, n_pages, i, j, pos,
   page_faults = 0, counter = 0, flag1, flag2;
   printf("Enter the number of frames (max %d): ",
   MAX_FRAMES); scanf("%d", &n_frames);
   printf("Enter the number of pages (max %d): ",
   MAX_PAGES); scanf("%d", &n_pages);
   printf("Enter the reference string (page
```

```
numbers):\n"); for (i = 0; i < n_pages; i++) {
   scanf("%d", &pages[i]);
```

```
}

// Initialize frames and time
arrays for (i = 0; i < n_frames;
i++) {
   frames[i] = -1;
   time[i] = 0;
}
// Simulate LRU Page
Replacement for (i = 0; i <
n_pages; i++) {
   flag1 = flag2 = 0;
   // Check if the page is already in a
   frame for (j = 0; j < n_frames; j++) {
      if (frames[j] ==
         pages[i]) { flag1 =
         flag2 = 1;
         time[j] = ++counter; // Update time for the page
         break;
      }
   }
   // If the page is not in a
   frame if (!flag1) {
      for (j = 0; j < n_frames; j++) {
         if (frames[j] == -1) { // Check for an empty
            frame frames[j] = pages[i];
            time[j] =
            ++counter;
            page_faults++;
            flag2 = 1;
            break;
```

```
            }
          }
        }
        // If no empty frame is available, replace the least recently used
        page if (!flag2) {
pos = findLRU(time, n_frames); frames[pos] = pages[i]; time[pos] = ++counter; page_faults++;
        }
        // Display the current state of frames
        printf("Page %d -> Frames: ",
        pages[i]); for (j = 0; j < n_frames;
        j++) {
          if (frames[j] != -1)
            printf("%d ",
          frames[j]); else
            printf("- ");
        }
        printf("\n");
      }
      printf("\nTotal Page Faults: %d\n",
      page_faults); getch();
      return 0;
    }
```

Output:



Result: Successfully simulated the LRU page replacement algorithm, thus optimizing page fault handling and memory management.

Ex.No:15 **Name:**

Date : **Reg.No:**

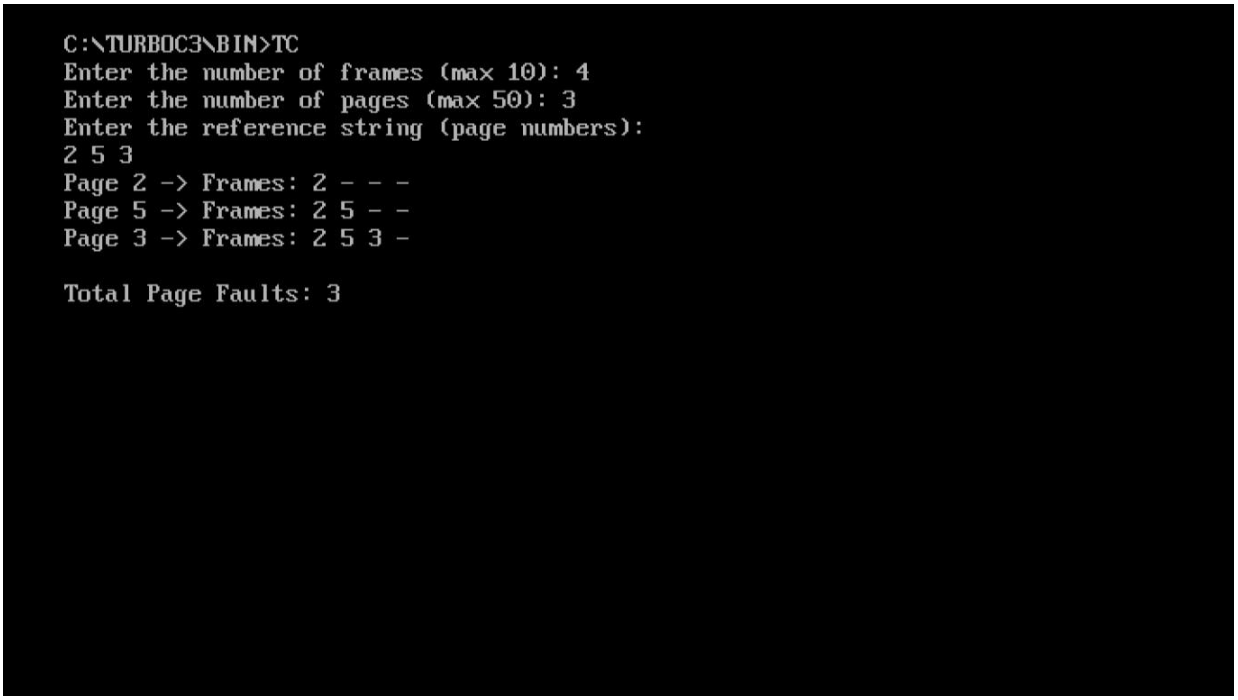
SIMULATION OF OPTIMAL PAGE REPLACEMENT ALGORITHM

```
#include      <stdio.h>
#include <conio.h>
#define MAX_FRAMES 10
#define MAX_PAGES 50

// Function to find the page to replace
int findOptimal(int pages[], int frames[], int n_pages, int current, int n_frames) { int farthest
   = current, pos = -1, i, j;
   for (i = 0; i < n_frames;
      i++) { int found = 0;
      for (j = current + 1; j < n_pages;
         j++) { if (frames[i] == pages[j])
         {
            if       (j       >
               farthest)    {
               farthest = j;
               pos = i;
            }
            found = 1;
            break;
         }
      }
      if (!found) return i; // If the frame is never used in the future
   }
   return pos == -1 ? 0 : pos; // Replace the least recently needed page
}

int main() {
```

```
    int frames[MAX_FRAMES], pages[MAX_PAGES];
int n_frames, n_pages, i, j, page_faults = 0, flag1, flag2;
    printf("Enter the number of frames (max %d): ",
    MAX_FRAMES); scanf("%d", &n_frames);
    printf("Enter the number of pages (max %d): ", MAX_PAGES);
    scanf("%d", &n_pages);
    printf("Enter the reference string (page
    numbers):\n"); for (i = 0; i < n_pages; i++) {
      scanf("%d", &pages[i]);
    }
    // Initialize frames
    for (i = 0; i < n_frames; i++)
      { frames[i] = -1;
    }
    // Simulate Optimal Page
    Replacement for (i = 0; i < n_pages;
    i++) {
      flag1 = flag2 = 0;
      // Check if the page is already in a
      frame for (j = 0; j < n_frames; j++) {
        if (frames[j] ==
          pages[i]) { flag1 = 1;
          break;
        }
      }
      // If the page is not in a
      frame if (!flag1) {
```

```
        for (j = 0; j < n_frames; j++) {
          if (frames[j] == -1) { // Check for an empty frame
            frames[j] = pages[i];
            page_faults
            ++;  flag2  =
            1;
            break;
          }
        }
      }

      // If no empty frame is available, replace the optimal
      page if (!flag2 && !flag1) {
        int pos = findOptimal(pages, frames, n_pages, i, n_frames);
        frames[pos] = pages[i];
        page_faults++;
      }

      // Display the current state of frames
      printf("Page  %d  ->  Frames:  ",
      pages[i]);  for  (j  =  0;  j  <  n_frames;
      j++) {
        if (frames[j] != -1)
          printf("%d          ",
        frames[j]); else
          printf("- ");
      }
      printf("\n");
    }
    printf("\nTotal  Page  Faults:  %d\n",
    page_faults); getch();
    return 0;
}
```

Output:



Result: This program successfully simulated the Optimal Page Replacement Algorithm, achieving minimal page faults and efficient memory management.