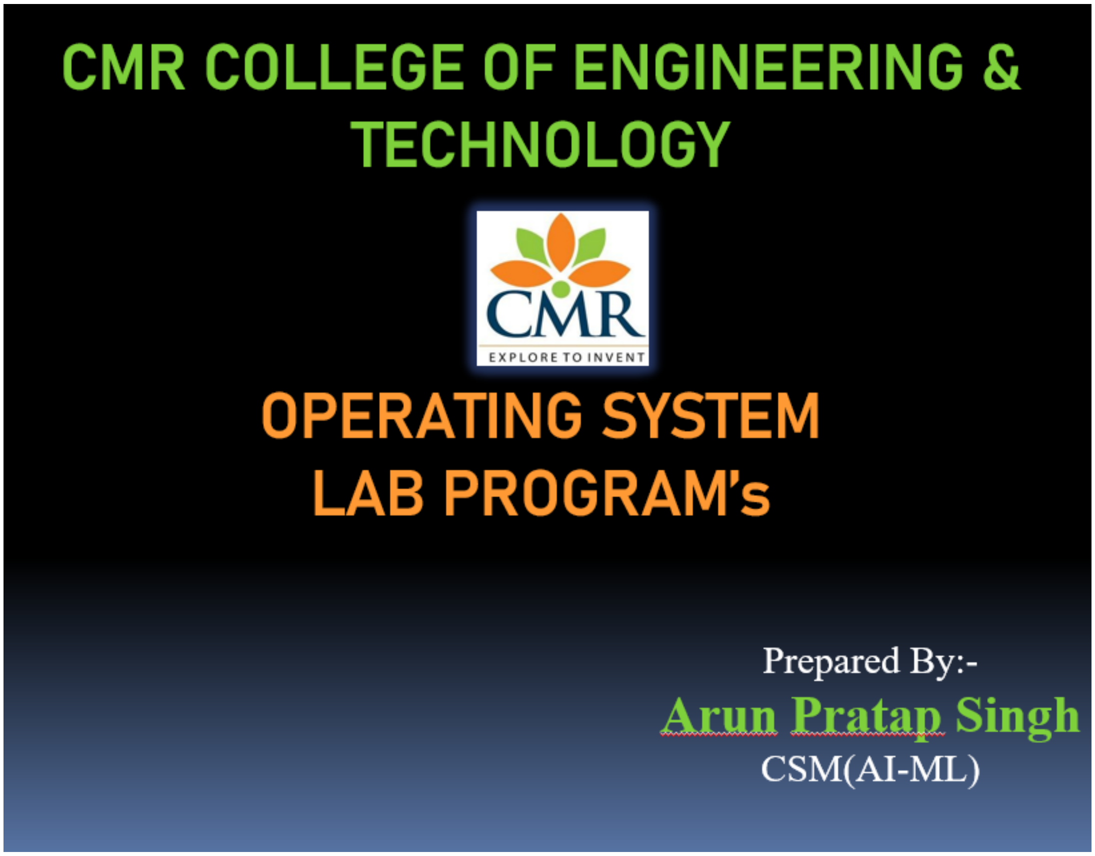
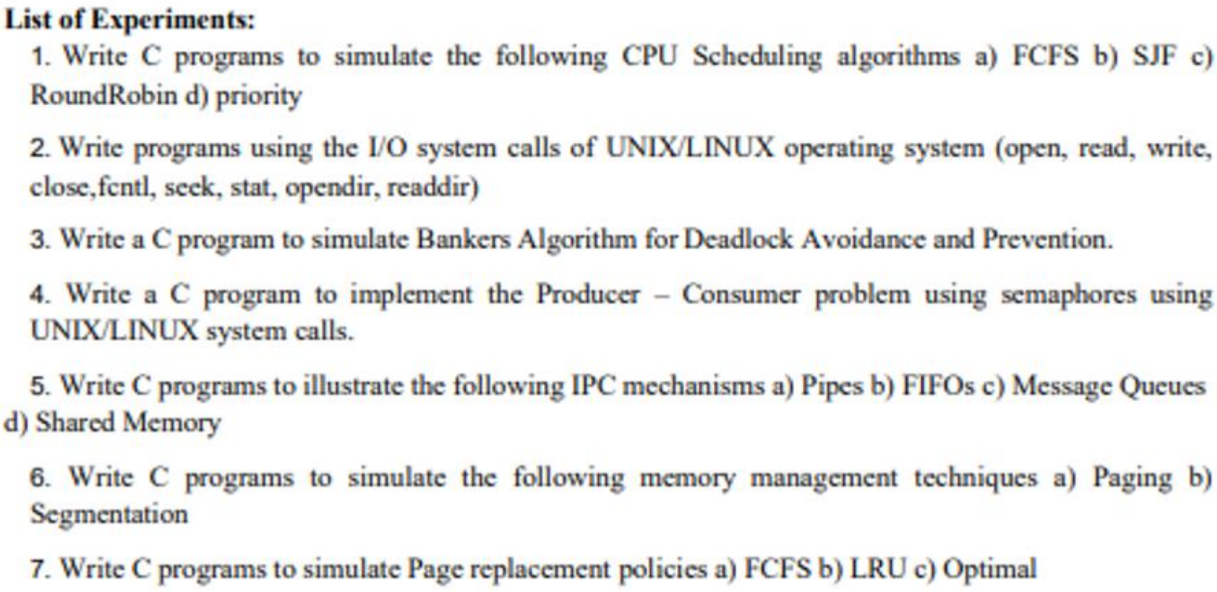


CMR COLLEGE OF ENGINEERING & TECHNOLOGY

CMR
EXPLORE TO INVENT

OPERATING SYSTEM LAB PROGRAM's

Prepared By:-
Arun Pratap Singh
CSM(AI-ML)



List of Experiments:

1. Write C programs to simulate the following CPU Scheduling algorithms a) FCFS b) SJF c) RoundRobin d) priority

2. Write programs using the I/O system calls of UNIX/LINUX operating system (open, read, write, close,fcntl, seek, stat, opendir, readdir)

3. Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention.

4. Write a C program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

5. Write C programs to illustrate the following IPC mechanisms a) Pipes b) FIFOs c) Message Queues d) Shared Memory

6. Write C programs to simulate the following memory management techniques a) Paging b) Segmentation

7. Write C programs to simulate Page replacement policies a) FCFS b) LRU c) Optimal

WEEK -1

Write C programs to simulate the following CPU Scheduling algorithms

// a). C program for implementation of FCFS CPU Scheduling

```
#include < stdio.h >
// Function to find the waiting time for all  Processes
void findWaitingTime( int processes[ ] , int n ,
                     int bt[ ], int wt[ ])
{
    // waiting time for first process is 0
    wt[ 0 ] = 0;

    // calculating waiting time
    for ( int i = 1 ; i < n ; i++ )
        wt[ i ] = bt[ i - 1 ] + wt[ i - 1 ] ;
}
// Function to calculate turn around time
void findTurnAroundTime ( int processes[ ], int n,
                        int bt[ ] , int wt [ ] , int tat[ ] )
{
    // calculating turnaround time by adding
    // bt [ i ] + wt [ i ]
    for ( int i = 0 ; i < n ; i + + )
        tat[i] = bt[i] + wt[i];
}
```

```
    }

// Function to calculate average time
void findavgTime(int processes[], int n, int bt[])
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    // Function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt);

    // Function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);

    // Display processes along with all details
    printf("Processes Burst time Waiting time Turn around time\n");

    // Calculate total waiting time and total turn
    // around time
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf(" %d ", (i + 1));
        printf("     %d ", bt[i]);
        printf("     %d", wt[i]);
        printf("     %d\n", tat[i]);
    }
    int s = (float)total_wt / (float)n;

    int t = (float)total_tat / (float)n;

    printf("Average waiting time = %d", s);

    printf("\n");

    printf("Average turn around time = %d ", t);

}


// Driver code

int main()

{

    // process id's

    int processes[] = {1, 2, 3};

    int n = sizeof processes / sizeof processes[0];


    // Burst time of all processesN

    int burst_time[] = {10, 5, 8};


    findavgTime(processes, n, burst_time);

    return 0;

}
```

Output:

```
/*
 * FCFS Scheduling Program in C
 */

#include <stdio.h>
int main()
{
    int pid[15];
    int bt[15];
    int n;
    printf("Enter the number of processes: ");
    scanf("%d",&n);

    printf("Enter process id of all the processes: ");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&pid[i]);
    }

    printf("Enter burst time of all the processes: ");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&bt[i]);
    }

    int i, wt[n];
    wt[0]=0;

    //for calculating waiting time of each process
    for(i=1; i<n; i++)
    {
        wt[i]= bt[i-1]+ wt[i-1];
    }

    printf("Process ID     Burst Time     Waiting Time     TurnAround Time\n");
    float twt=0.0;
    float tat= 0.0;
    for(i=0; i<n; i++)
    {
        printf("%d\t\t", pid[i]);
        printf("%d\t\t", bt[i]);
        printf("%d\t\t", wt[i]);

        //calculating and printing turnaround time of each process
        printf("%d\t\t", bt[i]+wt[i]);
        printf("\n");
        //for calculating total waiting time
        twt += wt[i];

        //for calculating total turnaround time
        tat += (wt[i]+bt[i]);
    }
    float att,awt;

    //for calculating average waiting time
    awt = twt/n;

    //for calculating average turnaround time
    att = tat/n;
    printf("Avg. waiting time= %f\n",awt);
    printf("Avg. turnaround time= %f",att);
}
```

OUTPUT:-

// b). C program for implementation of SJF CPU Scheduling

Types of SJF

1-Code for Non-Preemptive SJF CPU Scheduling

```
#include<stdio.h>

int main() {

  int time, burst_time[10], at[10], sum_burst_time = 0, smallest, n, i;

  int sumt = 0, sumw = 0;

  printf("enter the no of processes : ");

  scanf("%d", & n);

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

    printf("the arrival time for process P%d : ", i + 1);

    scanf("%d", & at[i]);
    printf("the burst time for process P%d : ", i + 1);

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

    sum_burst_time += burst_time[i];

  }

  burst_time[9] = 9999;

  for (time = 0; time < sum_burst_time;) {

    smallest = 9;

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

      if (at[i] <= time && burst_time[i] > 0 && burst_time[i] < burst_time[smallest])

        smallest = i;

    }

    printf("P[%d]\t|\t%d\t|\t%d\n", smallest + 1, time + burst_time[smallest] - at[smallest], time - at[smallest]);

    sumt += time + burst_time[smallest] - at[smallest];

    sumw += time - at[smallest];

    time += burst_time[smallest];

    burst_time[smallest] = 0;

  }

  printf("\n\n average waiting time = %f", sumw * 1.0 / n);

  printf("\n\n average turnaround time = %f", sumt * 1.0 / n);

  return 0;

}
```

OUTPUT:-

2-Code for Pre-emptive SJF CPU Scheduling

```
#include<stdio.h>

 int main()

{

    int burst_time[20],p[20],waiting_time[20],tat[20],i,j,n,total=0,pos,temp;

    float avg_waiting_time,avg_tat;

    printf("please enter number of process: ");

    scanf("%d",&n);

    printf("\n enter the Burst Time:\n");

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

    {

        printf("p%d:",i+1);

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

        p[i]=i+1;

    }

   // from here, burst times sorted

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

    {

        pos=i;

        for(j=i+1;j<n;j++)

        {

            if(burst_time[j]<burst_time[pos])

                pos=j;

        }

        temp=burst_time[i];
        burst_time[i]=burst_time[pos];

        burst_time[pos]=temp;

        temp=p[i];

        p[i]=p[pos];

        p[pos]=temp;

    }

    waiting_time[0]=0;

    for(i=1;i<n;i++)

    {

        waiting_time[i]=0;

        for(j=0;j<i;j++)

            waiting_time[i]+=burst_time[j];

        total+=waiting_time[i];

    }

    avg_waiting_time=(float)total/n;

    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");

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

    {

        tat[i]=burst_time[i]+waiting_time[i];

        total+=tat[i];

        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],burst_time[i],waiting_time[i],tat[i]);

    }

    avg_tat=(float)total/n;

    printf("\n\n the average Waiting Time=%f",avg_waiting_time);
```

```
    printf("\n  the average Turnaround Time=%f\n",avg_tat);
}
```

OUTPUT:-

// c). C program for implementation of Round-Robin CPU Scheduling

```
#include<stdio.h>
int main()
{
   int cnt,j,n,t,remain,flag=0,tq;
  int wt=0,tat=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  for(cnt=0;cnt<n;cnt++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process Number %d :",cnt+1);
    scanf("%d",&at[cnt]);
    scanf("%d",&bt[cnt]);
    rt[cnt]=bt[cnt];
  }
  printf("Enter Time Quantum:\t");
  scanf("%d",&tq);
  printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
```

```
for(t=0,cnt=0;remain!=0;)
{
  if(rt[cnt]<=tq && rt[cnt]>0)
  {
    t+=rt[cnt];
    rt[cnt]=0;
    flag=1;
  }
  else if(rt[cnt]>0)
  {
    rt[cnt]-=tq;
    t+=tq;
  }
  if(rt[cnt]==0 && flag==1)
  {
    remain--;
    printf("P[%d]\t|\t%d\t|\t%d\n",cnt+1,t-at[cnt],t-at[cnt]-bt[cnt]);
    wt+=t-at[cnt]-bt[cnt];
    tat+=t-at[cnt];
    flag=0;
  }
  if(cnt==n-1)
    cnt=0;
  else if(at[cnt+1]<=t)
    cnt++;
    else
      cnt=0;
  }
  printf("\nAverage Waiting Time= %f\n",wt*1.0/n);
  printf("Avg Turnaround Time = %f",tat*1.0/n);


  return 0;
}
```

OUTPUT:-

// d). C program for implementation of Priority CPU Scheduling

```
#include <stdio.h>

void swap(int *a,int *b) {

    int temp=*a;

    *a=*b;

    *b=temp;

}

int main() {

    int n;

    printf("Enter Number of Processes: ");

    scanf("%d",&n);
```

```
 // b is array for burst time, p for priority and index for process id

int b[n],p[n],index[n];

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

    printf("Enter Burst Time and Priority Value for Process %d: ",i+1);

    scanf("%d %d",&b[i],&p[i]);

    index[i]=i+1;

}

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

    int a=p[i],m=i;

     //Finding out highest priority element and placing it at its desired position

    for(int j=i;j<n;j++)  {

        if(p[j] > a)  {

            a=p[j];

            m=j;

        }

    }

     //Swapping processes

    swap(&p[i], &p[m]);

    swap(&b[i], &b[m]);

    swap(&index[i],&index[m]);

}
```

```
     // T stores the starting time of process

    int t=0;

     //Printing scheduled process

    printf("Order of process Execution is\n");

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

        printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);

        t+=b[i];

    }

    printf("\n");

    printf("Process Id     Burst Time   Wait Time    TurnAround Time\n");

    int wait_time=0;

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

        printf("P%d          %d          %d          %d\n",index[i],b[i],wait_time,wait_time + b[i]);

        wait_time += b[i];

    }

    return 0;

}
```

OUTPUT:-

WEEK- 2

Write programs using the I/O system calls of UNIX/LINUX operating system (open, read,write, close, fcntl, seek, stat, opendir, readdir)

//C program for OPEN System Call

```
#include <fcntl.h>

#include <unistd.h>

int main() {

   int fd = open("file.txt", O_RDONLY);

   if (fd == -1) {

         printf(“No Such File Exist”);

      // Handle error

   }

   // Perform operations on the file using the file descriptor

   close(fd);

   return 0;

}
```

OUTPUT:-

// C program to illustrate close system Call

```
#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

int main() {

int fd1 = open("foo.txt", O_RDONLY);

if (fd1 < 0) { perror("c1");

exit(1); }

printf("opened the fd = % d\n", fd1);

// Using close system Call

if (close(fd1) < 0) {

perror("c1");

exit(1);

}

printf("closed the fd.\n");

}
```

OUTPUT:-

//C program for Write and Read System Call

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

int main() {

// Open a source file for reading

int source_fd = open("source.txt", O_RDONLY);

if (source_fd == -1) {

perror("Failed to open source.txt");

exit(1);

}

// Create or open a destination file for writing

int dest_fd = open("destination.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);

if (dest_fd == -1) {

perror("Failed to open destination.txt");

close(source_fd); // Close the source file

exit(1);

}

// Read from the source file and write to the destination file

char buffer[4096]; // A buffer to hold data

ssize_t nread;

while ((nread = read(source_fd, buffer, sizeof(buffer))) > 0) {

if (write(dest_fd, buffer, nread) != nread) {
```

```
perror("Write error");
break;
}
}
// Check if there was an error during reading
if (nread < 0) {
perror("Read error");
}
// Close both files
close(source_fd);
close(dest_fd);
return 0;
}
```

OUTPUT:-

//PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEMS (OPENDIR, READDIR, CLOSEDIR)

```
#include<stdio.h>

#include<direct.h>

struct dirent *dptr;

int main(int argc, char *argv[]) {

char buff[100];

DIR *dirp; printf(“\n\n ENTER DIRECTORY NAME”);

scanf(“%s”, buff);

if((dirp=opendir(buff))==NULL) {

printf(“The given directory does not exist”);

exit(1);

}

while(dptr=readdir(dirp)) {

printf(“%s\n”,dptr->d_name);

}

 closedir(dirp);

}
```

OUTPUT:-

WEEK-3

Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention.

```
#include<stdio.h>
int main() {
    int p, c, count = 0, i, j, alc[5][3], max[5][3], need[5][3], safe[5], available[3], done[5], terminate = 0;
  printf("Enter the number of process and resources");
  scanf("%d %d", & p, & c);
    printf("enter allocation of resource of all process %dx%d matrix", p, c);
  for (i = 0; i < p; i++) {
    for (j = 0; j < c; j++) {
      scanf("%d", & alc[i][j]);
    }
  }
  printf("enter the max resource process required %dx%d matrix", p, c);
  for (i = 0; i < p; i++) {
    for (j = 0; j < c; j++) {
      scanf("%d", & max[i][j]);
    }
  }
  printf("enter the  available resource");
  for (i = 0; i < c; i++)
    scanf("%d", & available[i]);

  printf("\n need resources matrix are\n");
  for (i = 0; i < p; i++) {
    for (j = 0; j < c; j++) {
      need[i][j] = max[i][j] - alc[i][j];
      printf("%d\t", need[i][j]);
    }
    printf("\n");
  }
  for (i = 0; i < p; i++) {
    done[i] = 0;
  }
  while (count < p) {
    for (i = 0; i < p; i++) {
      if (done[i] == 0) {
        for (j = 0; j < c; j++) {
          if (need[i][j] > available[j])
            break;
        }
        //when need matrix is not greater then available matrix then if j==c will true
        if (j == c) {
```

```
     safe[count] = i;
     done[i] = 1;
     /* now process get execute release the resources and add them in available resources */
     for (j = 0; j < c; j++) {
      available[j] += alc[i][j];
     }
     count++;
     terminate = 0;
    } else {
     terminate++;
    }
   }
  }
  if (terminate == (p - 1)) {
   printf("safe sequence does not exist");
   break;
  }

 }
 if (terminate != (p - 1)) {
  printf("\n available resource after completion\n");
  for (i = 0; i < c; i++) {
   printf("%d\t", available[i]);
  }
  printf("\n safe sequence are\n");
  for (i = 0; i < p; i++) {
   printf("p%d\t", safe[i]);
  }
 }

 return 0;
}
```

OUTPUT:-

WEEK-4

Write a C program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

Producer-Consumer Problem

```
#include<stdio.h>
void main()
{
        int buffer[10], bufsize, in, out, produce, consume, choice=0;
        in = 0;
        out = 0;
        bufsize = 10;
        while(choice !=3)
        {
                printf(“\n1. Produce \t 2. Consume \t3. Exit”);
                printf(“\nEnter your choice: ”);
                scanf(“%d”, &choice);

                switch(choice) {
                        case 1: if((in+1)%bufsize==out)
                        printf(“\nBuffer is Full”);
                else{
                        printf(“\nEnter the value: “);
                        scanf(“%d”, &produce);
                        buffer[in] = produce;
                        in = (in+1)%bufsize;
                }
                break;
                case 2: if(in == out)
                printf("\nBuffer is Empty");
                else{
                        consume = buffer[out];
                        printf("\nThe consumed value is %d", consume);
                        out = (out+1)%bufsize;
                        }
                        break;
                        }
                }
}
```

OUTPUT:-

WEEK-5

Write C programs to illustrate the following IPC mechanisms

a) Pipes

b) FIFOs

c) Message Queues

d) Shared Memory

a) Pipes

```
#include <stdio.h>
#include<stdlib.h>
#include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";

int main()
{
    char inbuf[MSGSIZE];
    int p[2], i;

    if (pipe(p) < 0)
        exit(1);

    /* continued */
    /* write pipe */
    write(p[1], msg1, MSGSIZE);

    write(p[1], msg2, MSGSIZE);

    write(p[1], msg3, MSGSIZE);


    for (i = 0; i < 3; i++) {

        /* read pipe */

        read(p[0], inbuf, MSGSIZE);

        printf("% s\n", inbuf);

    }

    return 0;

}
```

Output:

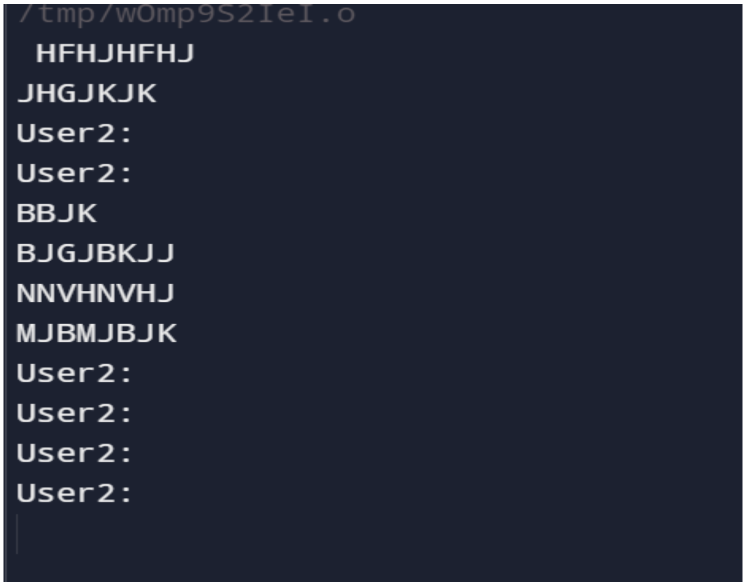
```
hello, world #1

hello, world #2
```

b) FIFOs

WRITE

```
// C program to implement one side of FIFO

// This side writes first, then reads

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <sys/stat.h>
```

```
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;

    // FIFO file path
    char * myfifo = "/tmp/myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>, <permission>)
    mkfifo(myfifo, 0666);

    char arr1[80], arr2[80];
    while (1)
    {
        // Open FIFO for write only
        fd = open(myfifo, O_WRONLY);

        // Take an input arr2ing from user.
        // 80 is maximum length
        fgets(arr2, 80, stdin);
        // Write the input arr2ing on FIFO

        // and close it

        write(fd, arr2, strlen(arr2)+1);

        close(fd);

        // Open FIFO for Read only

        fd = open(myfifo, O_RDONLY);

        // Read from FIFO

        read(fd, arr1, sizeof(arr1));

        // Print the read message

        printf("User2: %s\n", arr1);

        close(fd);

    }

    return 0;

}
```



```
/tmp/wOmp9S2IeI.o
 HFHJHFHJ
JHGJKJK
User2:
User2:
BBJK
BJGJBKJJ
NNVHNVHJ
MJBMJBJK
User2:
User2:
User2:
User2:
```

READ

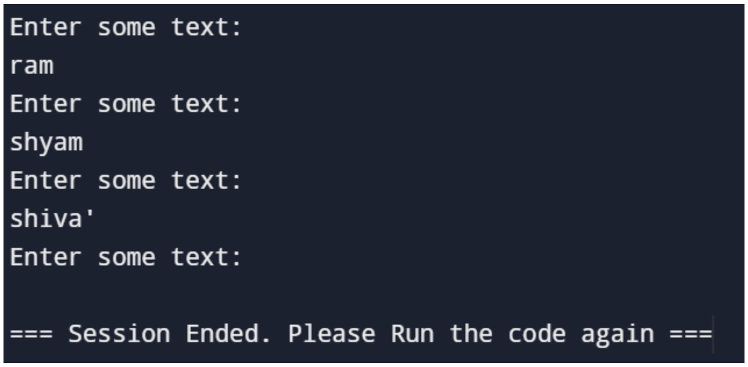
```
  // C program to implement one side of FIFO
// This side reads first, then reads
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd1;

    // FIFO file path
    char * myfifo = "/tmp/myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>,<permission>)
    mkfifo(myfifo, 0666);

    char str1[80], str2[80];
    while (1)
    {
        // First open in read only and read
        fd1 = open(myfifo,O_RDONLY);
        read(fd1, str1, 80);

        // Print the read string and close
        printf("User1: %s\n", str1);
        close(fd1);

        // Now open in write mode and write
        // string taken from user.
        fd1 = open(myfifo,O_WRONLY);
        fgets(str2, 80, stdin);
        write(fd1, str2, strlen(str2)+1);
        close(fd1);
    }
   return 0;
}
```

c) Message Queues

Write

```
#include<stdlib.h>
 #include<stdio.h>
 #include<string.h>
 #include<unistd.h>
 #include<sys/types.h>
 #include<sys/ipc.h>
 #include<sys/msg.h>
 #define MAX_TEXT 512   //maximum length of the message that can be sent allowed
 struct my_msg{
        long int msg_type;
        char some_text[MAX_TEXT];
 };
 int main()
 {
        int running=1;
        int msgid;
        struct my_msg some_data;
        char buffer[50]; //array to store user input
        msgid=msgget((key_t)14534,0666|IPC_CREAT);
        if (msgid == -1) // -1 means the message queue is not created
        {
```

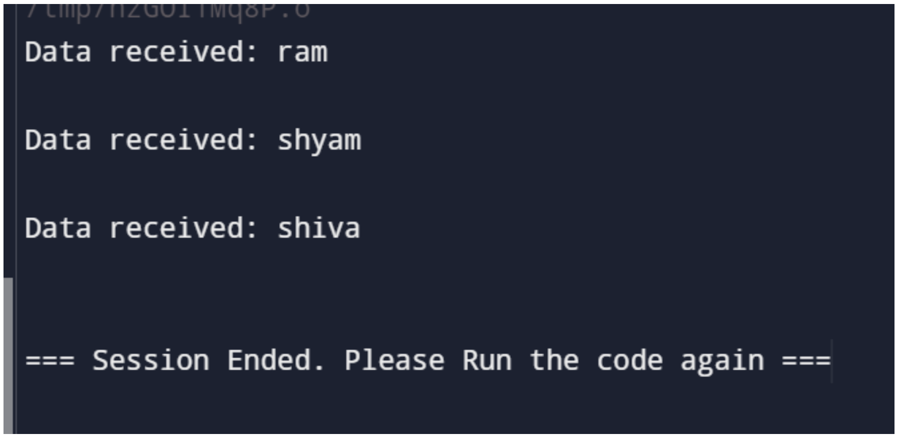
```
        printf("Error in creating queue\n");

        exit(0);

    }

    while(running)

    {

        printf("Enter some text:\n");

        fgets(buffer,50,stdin);

        some_data.msg_type=1;

        strcpy(some_data.some_text,buffer);

        if(msgsnd(msgid,(void *)&some_data, MAX_TEXT,0)==-1)

// msgsnd returns -1 if the message is not sent

        {

            printf("Msg not sent\n");

        }

        if(strncmp(buffer,"end",3)==0)

        {

            running=0;

        }    }    }
```



```
Enter some text:
ram
Enter some text:
shyam
Enter some text:
shiva'
Enter some text:

=== Session Ended. Please Run the code again ===
```

Read

```
#include<stdlib.h>
 #include<stdio.h>
 #include<string.h>
 #include<unistd.h>
 #include<sys/types.h>
 #include<sys/ipc.h>
 #include<sys/msg.h>
 struct my_msg{
        long int msg_type;
        char some_text[BUFSIZ];
 };
 int main()
 {
        int running=1;
        int msgid;
        struct my_msg some_data;
        long int msg_to_rec=0;
        msgid=msgget((key_t)14534,0666|IPC_CREAT);
        while(running)
        {
                msgrcv(msgid,(void *)&some_data,BUFSIZ,msg_to_rec,0);
                printf("Data received: %s\n",some_data.some_text);
                if(strncmp(some_data.some_text,"end",3)==0)
                {
                    running=0;
                }
        }
        msgctl(msgid,IPC_RMID,0);
 }
```

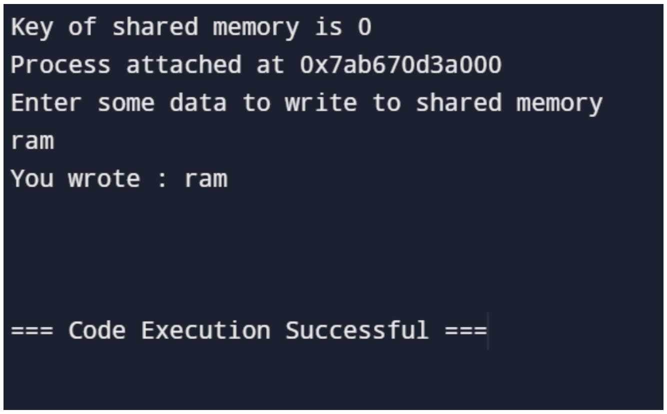


```
/tmp/nZG01TMq8P.o
Data received: ram

Data received: shyam

Data received: shiva


=== Session Ended. Please Run the code again ===
```

d) Shared Memory

Shared Memory (write)

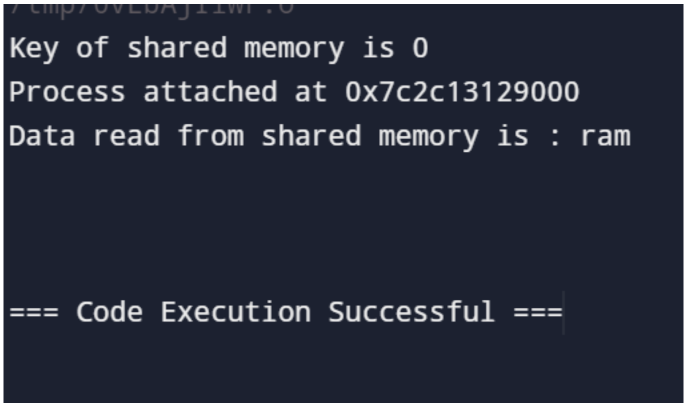
```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/shm.h>

#include<string.h>

int main()

{

int i;

void *shared_memory;

char buff[100];

int shmid;

shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);

printf("Key of shared memory is %d\n",shmid);

shared_memory=shmat(shmid,NULL,0);

printf("Process attached at %p\n",shared_memory);

printf("Enter some data to write to shared memory\n");

read(0,buff,100); //get some input from user

strcpy(shared_memory,buff);

printf("You wrote : %s\n",(char *)shared_memory);

}
```



```
Key of shared memory is 0
Process attached at 0x7ab670d3a000
Enter some data to write to shared memory
ram
You wrote : ram



=== Code Execution Successful ===
```

Shared Memory (Read)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
int i;
void *shared_memory;
char buff[100];
int shmid;
shmid=shmget((key_t)2345, 1024, 0666);
printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0); //process attached to shared memory segment
printf("Process attached at %p\n",shared_memory);

printf("Data read from shared memory is : %s\n",(char *)shared_memory);

}
```



```
/tmp/0VLbAjI1Wr.o
Key of shared memory is 0
Process attached at 0x7c2c13129000
Data read from shared memory is : ram



=== Code Execution Successful ===
```

Week – 06

Write a C program to simulate the following techniques of memory management

a) Paging

b) Segmentation

a) Paging

```
#include<stdio.h>

int main()

{

 int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;

 int s[10], fno[10][20];

printf("\nEnter the memory size -- ");

scanf("%d",&ms);

printf("\nEnter the page size -- ");

scanf("%d",&ps);


nop = ms/ps;

printf("\nThe no. of pages available in memory are -- %d ",nop);


printf("\nEnter number of processes -- ");

 scanf("%d",&np);

rempages = nop;

for(i=1;i<=np;i++)
```

```
{
printf("\nEnter no. of pages required for p[%d]-- ",i);
 scanf("%d",&s[i]);

if(s[i] >rempages)
{
printf("\nMemory is Full");
break;
}
rempages = rempages - s[i];

printf("\nEnter pagetable for p[%d] --- ",i);
 for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]);
}
printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset -- ");
scanf("%d %d %d",&x,&y, &offset);
if(x>np || y>=s[i] || offset>=ps)
printf("\nInvalid Process or Page Number or offset");
else
{ pa=fno[x][y]*ps+offset;
printf("\nThe Physical Address is -- %d",pa);
}

}
```

INPUT

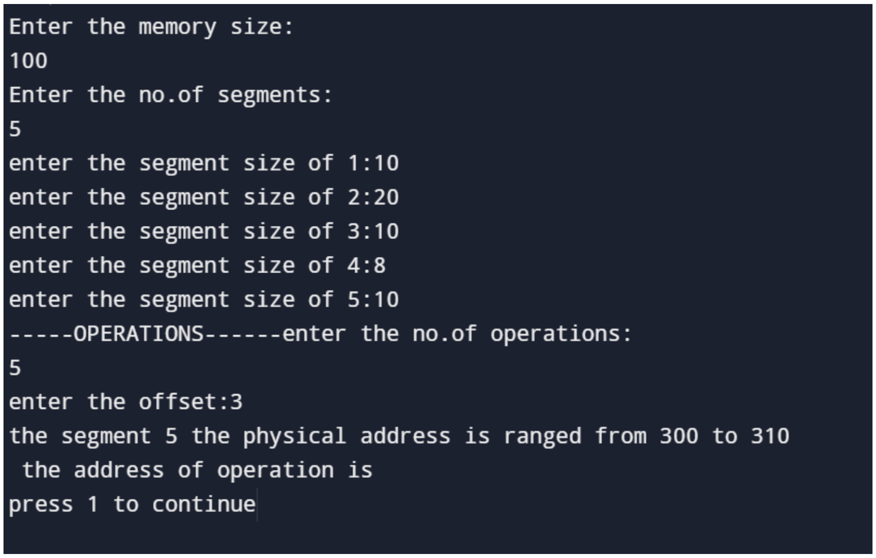
```
Enter the memory size – 1000 Enter the page size -- 100
The no. of pages available in memory are -- 10
Enter number of processes -- 3
Enter no. of pages required for p[1]-- 4
Enter pagetable for p[1] --- 8 6
9
5
Enter no. of pages required for p[2]-- 5
Enter pagetable for p[2] --- 1 4 5 7 3
Enter no. of pages required for p[3]-- 5
```

OUTPUT

```
Memory is Full
Enter Logical Address to find Physical Address Enter process no. and pagenumber and
offset  -- 2
3
60
The Physical Address is --  760
```

b) Segmentation

```
#include <stdio.h>

int main()

{

    int n,nm,p,x=0,y=1,t=300,of,i;

    printf("Enter the memory size:\n");

    scanf("%d",&nm);

    printf("Enter the no.of segments:\n");

    scanf("%d",&n);

    int s[n];

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

    {

        printf("enter the segment size of %d:",i+1);

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

        x+=s[i];

        if(x>nm)

        {

            printf("memory full segment %d is not allocated",i+1);

            x-=s[i];

            s[i]=0;

        }

    }

    printf("-----OPERATIONS------");
    while(y==1)
    {
        printf("enter the no.of operations:\n");
        scanf("%d",&p);
        printf("enter the offset:");
        scanf("%d",&of);
        if(s[p-1]==0)
        {
            printf("segment is not allocated\n");
        }
        else if(of>s[p-1])
        {
            printf("out of range!..");
        }
        else
        {
            printf("the segment %d the physical address is ranged from %d to %d\n the address of operation is\n",p,t,t+s[p-1],t+of);
        }
        printf("press 1 to continue");
        scanf("%d",&y);
    }
}
```



```
Enter the memory size:
100
Enter the no.of segments:
5
enter the segment size of 1:10
enter the segment size of 2:20
enter the segment size of 3:10
enter the segment size of 4:8
enter the segment size of 5:10
-----OPERATIONS------enter the no.of operations:
5
enter the offset:3
the segment 5 the physical address is ranged from 300 to 310
 the address of operation is
press 1 to continue
```

Week-07

Write a C program to simulate Page Replacement Policies

a). FCFS

b).LRU

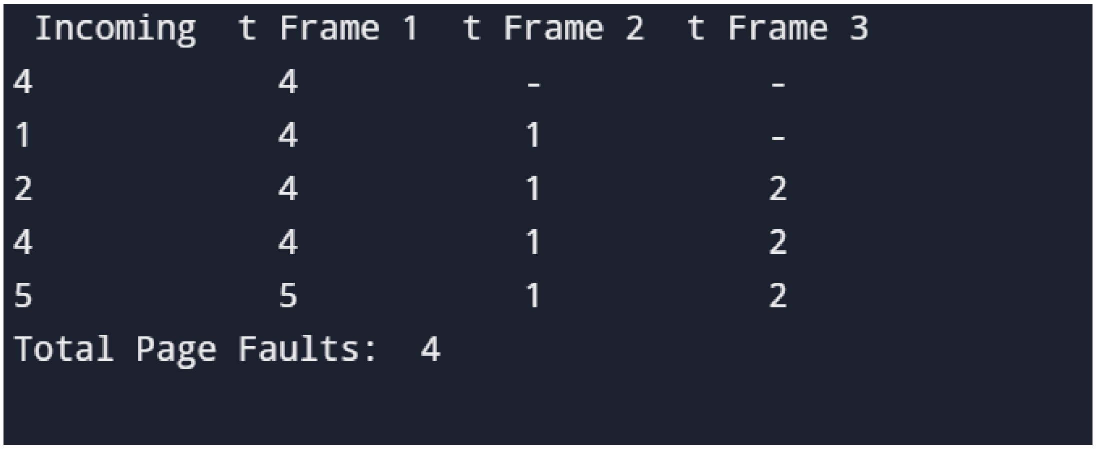
c). Optimal

a). FCFS

FIFO

```
#include<stdio.h>

int main()

{

   int incomingStream[] = {4 , 1 , 2 , 4 , 5};

   int pageFaults = 0;

   int frames = 3;

   int m, n, s, pages;

   pages = sizeof(incomingStream)/sizeof(incomingStream[0]);

   printf( " Incoming \ t Frame 1 \ t Frame 2 \ t Frame 3 " ) ;

   int temp[ frames ];

   for(m = 0; m < frames; m++)

   {

      temp[m] = -1;

   }

   for(m = 0; m < pages; m++)

   {

      s = 0;
```

```
for(n = 0; n < frames; n++)
{
    if(incomingStream[m] == temp[n])
    {
        s++;
        pageFaults--;
    }
}
pageFaults++;
if((pageFaults <= frames) && (s == 0))
{
    temp[m] = incomingStream[m];
}
else if(s == 0)
{
    temp[(pageFaults - 1) % frames] = incomingStream[m];
}
printf("\n");
printf("%d\t\t\t",incomingStream[m]);
for(n = 0; n < frames; n++)
{
    if(temp[n] != -1)
        printf(" %d\t\t\t", temp[n]);
    else
        printf(" - \t\t\t");
        }
    }
    printf("\nTotal Page Faults:\t%d\n", pageFaults);
    return 0;
}
```



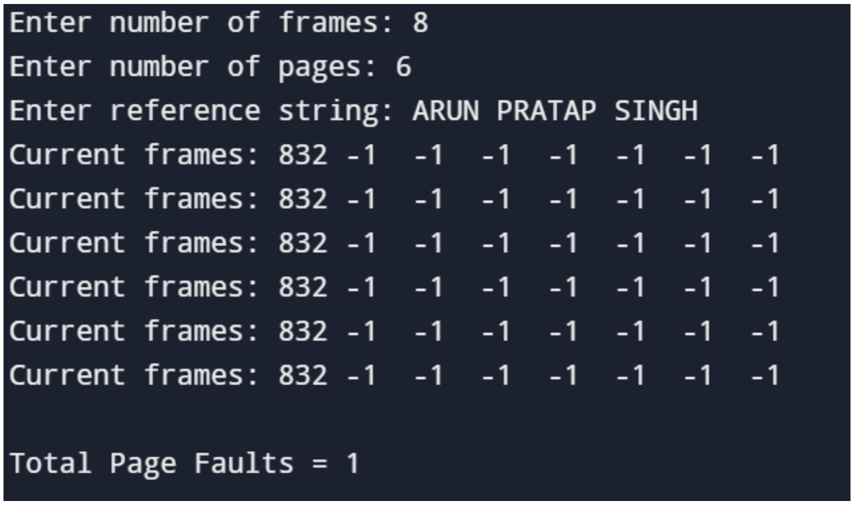
```
 Incoming  t Frame 1  t Frame 2  t Frame 3
4            4           -           -
1            4           1           -
2            4           1           2
4            4           1           2
5            5           1           2
Total Page Faults:  4
```

b).LRU

```
#include <stdio.h>


// Function to find the index of the least recently used page in frames

int findLRU(int time[], int n) {

    int i, minimum = time[0], pos = 0;


    for (i = 1; i < n; ++i) {

        if (time[i] < minimum) {

            minimum = time[i];

            pos = i;

        }

    }

    return pos;

}

  int main() {

    int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], i, j, pos, faults = 0;


    printf("Enter number of frames: ");

    scanf("%d", &no_of_frames);


    printf("Enter number of pages: ");

    scanf("%d", &no_of_pages);


    printf("Enter reference string: ");
```

```
for (i = 0; i < no_of_pages; ++i) {
    scanf("%d", &pages[i]);
}

for (i = 0; i < no_of_frames; ++i) {
    frames[i] = -1;
    time[i] = 0; // Initialize the time array to 0
}

for (i = 0; i < no_of_pages; ++i) {
    int page = pages[i];
    int page_found = 0;

    // Check if the page is already in frames
    for (j = 0; j < no_of_frames; ++j) {
        if (frames[j] == page) {
            time[j] = counter++; // Update the time of use
            page_found = 1;
            break;
        }
    }

    // If the page is not in frames, find the LRU page to replace
    if (!page_found) {
        pos = findLRU(time, no_of_frames);
```

```
            frames[pos] = page; // Replace the LRU page
            time[pos] = counter++; // Update the time of use
            faults++;
        }

        // Print the current state of frames
        printf("Current frames: ");
        for (j = 0; j < no_of_frames; ++j) {
            printf("%d\t", frames[j]);
        }
        printf("\n");
    }

    printf("\nTotal Page Faults = %d\n", faults);

    return 0;
}
```

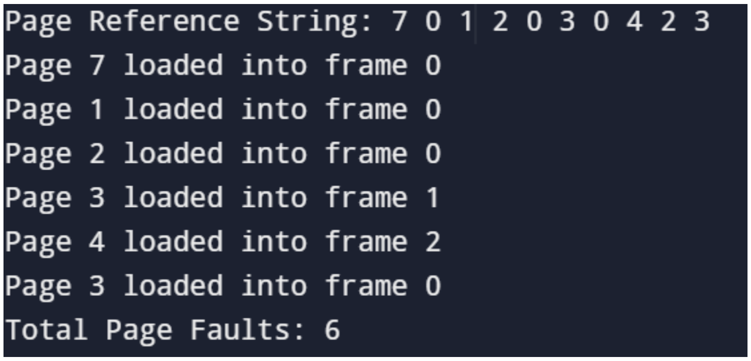


```
Enter number of frames: 8
Enter number of pages: 6
Enter reference string: ARUN PRATAP SINGH
Current frames: 832 -1   -1   -1   -1   -1   -1   -1
Current frames: 832 -1   -1   -1   -1   -1   -1   -1
Current frames: 832 -1   -1   -1   -1   -1   -1   -1
Current frames: 832 -1   -1   -1   -1   -1   -1   -1
Current frames: 832 -1   -1   -1   -1   -1   -1   -1
Current frames: 832 -1   -1   -1   -1   -1   -1   -1

Total Page Faults = 1
```

c). Optimal

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


#define NUM_FRAMES 3

#define NUM_PAGES 10


// Function to find the page that will be referenced furthest in the future

int findOptimalPage(int page[], int pageFrames[], int index, int numFrames) {

    int farthest = -1;

    int farthestIndex = -1;


    for (int i = 0; i < numFrames; i++) {

        int j;

        for (j = index; j < NUM_PAGES; j++) {

            if (pageFrames[i] == page[j]) {


if (j > farthest) {


farthest = j;


farthestIndex = i;


}
break;
            }
        }

        if (j == NUM_PAGES) {

return i;
        }
    }

    if (farthestIndex == -1) {
        return 0;
    }

    return farthestIndex;
}

int main() {
    int pageReferences[NUM_PAGES] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3};
    int pageFrames[NUM_FRAMES];
    bool isPageInFrame[NUM_FRAMES];
    int pageFaults = 0;

    for (int i = 0; i < NUM_FRAMES; i++) {
```

```
pageFrames[i] = -1;

isPageInFrame[i] = false;
    }


printf("Page Reference String: ");
    for (int i = 0; i < NUM_PAGES; i++) {

printf("%d ", pageReferences[i]);
    }
    printf("\n");

    for (int i = 0; i < NUM_PAGES; i++) {
        int page = pageReferences[i];

        if (!isPageInFrame[page]) {
            int pageToReplace = findOptimalPage(pageReferences, pageFrames, i + 1,
NUM_FRAMES);

pageFrames[pageToReplace] = page;

isPageInFrame[pageToReplace] = true;
pageFaults++;



printf("Page %d loaded into frame %d\n", page, pageToReplace);
        }
    }



printf("Total Page Faults: %d\n", pageFaults);

    return 0;
}
```



```
Page Reference String: 7 0 1 2 0 3 0 4 2 3
Page 7 loaded into frame 0
Page 1 loaded into frame 0
Page 2 loaded into frame 0
Page 3 loaded into frame 1
Page 4 loaded into frame 2
Page 3 loaded into frame 0
Total Page Faults: 6
```