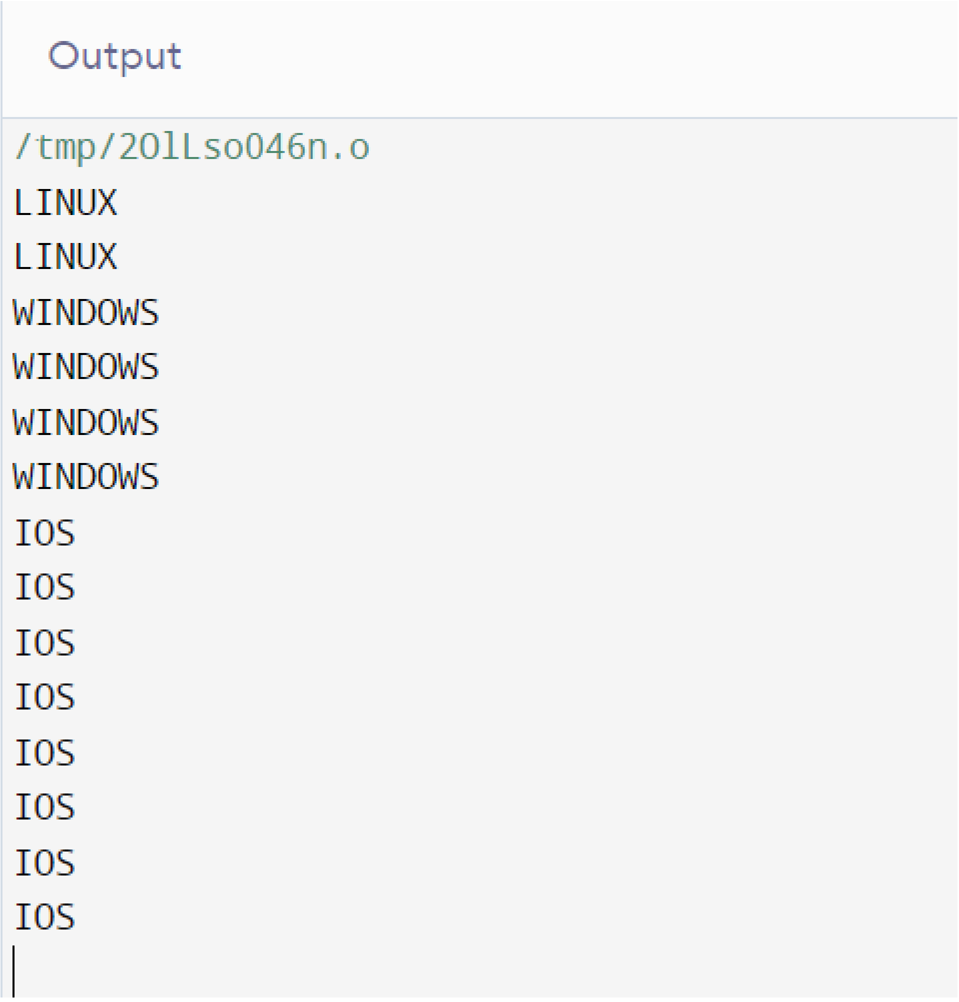
Q1. Demonstration of FORK() System Call.

Code:

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    fork();
    printf("LINUX\n");
    fork();
    printf("WINDOWS\n");
    fork();
    printf("IOS\n");
    return 0;
}
```

*****OUTPUT*****



```
/tmp/2OlLso046n.o
LINUX
LINUX
WINDOWS
WINDOWS
WINDOWS
WINDOWS
IOS
IOS
IOS
IOS
IOS
IOS
IOS
IOS
```

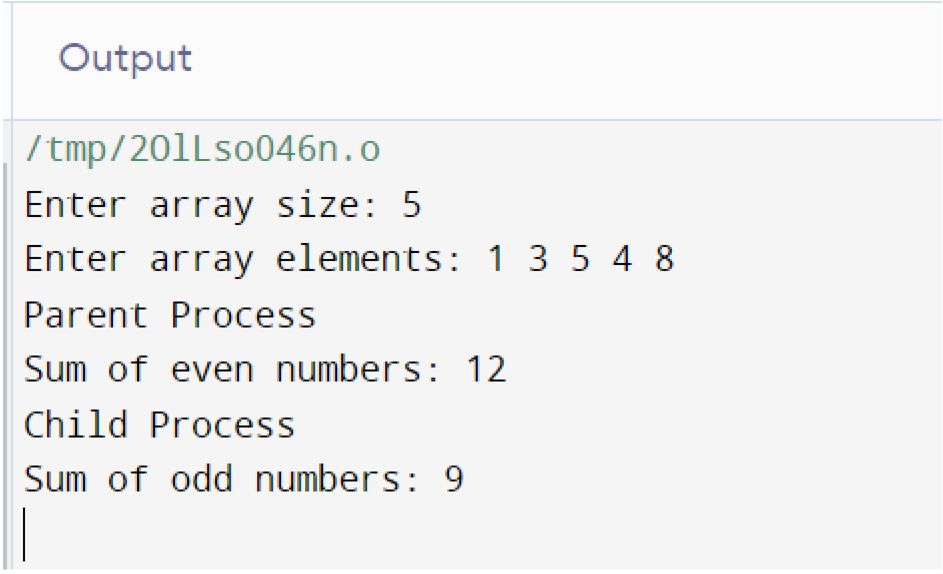
Q2. Parent Process Computes the sum of EVEN And Child Processes Computes the sum Of ODD numbers using FORK().

Code:

```
#include <stdio.h>
#include <unistd.h>
int main()
{
   int s;
   printf("Enter array size: ");
   scanf("%d",&s);
   int a[s];
   printf("Enter array elements: ");
   for(int x=0;x<s;x++)
   {
      scanf("%d",&a[x]);
   }
   int n=fork();
   int odd_s=0, even_s=0, i;
   if(n>0)
   {
      for(int i=0;i<s;i++)
      {
         if(a[i]%2==0)
         {
            even_s+=a[i];
         }
      }
      printf("Parent Process\n");
      printf("Sum of even numbers: %d\n",even_s);
```

```
    }
    else
    {
        for(int i=0;i<s;i++)
        {
            if(a[i]%2!=0)
            {
                odd_s+=a[i];
            }
        }
        printf("Child Process\n");
        printf("Sum of odd numbers: %d\n",odd_s);
    }
    return 0;
}
```

*****OUTPUT*****



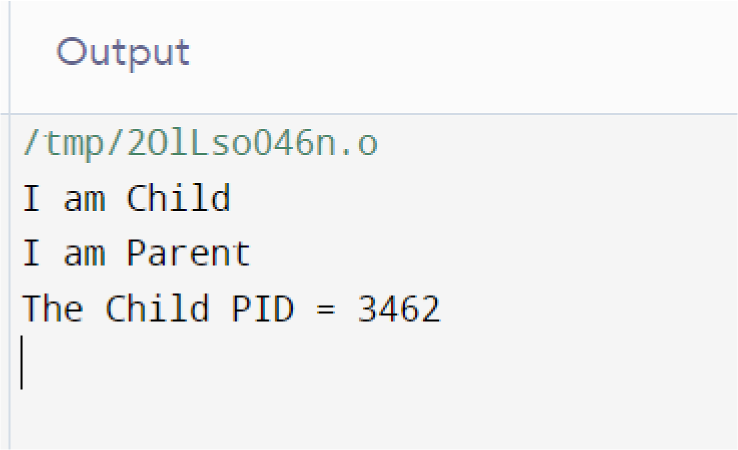
```
/tmp/2OlLso046n.o
Enter array size: 5
Enter array elements: 1 3 5 4 8
Parent Process
Sum of even numbers: 12
Child Process
Sum of odd numbers: 9
```

Q3. Demonstration of WAIT() System Call.

Code:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
int main()
{
   pid_t pid;
   pid=fork();
   if(pid==0)
   {
      printf("I am Child\n");
      exit(0);
   }
   else
   {
      wait(NULL);
      printf("I am Parent\n");
      printf("The Child PID = %d\n",pid);
   }
   return 0;
}
```

*****OUTPUT*****



```
/tmp/2OlLso046n.o
I am Child
I am Parent
The Child PID = 3462
```

Q4. Implementation of ORPHAN PROCESS & ZOMBIE PROCESS.

Code:

ORPHAN

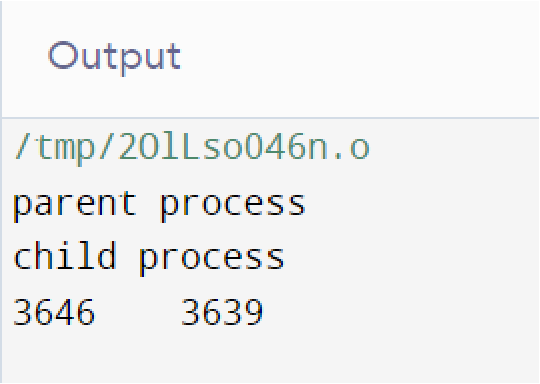
```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
int main()
{
   pid_t id;
   id=fork();
   if(id>0)
   {
      printf("parent process\n");
      printf("%d\t%d\n",getpid(),getppid());
      exit(0);
      printf("Error");
   }
   else if(id==0)
   {
      printf("child process\n");
      sleep(50);
      printf("%d\t%d\n",getpid(),getppid());
   }
}
```

ZOMBIE

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
int main()
{
   pid_t id;
   id=fork();
   if(id>0)
   {
      sleep(50);
      printf("parent process\n");
      printf("%d\t%d\n",getpid(),getppid());
   }
   else if(id==0)
   {
      printf("child process\n");
      printf("%d\t%d\n",getpid(),getppid());
   }
}
```

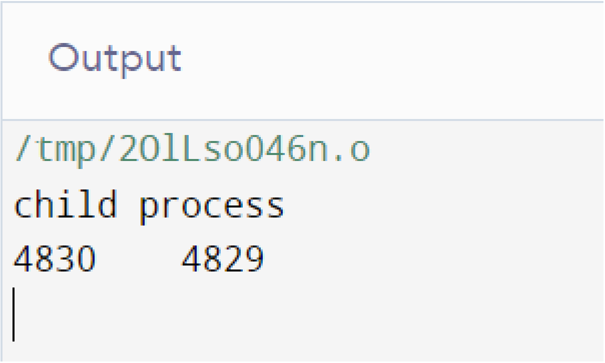
*****OUTPUT*****

ORPHAN:



```
/tmp/2OlLso046n.o
parent process
child process
3646    3639
```

ZOMBIE:



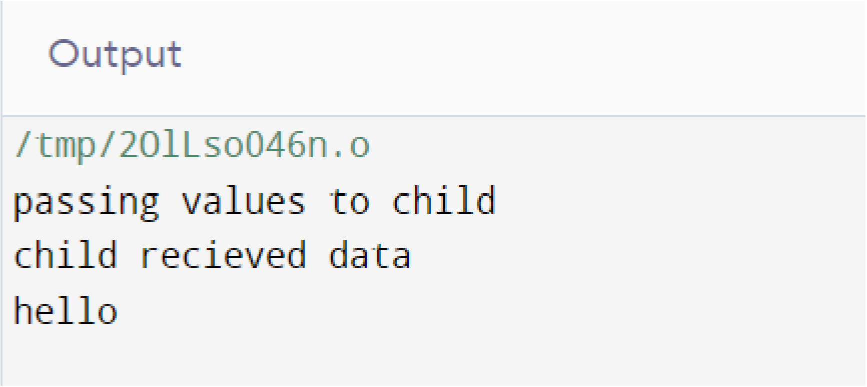
```
/tmp/2OlLso046n.o
child process
4830    4829
```

Q5. Implementation of PIPE.

Code:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main()
{
   int fd[2],n;
   char buffer[100];
   pid_t p;
   pipe(fd);
   p=fork();
   if(p>0)
   {
      printf("passing values to child\n");
      write(fd[1],"hello\n",6);
   }
   else
   {
      printf("child recieved data\n");
      n=read(fd[0],buffer,100);
      write(1,buffer,n);
   }
}
```

*****OUTPUT*****



```
/tmp/2OlLso046n.o
passing values to child
child recieved data
hello
```

Q6. Implementation of FIFO.

Code:

WRITER PROCESS

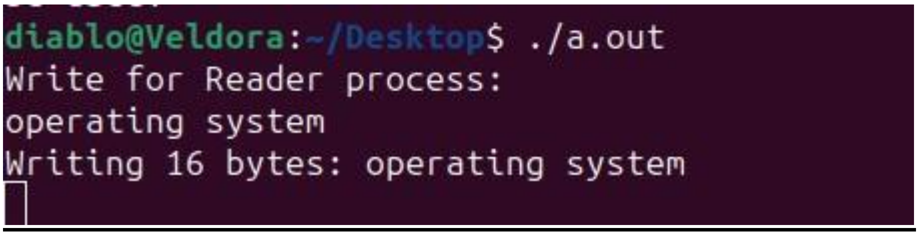
```
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
int main()
{
   int file,n;
   char s[100];
   mknod("myfifo",S_IFIFO|0666,0);
   printf("Write for Reader process:\n");
   file=open("myfifo",O_WRONLY);
   while(gets(s))
   {
      n=write(file,s,strlen(s));
      printf("Writing %d bytes: %s\n",n,s);
   }
   return 0;
}
```

READER PROCESS

```
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
int main()
{
   int file,n;
   char a[100];
   mknod("myfifo",S_IFIFO|0666,0);
   file=open("myfifo",O_RDONLY);
   printf("If you have a write process then type data\n");
   do
   {
      n=read(file,a,sizeof(a));
      a[n]='\0';
      printf("Reader process read %d bytes: %s\n",n,a);
   }while(n>0);
   return 0;
}
```

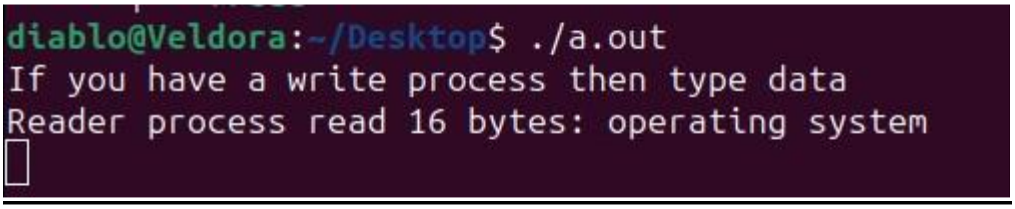
*****OUTPUT*****

WRITER PROCESS:



```
diablo@Veldora:~/Desktop$ ./a.out
Write for Reader process:
operating system
Writing 16 bytes: operating system
```

READER PROCESS:



```
diablo@Veldora:~/Desktop$ ./a.out
If you have a write process then type data
Reader process read 16 bytes: operating system
```

Q7. Implementation of MESSAGE QUEUE.

Code:

WRITER PROCESS

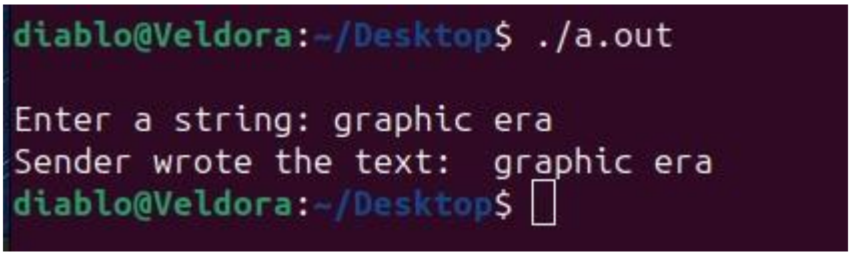
```
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>

struct msgbuff
{
   long mtype;
   char mtext[100];
}mb;

int main()
{
   key_t key;
   int msgid,c;
   key=ftok("progfile",'A');
   msgid=msgget(key,0666|IPC_CREAT);
   mb.mtype=1;
   printf("\nEnter a string: ");
   gets(mb.mtext);
   c=msgsnd(msgid,&mb,strlen(mb.mtext),0);
   printf("Sender wrote the text:\t%s\n",mb.mtext);
   return 0;
}
```

READER PROCESS

```
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>

struct msgbuff
{
   long mtype;
   char mtext[100];
}mb;

int main()
{
   key_t key;
   int msgid,c;
   key=ftok("progfile",'A');
   msgid=msgget(key,0666|IPC_CREAT);
   msgrcv(msgid,&mb,sizeof(mb),1,0);
   printf("Data Received is:\t%s\n",mb.mtext);
   msgctl(msgid,IPC_RMID,NULL);
   return 0;
}
```

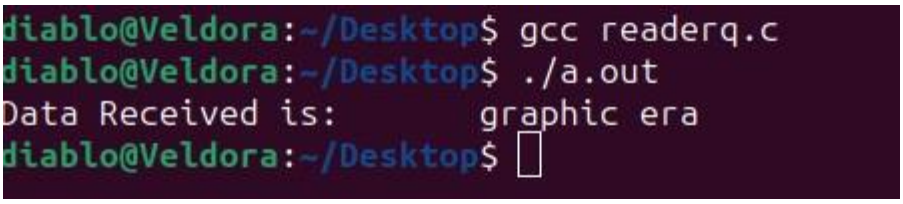
*****OUTPUT*****

WRITER PROCESS:



```
diablo@Veldora:~/Desktop$ ./a.out

Enter a string: graphic era
Sender wrote the text:  graphic era
diablo@Veldora:~/Desktop$ 
```

READER PROCESS:



```
diablo@Veldora:~/Desktop$ gcc readerq.c
diablo@Veldora:~/Desktop$ ./a.out
Data Received is:       graphic era
diablo@Veldora:~/Desktop$ 
```

Q8. Implementation of SHARED MEMORY.

Code:

WRITER PROCESS

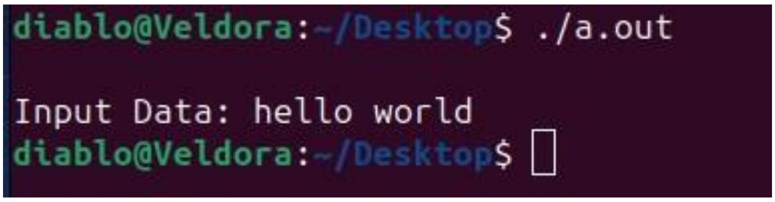
```
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

int main()
{
   key_t key;
   int shmid;
   void *ptr;
   key=ftok("shmfile",'A');
   shmid=shmget(key,1024,0666|IPC_CREAT);
   ptr=shmat(shmid,(void *)0,0);
   printf("\nInput Data: ");
   gets(ptr);
   shmdt(ptr);
   return 0;
}
```

READER PROCESS

```
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

int main()
{
   key_t key;
   int shmid;
   void *ptr;
   key=ftok("srfile",'A');
   shmid=shmget(key,1024,0666|IPC_CREAT);
   ptr=shmat(shmid,(void *)0,0);
   printf("\nThe data stored is: %s\n",ptr);
   shmdt(ptr);
   shmctl(shmid,IPC_RMID,NULL);
   return 0;
}
```

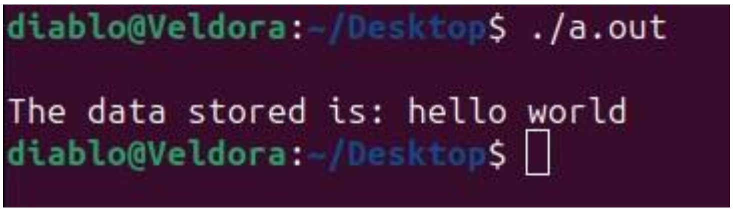
*****OUTPUT*****

WRITER PROCESS:



```
diablo@Veldora:~/Desktop$ ./a.out

Input Data: hello world
diablo@Veldora:~/Desktop$ 
```

READER PROCESS:



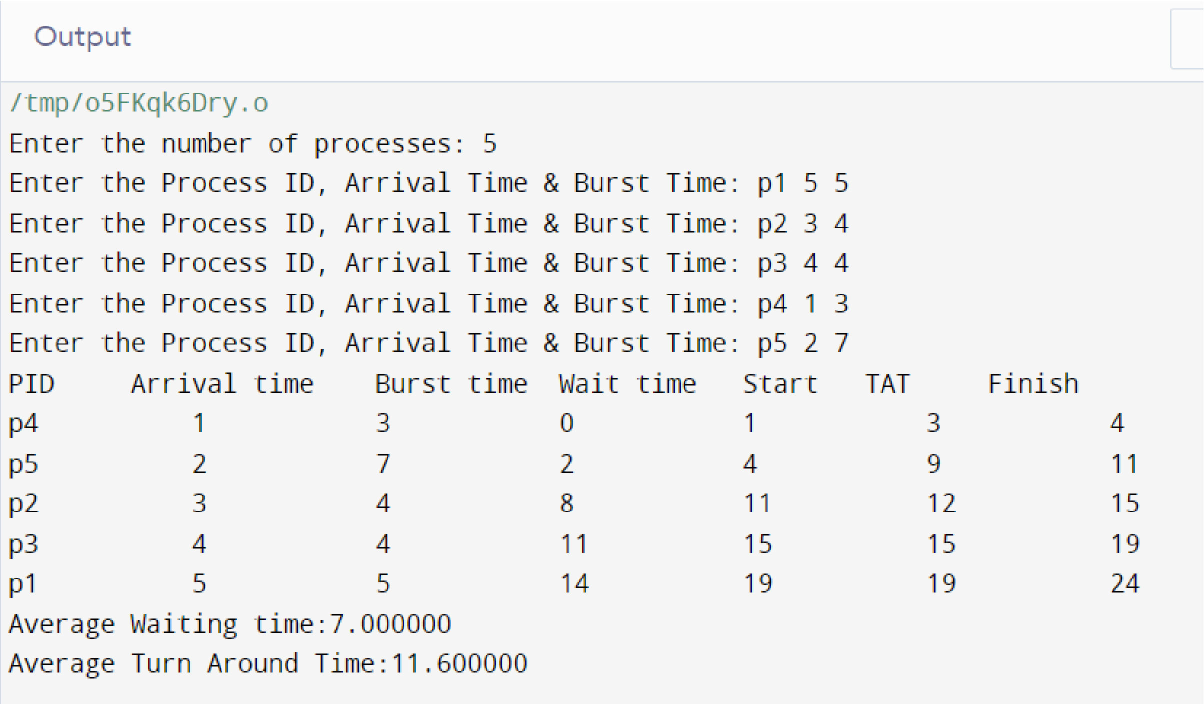
```
diablo@Veldora:~/Desktop$ ./a.out

The data stored is: hello world
diablo@Veldora:~/Desktop$ 
```

Q9. Implementation of the First Come First Served (FCFS) Scheduling Algorithm.

Code:

```
#include<stdio.h>
#include<string.h>
int main()
{
    char pn[10][10],t[10];
    int arr[10],bur[10],star[10],finish[10],tat[10],wt[10],i,j,n,temp;
    int totwt=0,tottat=0;
    printf("Enter the number of processes: ");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("Enter the Process ID, Arrival Time & Burst Time: ");
        scanf("%s%d%d",&pn[i],&arr[i],&bur[i]);
    }
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            if(arr[i]<arr[j])
            {
                temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
                temp=bur[i];
                bur[i]=bur[j];
                bur[j]=temp;
                strcpy(t,pn[i]);
                strcpy(pn[i],pn[j]);
                strcpy(pn[j],t);
            }

        }
    }
    for(i=0; i<n; i++)
    {
        if(i==0)
            star[i]=arr[i];
        else
            star[i]=finish[i-1];
        wt[i]=star[i]-arr[i];
        finish[i]=star[i]+bur[i];
        tat[i]=finish[i]-arr[i];
    }
    printf("\nPID \tArrival time\tBurst time\tWait time\tStart \tTAT \tFinish");
    for(i=0; i<n; i++)
    {
printf("\n%s\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d",pn[i],arr[i],bur[i],wt[i],star[i],tat[i]
,finish[i]);
        totwt+=wt[i];
        tottat+=tat[i];
    }
    printf("\nAverage Waiting time:%f",(float)totwt/n);
    printf("\nAverage Turn Around Time:%f",(float)tottat/n);
    return 0;
}
```

*******OUTPUT*******



```
/tmp/o5FKqk6Dry.o
Enter the number of processes: 5
Enter the Process ID, Arrival Time & Burst Time: p1 5 5
Enter the Process ID, Arrival Time & Burst Time: p2 3 4
Enter the Process ID, Arrival Time & Burst Time: p3 4 4
Enter the Process ID, Arrival Time & Burst Time: p4 1 3
Enter the Process ID, Arrival Time & Burst Time: p5 2 7
PID     Arrival time    Burst time  Wait time   Start   TAT     Finish
p4          1           3           0           1           3           4
p5          2           7           2           4           9           11
p2          3           4           8           11          12          15
p3          4           4           11          15          15          19
p1          5           5           14          19          19          24
Average Waiting time:7.000000
Average Turn Around Time:11.600000
```

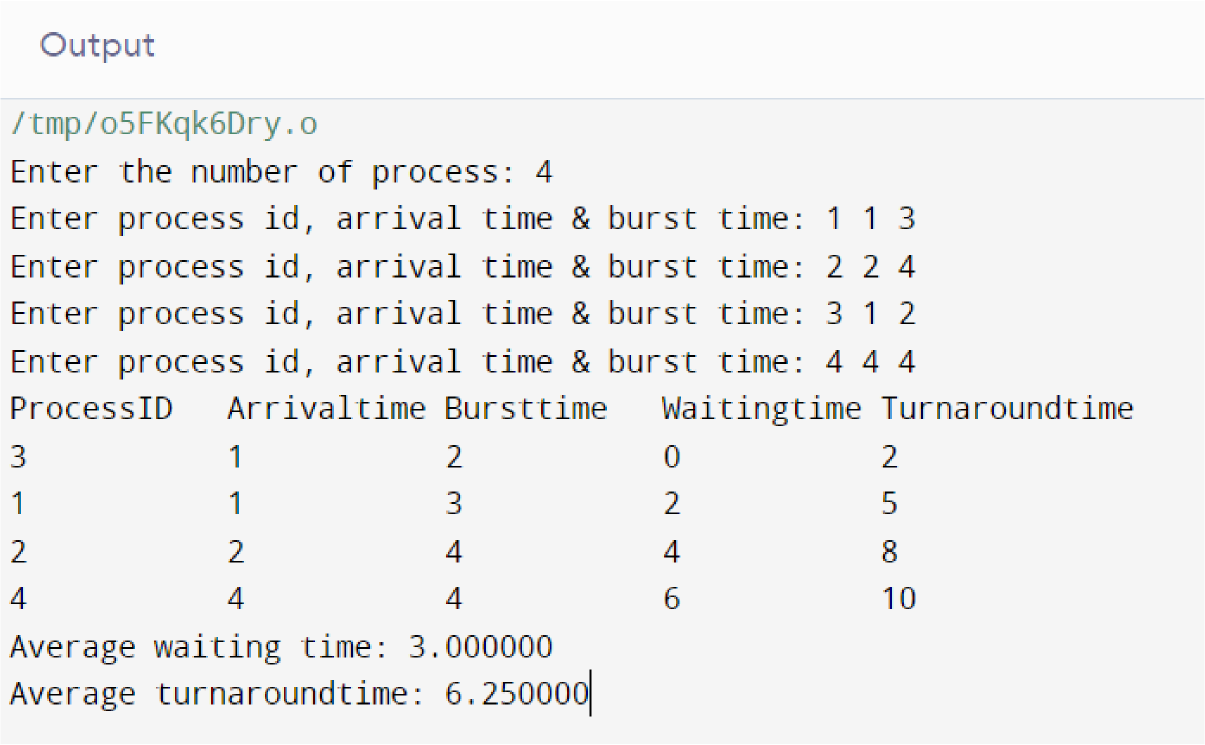
Q10. Implementation of Shortest Job First (SJF) Scheduling Algorithm.

Code:

```
#include<string.h>
int main()
{
   int bt[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
   int totwt=0,totta=0;
   double awt,ata;
   char pn[10][10],t[10];
   printf("Enter the number of process: ");
   scanf("%d",&n);
   for(i=0; i<n; i++)
   {
      printf("Enter process id, arrival time & burst time: ");
      scanf("%s%d%d",pn[i],&at[i],&bt[i]);
   }
   for(i=0; i<n; i++)
      for(j=0; j<n; j++)
      {
         if(bt[i]<bt[j])
         {
            temp=at[i];
            at[i]=at[j];
            at[j]=temp;
            temp=bt[i];
            bt[i]=bt[j];
            bt[j]=temp;
            strcpy(t,pn[i]);
            strcpy(pn[i],pn[j]);
```

```
                strcpy(pn[j],t);
            }
        }
    for(i=0; i<n; i++)
    {
        if(i==0)
            st[i]=at[i];
        else
            st[i]=ft[i-1];
        wt[i]=st[i]-at[i];
        ft[i]=st[i]+bt[i];
        ta[i]=ft[i]-at[i];
        totwt+=wt[i];
        totta+=ta[i];
    }
    awt=(double)totwt/n;
    ata=(double)totta/n;
    printf("\nProcessID\tArrivaltime\tBursttime\tWaitingtime\tTurnaroundtime");
    for(i=0; i<n; i++)
    {
        printf("\n%s\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d",pn[i],at[i],bt[i],wt[i],ta[i]);
    }
    printf("\nAverage waiting time: %f",awt);
    printf("\nAverage turnaroundtime: %f",ata);
    return 0;
}
```

*****OUTPUT*****



Output

```
/tmp/o5FKqk6Dry.o
Enter the number of process: 4
Enter process id, arrival time & burst time: 1 1 3
Enter process id, arrival time & burst time: 2 2 4
Enter process id, arrival time & burst time: 3 1 2
Enter process id, arrival time & burst time: 4 4 4
ProcessID   Arrivaltime Bursttime   Waitingtime Turnaroundtime
3           1           2           0           2
1           1           3           2           5
2           2           4           4           8
4           4           4           6           10
Average waiting time: 3.000000
Average turnaroundtime: 6.250000
```

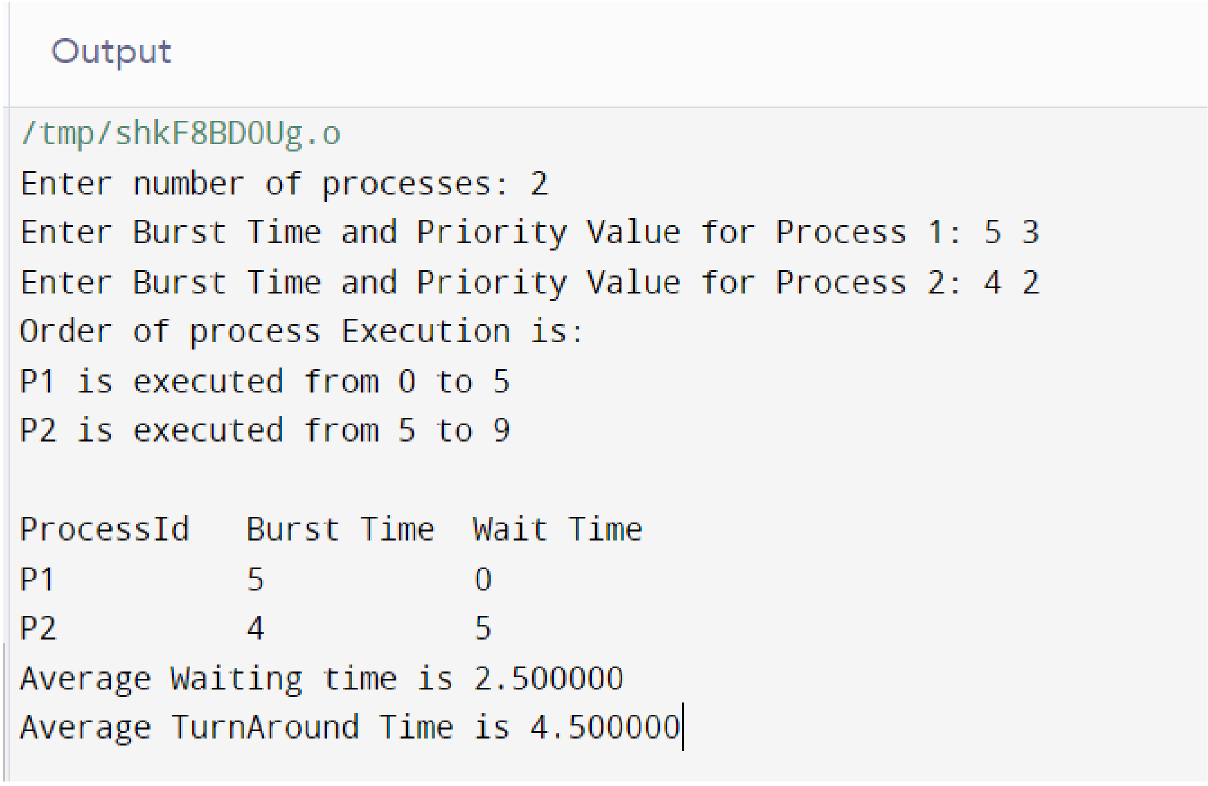
Q11. Implementation of Priority Scheduling Algorithm.

Code:

```
#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);
   int burst[n],priority[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",&burst[i],&priority[i]);
      index[i]=i+1;
   }
   for(int i=0;i<n;i++)
   {
      int temp=priority[i],m=i;

      for(int j=i;j<n;j++)
      {
         if(priority[j] > temp)
         {
            temp=priority[j];
            m=j;
        }
    }

    swap(&priority[i], &priority[m]);
    swap(&burst[i], &burst[m]);
    swap(&index[i],&index[m]);
}
int t=0;
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+burst[i]);
    t+=burst[i];
}
printf("\n");
printf("ProcessId\tBurst Time\tWait Time\n");
int wait_time=0;
int total_wait_time = 0;
for(int i=0;i<n;i++)
{
    printf("P%d\t\t\t%d\t\t\t%d\n",index[i],burst[i],wait_time);
    total_wait_time += wait_time;
    wait_time += burst[i];
}

float avg_wait_time = (float) total_wait_time / n;
printf("Average Waiting time is %f\n", avg_wait_time);
```

```
    int total_Turn_Around = 0;
    for(int i=0; i < n; i++){
        total_Turn_Around += burst[i];
    }
    float avg_Turn_Around = (float) total_Turn_Around / n;
    printf("Average TurnAround Time is %f",avg_Turn_Around);
    return 0;
}
```

*******OUTPUT*******



```
/tmp/shkF8BD0Ug.o
Enter number of processes: 2
Enter Burst Time and Priority Value for Process 1: 5 3
Enter Burst Time and Priority Value for Process 2: 4 2
Order of process Execution is:
P1 is executed from 0 to 5
P2 is executed from 5 to 9

ProcessId   Burst Time   Wait Time
P1          5            0
P2          4            5
Average Waiting time is 2.500000
Average TurnAround Time is 4.500000
```

Q12. Implementation of First In First Out (FIFO) Page Replacement Policy.

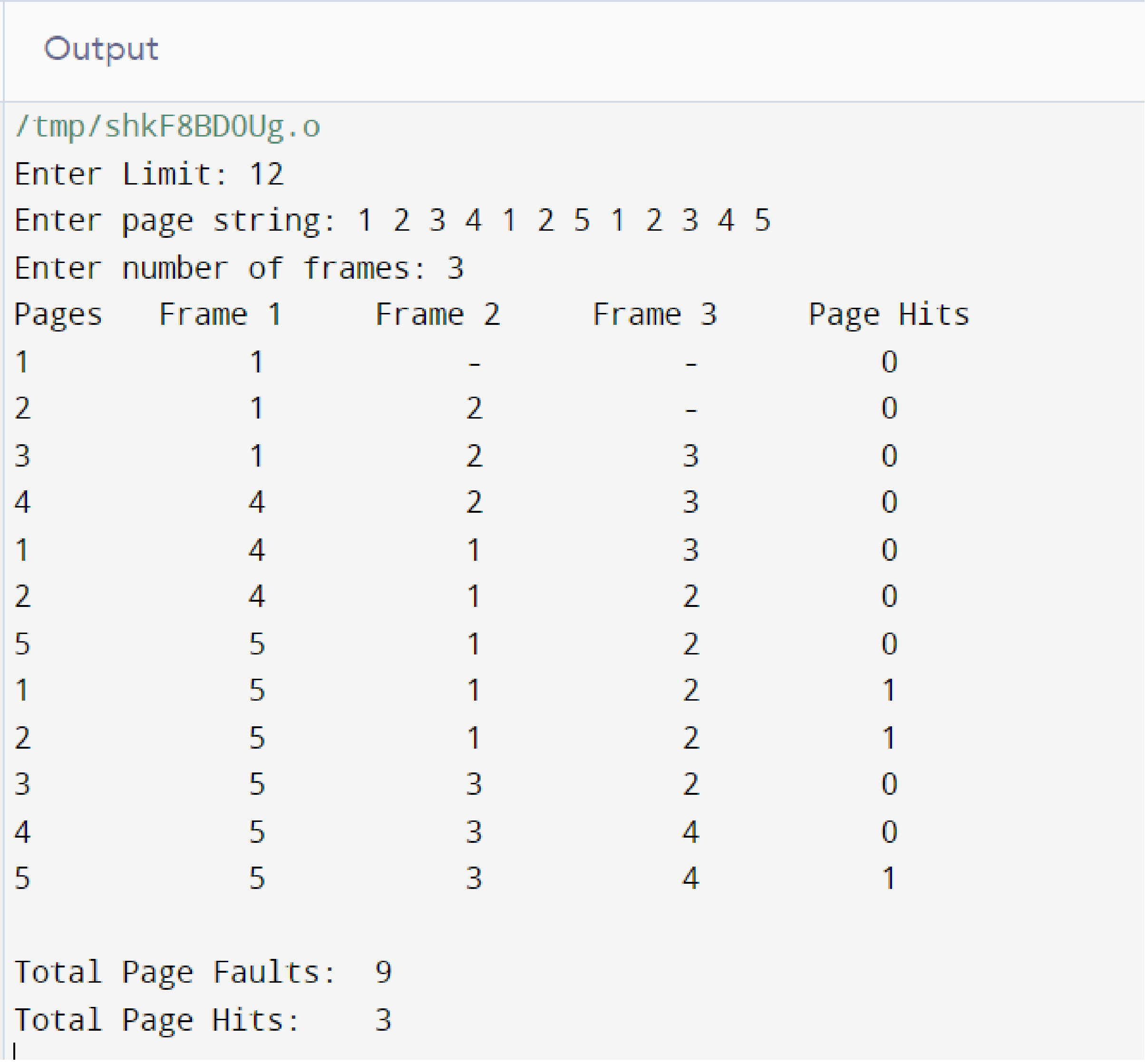
Code:

```
#include <stdio.h>
int main()
{
    int num;
    printf("Enter Limit: ");
    scanf("%d",&num);
    printf("Enter page string: ");
    int incomingStream[num];
    for(int i=0;i<num;i++)
    {
        scanf("%d",&incomingStream[i]);
    }
    int pageFaults = 0;
    int pageHits = 0;
    int frames;
    printf("Enter number of frames: ");
    scanf("%d",&frames);
    int m, n, s, pages;
    pages = sizeof(incomingStream) / sizeof(incomingStream[0]);
    printf("Pages\tFrame 1\t\tFrame 2\t\tFrame 3\t\tPage Hits\n");
    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++;
        pageHits++;
    }
}
if (s == 0)
{
    pageFaults++;
    if ((pageFaults <= frames))
    {
        temp[m] = incomingStream[m];
    }
    else
    {
        temp[(pageFaults - 1) % frames] = incomingStream[m];
    }
}
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("%d\n", s);
```

```
    }
    printf("\nTotal Page Faults:\t%d\n", pageFaults);
    printf("Total Page Hits:\t%d\n", pageHits);
    return 0;
}
```

*******OUTPUT*******



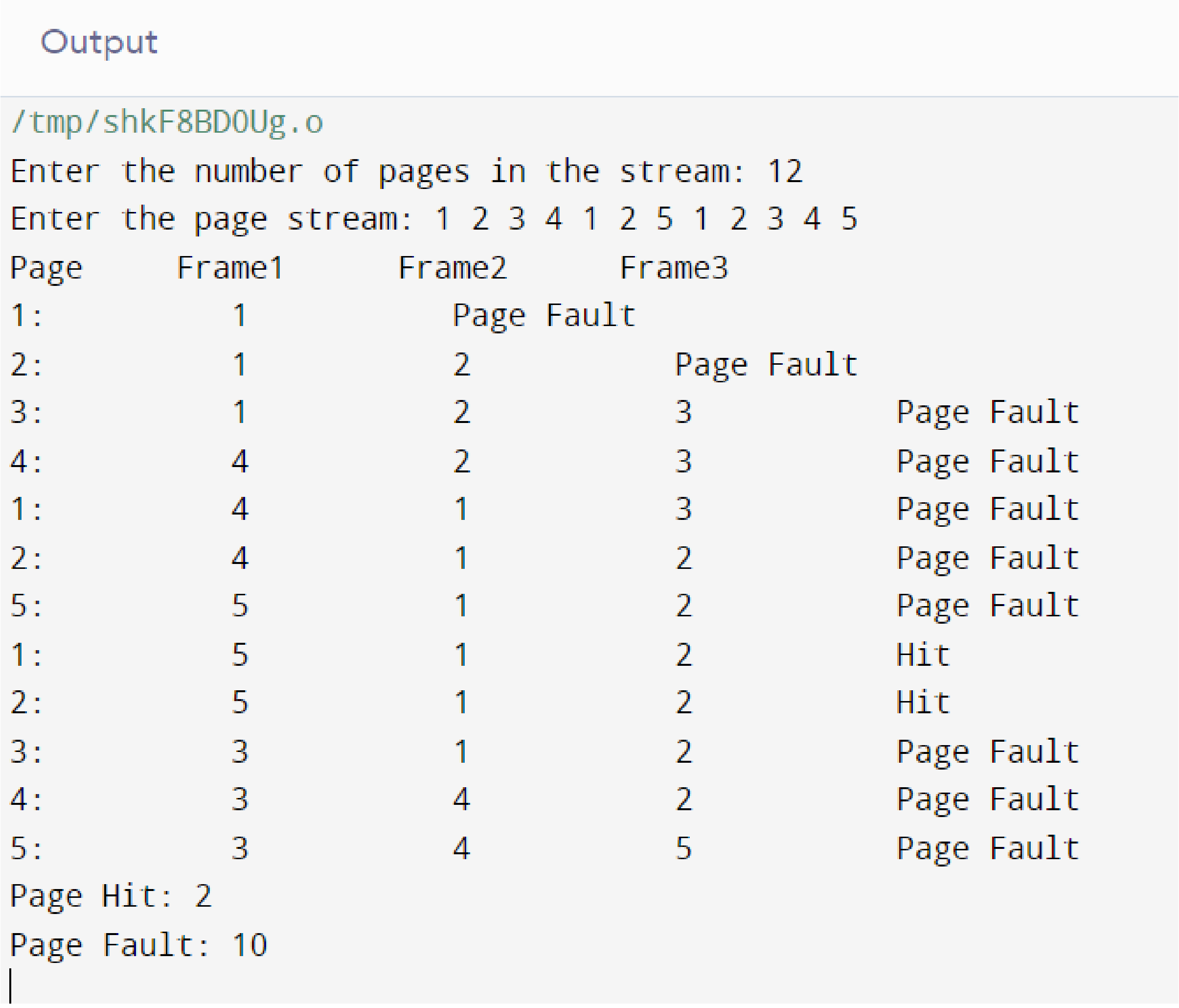
```
/tmp/shkF8BD0Ug.o
Enter Limit: 12
Enter page string: 1 2 3 4 1 2 5 1 2 3 4 5
Enter number of frames: 3
Pages   Frame 1     Frame 2     Frame 3     Page Hits
1            1           -           -          0
2            1           2           -          0
3            1           2           3          0
4            4           2           3          0
1            4           1           3          0
2            4           1           2          0
5            5           1           2          0
1            5           1           2          1
2            5           1           2          1
3            5           3           2          0
4            5           3           4          0
5            5           3           4          1

Total Page Faults:  9
Total Page Hits:    3
```

Q13. Implementation of LRU Page Replacement Policy.

Code:

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

int checkHit(int incomingPage, int queue[], int occupied) {
    for (int i = 0; i < occupied; i++) {
        if (incomingPage == queue[i])
            return 1;
    }
    return 0;
}

void printFrame(int queue[], int occupied) {
    for (int i = 0; i < occupied; i++)
        printf("%d\t\t\t", queue[i]);
}

int main() {
    int n;
    printf("Enter the number of pages in the stream: ");
    scanf("%d", &n);

    int incomingStream[n];
    printf("Enter the page stream: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &incomingStream[i]);

    int frames = 3;
int queue[n];
int distance[n];
int occupied = 0;
int pagefault = 0;
int pagehit = 0;

printf("Page\t Frame1 \t Frame2 \t Frame3\n");

for (int i = 0; i < n; i++) {
    printf("%d:  \t\t", incomingStream[i]);

    if (checkHit(incomingStream[i], queue, occupied)) {
        printFrame(queue, occupied);
        printf("Hit");
        pagehit++;
    } else if (occupied < frames) {
        queue[occupied] = incomingStream[i];
        pagefault++;
        occupied++;
        printFrame(queue, occupied);
        printf("Page Fault");
    } else {
        int max = INT_MIN;
        int index;
        for (int j = 0; j < frames; j++) {
            distance[j] = 0;
            for (int k = i - 1; k >= 0; k--) {
                ++distance[j];
                if (queue[j] == incomingStream[k])
                    break;
                }
                if (distance[j] > max) {
                    max = distance[j];
                    index = j;
                }
            }
            queue[index] = incomingStream[i];
            printFrame(queue, occupied);
            printf("Page Fault");
            pagefault++;
        }

        printf("\n");
    }

    printf("Page Hit: %d\n", pagehit);
    printf("Page Fault: %d\n", pagefault);

    return 0;
}
```

*******OUTPUT*******



```
/tmp/shkF8BD0Ug.o
Enter the number of pages in the stream: 12
Enter the page stream: 1 2 3 4 1 2 5 1 2 3 4 5
Page     Frame1      Frame2      Frame3
1:          1           Page Fault
2:          1           2           Page Fault
3:          1           2           3           Page Fault
4:          4           2           3           Page Fault
1:          4           1           3           Page Fault
2:          4           1           2           Page Fault
5:          5           1           2           Page Fault
1:          5           1           2           Hit
2:          5           1           2           Hit
3:          3           1           2           Page Fault
4:          3           4           2           Page Fault
5:          3           4           5           Page Fault
Page Hit: 2
Page Fault: 10
```