

31. Construct a C program to simulate the First in First Out paging technique of memory management.

Aim: To Construct a C program to simulate the First in First Out paging technique of memory management.

Algorithm:

1. Create an array to represent the page frames in memory.
2. Initialize all page frames to -1, indicating that they are empty.
3. Initialize a queue to keep track of the order in which pages are loaded into memory.
4. Initialize variables for page hits and page faults to zero.
5. Read the reference string (sequence of page numbers) from the user or use a predefined array.
6. For each page in the reference string, do the following:
 7. Check if the page is already in memory (a page hit).
 8. If it's a page hit, update the display and move to the next page.
 10. If it's a page fault (page not in memory)
9. Increment the page fault count.
10. Remove the oldest page in memory (the one at the front of the queue).
11. Load the new page into the memory and enqueue it.
12. Update the display to show the page replacement
13. Continue this process for all pages in the reference string.
14. After processing all pages, display the total number of page faults

Program:

```
#include<stdio.h>

#include<conio.h>

main()

{

int i, j, k, f, pf=0, count=0, rs[25], m[10], n;

printf("\n Enter the length of reference string -- ");

scanf("%d",&n);

printf("\n Enter the reference string -- ");

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

```

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

printf("\n Enter no. of frames -- ");

scanf("%d",&f);

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

m[i]=-1;


printf("\n The Page Replacement Process is -- \n");

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

{

for(k=0;k<f;k++)

{

if(m[k]==rs[i])

break;


}

if(k==f)

{

m[count++]=rs[i];

pf++;


}


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

printf("\t%d",m[j]);

if(k==f)

```

```

printf("\tPF No. %d",pf);

printf("\n");

if(count==f)

count=0;

}

printf("\n The number of Page Faults using FIFO are %d",pf);

getch();

}

```

Output:

```

Reference String: 7 0 1 2 0 3 0 4 2 3 0 3 2

Page Replacement Order:
Page 7 -> 7 - -
Page 0 -> 7 0 -
Page 1 -> 7 0 1
Page 2 -> 2 0 1
Page 3 -> 2 3 1
Page 0 -> 2 3 0
Page 4 -> 4 3 0
Page 2 -> 4 2 0
Page 3 -> 4 2 3
Page 0 -> 0 2 3

-----
Process exited after 0.0623 seconds with return value 0
Press any key to continue . . . |

```

32. Construct a C program to simulate the Least Recently Used paging technique of memory management.

Aim: To Construct a C program to simulate the Least Recently Used paging technique of memory management.

Algorithm:

1. Create an array to represent the page frames in memory.
2. Initialize all page frames to -1, indicating that they are empty.
3. Create a queue or a data structure (e.g., a doubly-linked list) to maintain the order of pages based on their usage history.
4. Initialize a counter for page hits and page faults to zero.
5. Read the reference string (sequence of page numbers) from the user or use a predefined array.
6. or each page in the reference string, do the following:
7. Check if the page is already in memory (a page hit).
8. If it's a page hit, update the position of the page in the usage history data structure to indicate it was recently used.
9. If it's a page fault (page not in memory), do the following:
10. Increment the page fault count.
11. Find the least recently used page in the usage history data structure (e.g., the front of the queue or the tail of the list).
12. Remove the least recently used page from memory and the usage history.
13. Load the new page into memory and add it to the back of the usage history.
14. Update the display to show the page replacement.
15. Continue this process for all pages in the reference string.
16. After processing all pages, display the total number of page faults

Program:

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

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

```
main()
```

```

{
int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;

printf("Enter the length of reference string -- ");

scanf("%d",&n);

printf("Enter the reference string -- ");

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

{

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

flag[i]=0;

}

printf("Enter the number of frames -- ");

scanf("%d",&f);

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

{

count[i]=0;

m[i]=-1;

}

printf("\n\nThe Page Replacement process is -- \n");

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

{

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

{

if(m[j]==rs[i])

{

```

```
flag[i]=1;
```

```
count[j]=next;
```

```
next++;
```

```
}
```

```
}
```

```
if(flag[i]==0)
```

```
{
```

```
if(i<f)
```

```
{ m[i]=rs[i];
```

```
count[i]=next;
```

```
next++;
```

```
}
```

```
else
```

```
{ min=0;
```

```
for(j=1;j<f;j++)
```

```
if(count[min] > count[j])
```

```
min=j;
```

```
m[min]=rs[i];
```

```
count[min]=next;
```

```
next++;
```

```

}

pf++;

}

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

printf("%d\t", m[j]);

if(flag[i]==0)

printf("PF No. -- %d" , pf);

printf("\n");

}

printf("\nThe number of page faults using LRU are %d",pf);

getch();

}

```

Output:

```

Enter the length of reference string -- 3
Enter the reference string -- 2
3
1
Enter the number of frames -- 3

```

The Page Replacement process is --

```

2   -1   -1   PF No. -- 1
2   3    -1   PF No. -- 2
2   3    1    PF No. -- 3

```

The number of page faults using LRU are 3

33. Construct a C program to simulate the optimal paging technique of memory management

Aim: To Construct a C program to simulate the optimal paging technique of memory management

Algorithm:

1. Create an array to represent the page frames in memory.
2. Initialize all page frames to -1, indicating that they are empty.
3. Initialize a variable for page faults to zero.
4. Read the reference string (sequence of page numbers) from the user or use a predefined array.
5. For each page in the reference string, do the following:
6. Check if the page is already in memory (a page hit).
7. If it's a page hit, move to the next page.
8. If it's a page fault (page not in memory), do the following:
9. Increment the page fault count.
10. Calculate the future references of each page in memory by scanning the remaining part of the reference string.
11. Find the page that will not be used for the longest time in the future (the optimal page to replace).
12. Replace the optimal page with the new page.
13. Continue this process for all pages in the reference string.
14. After processing all pages, display the total number of page faults.

Program:

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

```
int main()
```

```
{
```

```
int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;
```

```
printf("Enter number of frames: ");
```



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

printf("Enter number of pages: ");

scanf("%d", &no_of_pages);

printf("Enter page 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;

}

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

flag1 = flag2 = 0;


for(j = 0; j < no_of_frames; ++j){

if(frames[j] == pages[i]){

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j < no_of_frames; ++j){

if(frames[j] == -1){

faults++;
```

```
frames[j] = pages[i];

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

flag3 =0;

for(j = 0; j < no_of_frames; ++j){

temp[j] = -1;


for(k = i + 1; k < no_of_pages; ++k){

if(frames[j] == pages[k]){

temp[j] = k;

break;

}

}

}

for(j = 0; j < no_of_frames; ++j){

if(temp[j] == -1){

pos = j;

flag3 = 1;

break;
```

```
}  
  
}  
  
if(flag3 ==0){  
    max = temp[0];  
    pos = 0;  
  
    for(j = 1; j < no_of_frames; ++j){  
        if(temp[j] > max){  
            max = temp[j];  
            pos = j;  
        }  
    }  
  
    frames[pos] = pages[i];  
    faults++;  
}  
  
printf("\n");  
  
for(j = 0; j < no_of_frames; ++j){  
    printf("%d\t", frames[j]);  
}  
  
}  
  
printf("\n\nTotal Page Faults = %d", faults);  
  
return 0;
```

}

Output:

```
Reference String: 7 0 1 2 0 3 0 4 2 3 0 3 2

Page Replacement Order:
Page 7 -> 7 - -
Page 0 -> 0 - -
Page 1 -> 0 1 -
Page 2 -> 0 2 -
Page 3 -> 0 2 3
Page 4 -> 4 2 3
Page 0 -> 0 2 3

Total Page Faults: 7

-----
Process exited after 0.05286 seconds with return value 0
Press any key to continue . . . |
```

34. Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a C program to simulate the file allocation strategy.

Aim: To write a C program to simulate the file allocation strategy.

Algorithm:

1. Define the structure of a record that will be stored in the file.
2. Create a file to represent the sequential file.
3. Write records to the file sequentially, one after the other.
4. To read a specific record:
5. Prompt the user for the record number they want to access.
6. Read and display all records from the beginning of the file up to the requested record.
7. Continue this process until the user decides to exit.

Program:

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

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

```
#include<stdlib.h>
```

```

int main()

{
int f[50], i, st, len, j, c, k, count = 0;

for(i=0;i<50;i++)

f[i]=0;

x : count=0;

printf("Enter starting block and length of files: ");

scanf("%d%d", &st,&len);

for(k=st;k<(st+len);k++)

if(f[k]==0)

count++;

if(len==count)

{

for(j=st;j<(st+len);j++)

if(f[j]==0)

{

f[j]=1;

printf("%d\t%d\n",j,f[j]);

}

if(j!=(st+len-1))

printf("The file is allocated to disk\n");

}

else

```

```

printf("The file is not allocated \n");

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

goto x;

else

exit(0);

getch();

}

```

Output:

```

Enter records (Enter '0' as record number to exit):
Record Number: 389
Data: JASWANTH
Record Number: 0
Enter the record number to read (0 to exit): 389
Record Number: 389
Data: JASWANTH
Enter the record number to read (0 to exit): 0

-----
Process exited after 29.88 seconds with return value 0
Press any key to continue . . . |

```

35.Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a C program to simulate the file allocation strategy.

Aim: To write a C program to simulate the file allocation strategy.

ALGORITHM:

1. Define the structure of a block that will be stored in the file.

2. Create a file to represent the indexed file.
3. Initialize an index block that contains pointers to data blocks.
4. To write a new block:
5. Prompt the user for the block number and the data to be written to the block.
6. Update the corresponding entry in the index block to point to the new data block.
7. Write the data block to the file.
8. To read a specific block:
9. Prompt the user for the block number they want to access.
10. Use the index block to find the pointer to the requested data block.
11. Read and display the data in the requested data block.
12. Continue this process until the user decides to exit.

Program:

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int main()

{

int f[50], index[50], i, n, st, len, j, c, k, ind, count=0;

for(i=0; i<50; i++)

f[i]=0;

x:printf("Enter the index block: ");

scanf("%d", &ind);

if(f[ind]!=1)

{

printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);
```



```

scanf("%d",&n);

}

else

{

printf("%d index is already allocated \n",ind);

goto x;

}

y: count=0;

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

{

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

if(f[index[i]]==0)

count++;

}

if(count==n)

{

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

f[index[j]]=1;

printf("Allocated\n");

printf("File Indexed\n");

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

printf("%d----->%d : %d\n",ind,index[k],f[index[k]]);

}

```

```
else  
  
{  
  
printf("File in the index is already allocated \n");  
  
printf("Enter another file indexed");  
  
goto y;  
  
}  
  
printf("Do you want to enter more file(Yes - 1/No - 0)");  
  
scanf("%d", &c);  
  
if(c==1)  
  
goto x;  
  
else  
  
exit(0);  
  
getch();  
  
}
```

Output:

```

Enter blocks (Enter '0' as block number to exit):
Block Number: 39
Data: JSAWNTH
Block Number: 43
Data: SAI
Block Number: 12
Data: FRIEND
Block Number: 0
Enter the block number to read (0 to exit): 12
Block Number: 12
Data: FRIEND
Enter the block number to read (0 to exit): 0

-----
Process exited after 34.15 seconds with return value 0
Press any key to continue . . . |

```

36. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a C program to simulate the file allocation strategy.

Aim: To design a C program to simulate the file allocation strategy.

ALGORITHM:

1. Define the structure of a block that will be stored in the file.
Each block contains a pointer to the next block.
2. Create a file to represent the linked allocation.
3. Create a directory entry for the file containing a pointer to the first and last blocks.
4. To write a new block:
5. Prompt the user for the block data.
6. Allocate a new block in the file.
7. If it's the first block, update the directory entry to point to it as both the first and last block.
8. If it's not the first block, update the previous block to point to the new block.
9. Update the new block's pointer to the next block (usually NULL for the last block).

- 10.To read a specific block:
- 11.Prompt the user for the block number they want to access.
- 12.Use the directory entry to find the first block of the file.
- 13.Traverse the linked list of blocks until you reach the desired block.
- 14.Read and display the data in the requested block.
- 15.Continue this process until the user decides to exit.

Program:

```
#include<conio.h>

#include<stdlib.h>

int main()

{

int f[50], p,i, st, len, j, c, k, a;

for(i=0;i<50;i++)

f[i]=0;

printf("Enter how many blocks already allocated: ");

scanf("%d",&p);

printf("Enter blocks already allocated: ");

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

{

scanf("%d",&a);

f[a]=1;

}

x: printf("Enter index starting block and length: ");
```

```
scanf("%d%d", &st,&len);

k=len;

if(f[st]==0)

{

for(j=st;j<(st+k);j++)

{

if(f[j]==0)

{

f[j]=1;

printf("%d----->%d\n",j,f[j]);

}

else

{

printf("%d Block is already allocated \n",j);

k++;

}

}

}

else

printf("%d starting block is already allocated \n",st);

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)
```

```
goto x;

else

exit(0);

}
```

Output:

```
Linked Allocation Simulation
Enter 'W' to write a block, 'R' to read a block, or 'Q' to quit: W
Enter data for the block: SAI IS WORST
Enter 'W' to write a block, 'R' to read a block, or 'Q' to quit: W
Enter data for the block: JASWANTH IS GOOD
Enter 'W' to write a block, 'R' to read a block, or 'Q' to quit: R
Enter the block number to read (1-2): 2
Block 2 Data: JASWANTH IS GOOD
Enter 'W' to write a block, 'R' to read a block, or 'Q' to quit: R
Enter the block number to read (1-2): 1
Block 1 Data: SAI IS WORST
Enter 'W' to write a block, 'R' to read a block, or 'Q' to quit: Q

-----
Process exited after 46.7 seconds with return value 0
Press any key to continue . . . |
```

37. Construct a C program to simulate the First Come First Served disk scheduling algorithm.

Aim: To Construct a C program to simulate the First Come First Served disk scheduling algorithm.

ALGORITHM:-

1. Start at the current position of the disk head.

2. For each disk request in the queue:
 - Move the disk head to the requested track.
 - Calculate the seek time as the absolute difference between the new position of the disk head and the previous position.
 - Add the seek time to the total seek time.
 - Update the previous position of the disk head to the current position.
3. Repeat step 2 for all disk requests in the queue.
4. After serving all the requests, calculate and display the total seek time.
5. Calculate and display the average seek time, which is the total seek time divided by the number of requests.

Program:

```
#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,n,TotalHeadMoment=0,initial;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

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

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

printf("Enter initial head position\n");

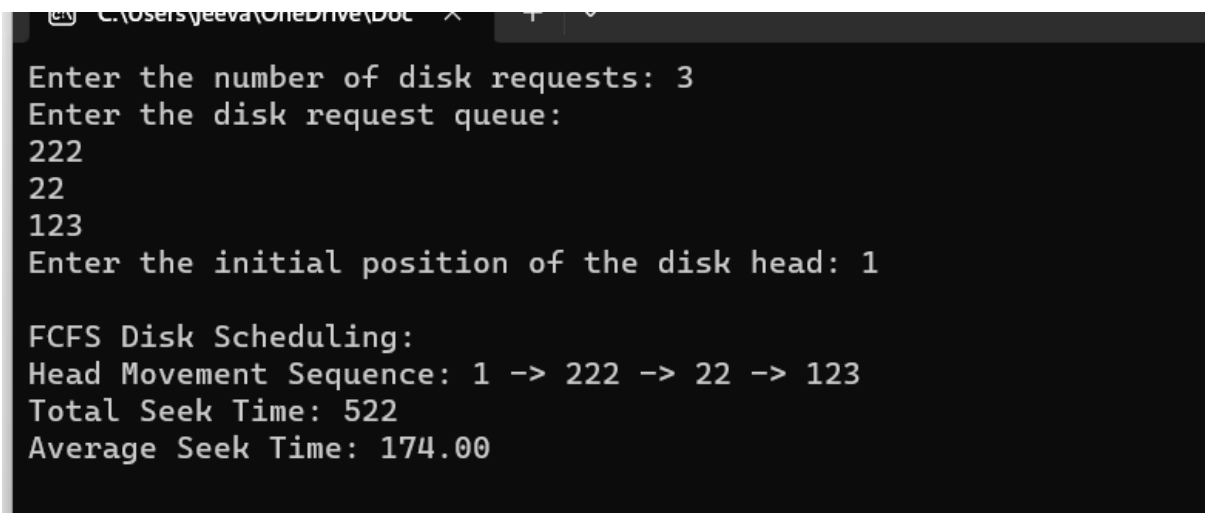
scanf("%d",&initial);
```

```

for(i=0;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
printf("Total head moment is %d",TotalHeadMoment);
return 0;
}

```

Output:



The screenshot shows a terminal window with the following text:

```

Enter the number of disk requests: 3
Enter the disk request queue:
222
22
123
Enter the initial position of the disk head: 1

FCFS Disk Scheduling:
Head Movement Sequence: 1 -> 222 -> 22 -> 123
Total Seek Time: 522
Average Seek Time: 174.00

```

38.Design a C program to simulate SCAN disk scheduling algorithm

Aim: To design a C program to simulate SCAN disk scheduling algorithm

Algorithm:

1. Determine the direction of movement (inward or outward) based on the queue of pending requests and the current position.
2. While servicing requests in the selected direction:
 - Move the disk head to the next track in the current direction.
 - Calculate the seek time as the absolute difference between the new position of the disk head and the previous position.
 - Add the seek time to the total seek time.
 - Update the previous position of the disk head to the current position.
3. If there are no more requests in the current direction, change direction to the opposite direction.
4. Repeat step 3 until all requests are serviced.
5. After serving all the requests, calculate and display the total seek time.
6. Calculate and display the average seek time, which is the total seek time divided by the number of requests.

Program:

```
#include <stdio.h>

#include <stdlib.h>

int main() {

int n, head, direction, total_head_movement = 0;

printf("Enter the number of disk requests: ");

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

```
int requests[n];

printf("Enter the disk requests:\n");

for (int i = 0; i < n; i++) {
    scanf("%d", &requests[i]);
}

printf("Enter the initial head position: ");
scanf("%d", &head);

printf("Enter the head movement direction (1 for right, 0 for left): ");
scanf("%d", &direction);

// Sort the requests to make it easier to simulate SCAN

for (int i = 0; i < n - 1; i++) {
    for (int j = i + 1; j < n; j++) {
        if ((direction == 1 && requests[i] > requests[j]) || (direction == 0 &&
requests[i] < requests[j])) {
            int temp = requests[i];
            requests[i] = requests[j];
            requests[j] = temp;
        }
    }
}
```

```
int index = 0;
```

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

```
    if ((direction == 1 && requests[i] > head) || (direction == 0 && requests[i] < head)) {
```

```
        index = i;
```

```
        break;
```

```
    }
```

```
}
```

```
if (direction == 1) {
```

```
    for (int i = index; i < n; i++) {
```

```
        total_head_movement += abs(requests[i] - head);
```

```
        head = requests[i];
```

```
    }
```

```
    total_head_movement += abs(requests[n - 1] - (direction == 1 ? 0 : 0));
```

```
    } else {
```

```
        for (int i = index - 1; i >= 0; i--) {
```

```
            total_head_movement += abs(requests[i] - head);
```

```
            head = requests[i];
```

```
        }
```

```
        total_head_movement += abs(requests[0] - (direction == 1 ? 0 : 0));
```

```
    }
```

```
printf("Total head movement is: %d\n", total_head_movement);
```

```
return 0;
```

```
}
```

Output:

```
t Enter the number of disk requests: 3
Enter the disk request queue:
12
34
45
Enter the initial position of the disk head: 45

SCAN (Elevator) Disk Scheduling:
Total Seek Time: 0
Average Seek Time: 0.00
```

39. Develop a C program to simulate C-SCAN disk scheduling algorithm

Aim: To Develop a C program to simulate C-SCAN disk scheduling algorithm

ALGORITHM:-

1. Start at the current position of the disk head.
2. Set the direction of movement to one side (e.g., right).
3. While servicing requests in the selected direction:
 - Move the disk head to the next track in the current direction.
 - Calculate the seek time as the absolute difference between the new position of the disk head and the previous position.
 - Add the seek time to the total seek time.
 - Update the previous position of the disk head to the current position.
4. If there are no more requests in the current direction:

- Move the disk head to the end of the disk in the current direction.
 - Change direction to the opposite side (e.g., left).
 - Continue servicing requests in the new direction.
5. Repeat step 3 and step 4 until all requests are serviced.
 6. After serving all the requests, calculate and display the total seektime.
 7. Calculate and display the average seek time, which is the total seektime divided by the number of requests.

Program:

```
#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

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

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

printf("Enter initial head position\n");

scanf("%d",&initial);

printf("Enter total disk size\n");

scanf("%d",&size);

printf("Enter the head movement direction for high 1 and for low 0\n");

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

```
for(i=0;i<n;i++)  
{  
for( j=0;j<n-i-1;j++)  
{  
if(RQ[j]>RQ[j+1])  
{  
int temp;  
temp=RQ[j];  
RQ[j]=RQ[j+1];  
RQ[j+1]=temp;  
}  
  
}  
}
```

```
int index;  
for(i=0;i<n;i++)  
{  
if(initial<RQ[i])  
{  
index=i;  
break;  
}
```

```

}
if(move==1)
{
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
initial=0;
for( i=0;i<index;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];

}
}
else
{
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

```

```

initial=RQ[i];

}

TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);

TotalHeadMoment=TotalHeadMoment+abs(size-1-0);

initial =size-1;

for(i=n-1;i>=index;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];


}

}

printf("Total head movement is %d",TotalHeadMoment);

return 0;

}

```

Output:

```

Enter the number of disk requests: 3
Enter the disk request queue:
12
13
14
Enter the initial position of the disk head: 5

C-SCAN Disk Scheduling:
Total Seek Time: 37
Average Seek Time: 12.33

```


40. Illustrate the various File Access Permission and different types users in Linux.

Aim: To write a c program about various File Access Permission and different types users in Linux.

ALGORITHM:

1. Create a file or identify an existing file to demonstrate permissions and users.
2. View the file's permissions using the `ls -l` command.
The output will look something like this:
.txt
 - The first character (-) represents the file type (a dash indicates a regular file).
 - The next three characters (rw-) represent the permissions for the file's owner (Read and Write, no Execute).
 - The next three characters (r--) represent the permissions for the file's group (Read, no Write or Execute).
 - The last three characters (r--) represent the permissions for others (Read, no Write or Execute).
 - The number 1 represents the number of hard links to the file.
 - owner is the username of the file's owner.
 - group is the name of the file's group.
 - 1234 is the file's size in bytes.
 - Oct 19 10:30 is the last modification timestamp.
 - file.txt is the file name.
3. Use the `chmod` command to change the file's permissions. For example, to give the group write permission, use `chmod g+w file.txt`.
4. Re-run `ls -l` to confirm the updated permissions.
5. You can also change the file's owner and group using the `chown` and `chgrp` commands, respectively.

To create and manage user accounts, you can use the `useradd` and `passwd` commands.

Program:

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

```
#include <stdlib.h>
```

```
#include <string.h>

#include <unistd.h>

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

int result;

char *filename = (char *)malloc(512);

if (argc < 2) {

strcpy(filename, "/usr/bin/adb");

} else {

strcpy(filename, argv[1]);

}

result = access (filename, R_OK);

if ( result == 0 ) {

printf("%s is readable\n",filename);

} else {

printf("%s is not readable\n",filename);

}

result = access (filename, W_OK);

if ( result == 0 ) {

printf("%s is Writeable\n",filename);

} else {

printf("%s is not Writeable\n",filename);

}


result = access (filename, X_OK);
```

```
if ( result == 0 ) {  
    printf("%s is executable\n",filename);  
} else {  
    printf("%s is not executable\n",filename);  
}  
free(filename);  
return 0;  
}
```

Output:

1. Compile the C program (assuming it's saved in a file named `change_permissions.c`):


bash

 Copy code

```
gcc -o change_permissions change_permissions.c
```

1. Run the program:

bash


 Copy code

```
./change_permissions
```

Output:

If the program executes successfully, it should display the following output:

arduino

 Copy code

```
File permissions changed successfully.
```