Experiment-6

Disk scheduling

Deepesh Tank
18075017

DATE / /
PAGE

Aim:-

write a C program to simulate the following disk scheduling algorithms.

a) FCFS
b) SSTF
c) SCAN
d) CSCAN
e) Look.

Theory :-

a) FCFS →

it is the simplest of all disk scheduling algorithm. request are addressed in the order of their arrival in the disk queue.

b) SSTF :- (shortest seek time first)

request having shortest seek time are executed first so seek time of each process calculated in advance in the queue and then they are scheduled in accordance to their calculated seek time.

c) SCAN :-

in this algorithm the disk arm moves into a particular direction and service the request coming in its path and after reaching end of disk. it reverse its direction and again services the request arriving in its path so this algorithm work as an elevator and hence also known as elevator algorithm.

As a result the request at mid range are serviced more and those arriving behind the disk arm will have to wait

d) CSCAN:-

in CSCAN algorithm disk arm instead of reversing its direction goes to other end of the disk and start servicing the request from there so the disk arm move in circular fashion and algorithm is also similar to SCAN algorithm and therefore it is named CSCAN algorithm

e) Look :-

it is similar to SCAN disk except for the difference that the disk arm in spite of going to end of the disk goes only to the last request to be served in front of the head and then reverses its direction from there only. Thus it prevent the extra delay which occured due to unnecessary traversal at the end of the disk.

Algorithm :-

a) FCFS :-

1. Let Request array represent an array storing index of tracks that have been requested in ascending order of their time of arrival head is the position of disk head
2. Let us one by one take the tracks in default order and calculate absolute distance of track from head.

3. increment total seek count with this distance.
4. currently serviced track positions now becomes the new head positions.
5. Go to step 2 until all tracks in request array have not been serviced.

b) SSTF

1) Let Request array represent an array storing indexes of tracks that have been requested. head is the position of disk head.
2) Find the positive distance of all tracks in the request array from head
3) find a track from requested array which has not been accessed yet and minimum distance from head.
4) increment total seek count with this distance.
5) currently serviced track position now becomes the new head position.
6) Go to step 2 until all tracks in request array have not been serviced.

c) SCAN.

1) Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival head is position of disk head.
2) Let direction represents whether head is moving toward left or right.
3) In the direction, in which head is serviced all tracks meby one.

4) calculate absolute distance of the track from head
5) increment the total seek count with distance
6) Currently serviced track position now become head position.
7) go to step 3 until we reach at one of the end of disk
8) if we reach at the end of the disk reverse the direction and go to step 2 until all tracks in request array have not been serviced.

d) CSCAN :-

1) step 1 same as that of scan algorithm
2) The head services only in the right direction from 0 to size of disk
3) while moving in the left direction do not service any of the track.
4) when we reach left end reverse the direction.
5) while moving in right direction it services all tracks one by one.
6) → step 4 of scan algorithm
7) step 5 of scan algorithm
8) step 6 of scan algorithm
9) step 7 of scan algorithm.
10) if we reach at the right end of the disk reverse the direction and go to step 3 until all tracks in request array have not been serviced.

e) Look

1) Let request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival head is the position of disk head.
2) intial direction in which head is moving is given and it services in the same direction.
3) The head services all the request one by one in the direction head is moving.
4) The head continues to move in the same direction until all the request in this direction are not finished.
5) while moving in this direction calculate absolute distance of the track from head.
6) increment total seek count with distance.
7) Currently serviced track position now becomes the new head position.
8) Go to step 5 until we reach at last request in this direction.
9) if we reach where no requests are needed to be serviced in this direction reverse the direction and go to step 3 until all tracks in request array have not been serviced.

program code:-

a) FCFS:-

```
# include<stdio.h>
int size=8;
void FCFS (int arr[], int head) {
  int seek_count=0;
int distance, curr_track;
for (int i=0; i<size; i++){
  curr_track = arr[i];
distance = abs[curr_track-head);
seek_count+=distance;
head = curr_track; }
prmtf ("Total number of seek operations %d\n", seek_count);
prmtf("seek sequence is \n");.
for (int i=0; i<size; i++) { prmtf ("%d \n", arr[i]);
   }

int main() {
     int arr [size] = { 176, 79, 34, 60, 92, 11, 41, 114}
     int head = 50;
     FCFS (arr, head);
     return 0; }
```

output :-

Total number of seek operations = 510

Seek sequence is

176

79

34

60

92

11

41

114

b) SSTF:-

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(){  int n=8;
int queue[8]={98,183,37,122,14,124,65,67};
int head=53, seek=0, i, temp;
float average;
for(i=1;i<n;i++) t[i]=abs(head-queue[i]);
for(i=0;i<n;i++){
for(j=i+1;j<n;j++){
if(t[i]>t[j]){ temp=t[i];
               t[i]=t[j];
               t[j]=temp;
               temp=queue[i]
               queue[i]=queue[j]
               queue[j]=temp;}}}

for(i=1;i<n-1;i++)
{  seek=seek+abs(head-queue[i]);
   head=queue[i];}
printf("\n Total seek time is %d \t",seek);
avg=seek/float(n);
printf("\n average seek time is %f\t",avg);
getch();
}
```

output

Total seek time is 236

Average seek time is 29.500000

c) SCAN:-

```
# include <stdio.h>
int size=8; int disk_size =200;
void scan(int arr[], int head, string direction) {
int seek_count=0;
int distance, curr_track;
vector <int> left, right
vector<int>seek_sequence;
if (direction == "left") left.push_back(0);
else if (direction == "right") right.push_back(disk_size-1);
for (int i=0; i<size; i++) {
    if (arr[i] <head) left.push_back(arr[i]);
if (arr[i] >head) right.push_back(arr[i]); }
int run=2;
while(run--){
if (direction == 'left'){
    for (int i = left.size() -1; i>=0; i--)
        curr_track = left[i];
seek_sequence.push_back(curr_track);
distance =abs (current_track-head);
seek_count+=distance;
head = curr_track; }
direction="right"; }
else if (direction=='right') {
    for (int i=0; i< right.size(); i++) {
        curr_track =right[i];
seek_sequence.push_back(curr_track);
distance =abs(curr_track - head);
seek_count+= distance;
head= curr_track; }
  direction= 'left'; } }
```

```
printf("Total number of seek operation = %d \n", seek_count);
printf(" seek sequence \n");
for(int i=0; i< seek_sequence.size(); i++)
printf(" %d \n", seek_sequence[i]); }
int main()
{  int arr[size] = {176, 79, 34, 60, 92, 11, 41, 114};
    int head = 50;
    string direction = "left";
    scan(arr, head, direction);
    return 0;
}
```

output:-

Total number of seek operations = 226

seek sequence is

41

34

11

0

60

79

92

114

176

d) CSCAN

```
#include<stdio.h>
int size=8;
int disk_size=200;
void CSCAN (int arr[], int head){
 int seek_count=0;
 int distance, cur_track;
 vector<int> left, right;
 vector<int> seek_sequence;
 left.push_back(0);
 right.push_back (disk_size-1);
 for (int i=0; i<size; i++){
   if (arr[i]<head) left.push_back (arr[i]);
   if (arr[i]>head) right.push_back (arr[i]);
 std::sort (left.begin(), left.end());
 std::sort (right.begin(), right.end());
 for (int i=0; i<right.size(); i++) {
   cur_track=right[i];
  seek_sequence.push_back (cur_track);
   distance = abs (cur_track - head);
   seek_count += distance;
   head = cur_track; }
 printf("Total number of seek operation %d \n", seek_count);
 printf ("seek sequence is \n");
 for (int i=0; i<seek_sequence.size(); i++){
     printf ("%d \n", seek_sequence[i]); }
}
```

```
int main(){
int arr[size] = {176,79,34,60,92,11,41,114};
int head = 50;
prntf("inhial posihon of head %d \n", head);
CSCAN(arr, head);
retorn 0;
y
```

Output:-

inhial posihon of head = 50
Total numbee of seek aperahon = 190
seek sequence is
50
72
92
114
176
199
0
11
34
41

e) Look :-

```
#include<stdio.h>
int size=8;
int disk_size=200;
void look(int arr[], int head, string direction){
 int seek_count=0;
 int distance, curr_track;
vector<int>left, right;
vector<int> seek_sequence;
for(int i=0; i<size; i++){
 if (arr[i]<head) left.push_back(arr[i]);
if (arr[i]>head)right.push_back (arr[i]); }
std::sort (left.begin(), left.end());
std::sort (right.begin(), right.end());
int run=2
while (run--){
if (direction == "left"){
 for (int i=left.size()-1; i>=0; i--){
        curr_track=left[i];
seek_sequence.push_back(curr_track);
distance =abs(curr_track-head);
seek_count += distance;
head = curr_track; }
  direction="right"; }
else if (direct=="right"){
for(int i=0; i<right.size(); i++){
    curr_track = right[i];
 seek_sequence.push_back(curr_track);
distance = abs(curr_track - head);
seek_count += distance;
head = curr_track; direction="left"; } }
```

```
printf("Total number of seek operations = %d \n", seek_count);
prmtf("seek sequence is \n");
for (int i=0; i< seek_sequence.size(); i++){
    cout printf("%d \n", seek_sequence[i];}
}

int main()
{ int arr[size] ={ 176, 79, 34, 60, 92, 11, 41, 114};
int head = 50;
string direct = "right";
printf("Intially position of head %d \n", head);
look(arr, head, direct);
return 0;
}
```

Output :-

intial position of head = 50
Total number of seek operations = 291
seek sequence is
60
79
92
114
176
41
34
11