Lab 10: Disk Scheduling Algorithms

1. Objective

- Understand Disk Scheduling Algorithm and read its code for SSTF, SCAN, CSCAN in C, try to write other kinds of disk scheduling algorithms such as LOOK and SAFT.

2. Syllabus

- Understand Disk Scheduling algorithms;
- Implement these algorithms with C.

3. Prerequisite

- C language
- Computer which run Linux system (e.g. Ubuntu)

4. Concepts and Principles of this lab

Please refer to the chapter 9. The main contents have been lectured in the eighth week, 12/6/2017 and 12/11/2017. The slide of this chapter can be found in the course website: http://www.thinkmesh.net/~jiangxl/teaching/106F14A/.

5. Experimental Contents

5.1 Program for Shortest Seek Time First

Description:

In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

Note:

An Example of input:
Size of queue: 8
The queue: 98 183 37 122 14 124 65 67
Initial head position: 53

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    int queue[100],t[100],head,seek=0,n,i,j,temp;
    float avg;
    // clrscr();
    printf("*** SSTF Disk Scheduling Algorithm ***\n");
    printf("Enter the size of Queue\t");
```

```
    scanf("%d",&n);
    printf("Enter the Queue\t");
    for(i=0;i<n;i++)
    {
        scanf("%d",&queue[i]);
    }
    printf("Enter the initial head position\t");
    scanf("%d",&head);
    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("\nTotal Seek Time is%d\t",seek);
    avg=seek/(float)n;
    printf("\nAverage Seek Time is %f\t",avg);
    getch();
}
```

File name: SSTF.c

How to run: gcc -o SSTF SSTF.c
./SSTF

5.2 Program for SCAN algorithm

Description: In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works like an elevator and hence also

known as elevator algorithm**.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Note: An Example of input:

Total number of location: 8

Position of disk head: 53

Elements of disk head queue: 98 183 37 122 14 124 65 67

```
#include<stdio.h>
#include<conio.h>
void scan_algorithm(int left[], int right[], int count, int limit)
{
	int arr[20];
	int x = count - 1, y = count + 1, c = 0, d = 0, j;
	while(x > -1)
	{
		printf("\nX:\t%d", x);
		printf("\nLeft[X]:\t%d", left[x]);
		arr[d] = left[x];
		x--;
		d++;
	}
	arr[d] = 0;
	while(y < limit + 1)
	{
		arr[y] = right[c];
		c++;
		y++;
	}
	printf("\nScanning Order:\n");
	for(j = 0; j < limit + 1; j++)
	{
		printf("\n%d", arr[j]);
	}
}
void division(int elements[], int limit, int disk_head)
{
	int count = 0, p, q, m, x;
	int left[20], right[20];
	for(count = 0; count < limit; count++)
	{
		if(elements[count] > disk_head)
		{
			printf("\nBreak Position:\t%d\n", elements[count]);
			break;
```

```
                }
        }
        printf("\nValue:\t%d\n", count);
        q = 1;
        p = 0;
        m = limit;
        left[0] = elements[0];
        printf("\nLeft:\t%d", left[0]);
        while(q < count)
        {
                printf("\nElement[l] value:\t%d" , elements[q]);
                left[q] = elements[q];
                printf("\nLeft:\t%d", left[q]);
                q++;
                printf("\nl:\t%d", q);
        }
        x = count;
        while(x < m)
        {
                right[p] = elements[x];
                printf("\nRight:\t%d", right[p]);
                printf("\nElement:\t%d", elements[x]);
                p++;
                x++;
        }
        scan_algorithm(left, right, count, limit);
}
void sorting(int elements[], int limit)
{
        int location, count, j, temp, small;
        for(count = 0; count < limit - 1; count++)
        {
                small = elements[count];
                location = count;
                for(j = count + 1; j < limit; j++)
                {
                        if(small > elements[j])
                        {
                                small = elements[j];
                                location = j;
                        }
                }
                temp = elements[location];
                elements[location] = elements[count];
```

```
                elements[count] = temp;
        }
}

int main()
{
        int count, disk_head, elements[20], limit;
        printf("Enter total number of locations:\t");
        scanf("%d", &limit);
        printf("\nEnter position of disk head:\t");
        scanf("%d", &disk_head);
        printf("\nEnter elements of disk head queue\n");
        for(count = 0; count < limit; count++)
        {
                printf("Element[%d]:\t", count + 1);
                scanf("%d", &elements[count]);
        }
        sorting(elements, limit);
        division(elements, limit, disk_head);
        getch();
        return 0;
}
```

File name: SCAN.c

How to run: gcc -o SCAN SCAN.c
./SCAN

5.3 Program for First In First Out (FIFO) algorithm

Description: The CSCAN algorithm is a disk scheduling algorithm that helps in determining the motion of a In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area. These situations are avoided in CSAN algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Note: An Example of input:
Maximum Range of Disk: 5
Initial Head Position: 1
Que Request Size: 5
Disk Queue Element Positions: 1 2 3 4 5

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
```

```
int queue1[30], queue2[30], queue3[30];
int limit, disk_head, count = 0, j, seek_time = 0, range, diff;
int t1, t2 = 0, t3 = 0;
float avg_seek_time;
printf("Maximum Range of Disk:\t");
scanf("%d", &range);
printf("Initial Head Position:\t");
scanf("%d", &disk_head);
printf("Queue Request Size:\t");
scanf("%d", &limit);
printf("Disk Queue Element Positions:\n");
while(count < limit)
{
        scanf("%d", &t1);
        if(t1 >= disk_head)
        {
                queue1[t2] = t1;
                t2++;
        }
        else
        {
                queue2[t3] = t1;
                t3++;
        }
        count++;
}
count = 0;
while(count < t2 - 1)
{
        j = count + 1;
        while(j < t2)
        {
                if(queue1[count] > queue1[j])
                {
                        t1 = queue1[count];
                        queue1[count] = queue1[j];
                        queue1[j] = t1;
                }
                j++;
        }
        count++;
}
count = 0;
while(count < t3 - 1)
```

```
        {
                j = count + 1;
                while(j < t3)
                {
                        if(queue2[count] > queue2[j])
                        {
                                t1 = queue2[count];
                                queue2[count] = queue2[j];
                                queue2[j] = t1;
                        }
                        j++;
                }
                count++;
        }
        count = 1;
        j = 0;
        while(j < t2)
        {
                queue3[count] = queue1[j];
                queue3[count] = range;
                queue3[count + 1] = 0;
                count++;
                j++;
        }
        count = t2 + 3;
        j = 0;
        while(j < t3)
        {
                queue3[count] = queue2[j];
                queue3[0] = disk_head;
                count++;
                j++;
        }
        for(j = 0; j <= limit + 1; j++)
        {
                diff = abs(queue3[j + 1] - queue3[j]);
                seek_time = seek_time + diff;
                printf("\nDisk Head:\t%d -> %d [Seek Time: %d]\n", queue3[j], queue3[j + 1], diff);
        }
        printf("\nTotal Seek Time:\t%d\n", seek_time);
        avg_seek_time = seek_time / (float)limit;
        printf("\nAverage Seek Time:\t%f\n", avg_seek_time);
        return 0;
}
```

File name: CSCAN.c

How to run: gcc -o CSCAN CSCAN.c
./CSCAN

6. Conclusion:

In this chapter, four experiments have been completed. Now let's try to search and read some other algorithms, such as LOOK, SAFT.