Study Set for Lecture 13: Mass Storage Structure

Due Dec 7 at 11:59pm **Points** 10 **Questions** 7
Available Nov 24 at 12pm - Dec 8 at 8pm 14 days **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect13_Mass_Storage_Structure.pdf**.

Then answer the questions from this study set and submit them to gain access to the further part of the course.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	10,229 minutes	10 out of 10 *

* Some questions not yet graded

Correct answers are hidden.

Score for this attempt: **10** out of 10 *
Submitted Dec 6 at 11:18am
This attempt took 10,229 minutes.

Question 1

Not yet graded / 0 pts

Explain what NAS and SAN are.
Describe the difference between the two.

Your Answer:

Both network-attached storage (NAS) and storage area network (SAN) were developed to solve the problem of making stored data available to a

lot of users at once. Each of them provides dedicated storage for a group of users, but they couldn't be more different in their approach to achieving their mission.

A NAS is a single storage device that serves files over Ethernet and is relatively inexpensive and easy to set up, while a SAN is a tightly coupled network of multiple devices that work with block-based data and is more expensive and complex to set up and manage. From a user perspective, the biggest difference between NAS and SAN is that NAS devices look like volumes on a file server and use protocols like NFS and SMB/CIFS, while SAN-connected disks appear to the user as local drives.

Benefits of NAS

A NAS is frequently the next step up for a home office or small business that is using DAS (direct attached storage). The move up to NAS results from the desire to share files locally and remotely, having files available 24/7, data redundancy, the ability to replace and upgrade hard drives in the system, and and the availability of other services such as automatic backup.

Summary of NAS Benefits

- Relatively inexpensive
- 24/7 and remote data availability
- Good expandability
- Redundant storage architecture
- Automatic backups to other devices and cloud
- Flexibility

Limitations of NAS

The weaknesses of a NAS are related to scale and performance. As more users need access, the server might not be able to keep up and could require the addition of more server horsepower. The other weakness is related to the nature of Ethernet itself. By design, Ethernet transfers data from one place to another via packets, dividing the source into a number of segments and sending them along to their destination. Any of those packets could be delayed, or sent out of order, and might not be available to the user until all of the packets arrive and are put back in order.

Any latency (slow or retried connections) is usually not noticed by users for small files, but can be a major problem in demanding environments such as video production, where files are extremely large and latency of more than a few milliseconds can disrupt production steps such as rendering.

Benefits of SAN

Because it's considerably more complex and expensive than NAS, SAN is typically used by large corporations and requires administration by an IT staff. For some applications, such as video editing, it's especially desirable due to its high speed and low latency. Video editing requires fair and prioritized bandwidth usage across the network, which is an advantage of SAN.

A primary strength of a SAN is that all of the file access negotiation happens over Ethernet while the files are served via extremely high speed Fibre Channel, which translates to very snappy performance on the client workstations, even for very large files. For this reason SAN is widely used today in collaborative video editing environments.

Summary of SAN Benefits

- Extremely fast data access
- Dedicated network for storage relieves stress on LAN
- Highly expandable
- OS level (block level) access to files
- High quality-of-service for demanding applications such as video editing

Limitations of SAN

The challenge of SAN can be summed up in its cost and administration requirements — having to dedicate and maintain both a separate Ethernet network for metadata file requests and implement a Fibre Channel network can be a considerable investment. That being said, SANs are really the only way to provide very fast data access for a large number of users that also can scale to supporting hundreds of users at the same time.

What's the Diff: NAS vs SAN

NAS	SAN
Typically used in homes and small to medium sized businesses.	Typically used in professional and enterprise environments.
Less expensive	More expensive
Easier to manage	Requires more administration
Data accessed as if it were a network-attached drive (files)	Servers access data as if it were a local hard drive (blocks)
Speed dependent on local TCP/IP usually Ethernet network, typically 100 megabits to one gigabit per second. Generally slower throughput and higher latency due to slower file system layer.	High speed using Fibre Channel, 2 gigabits to 128 gigabits per second. Some SANs use iSCSI as a less expensive but slower alternative to Fibre Channel.
I/O protocols: NFS, SMB/CIFS, HTTP	SCSI, iSCSI, FCoE
Lower-end not highly scalable; high-end NAS scale to petabytes using clusters or scale-out nodes	Network architecture enables admins to scale both performance and capacity as needed
Does not work with virtualization	Works with virtualization
Requires no architectural changes	Requires architectural changes
Entry level systems often have a single point of failure, e.g. power supply	Fault tolerant network with redundant functionality
Susceptible to network bottlenecks	Not affected by network traffic bottlenecks. Simultaneous access to cache, benefiting applications such as video editing.
File backups and snapshots economical and schedulable.	Block backups and mirrors require more storage.

Question 2

Not yet graded / 0 pts

Implement two functions that translate a logical block address (LBA) into a cylinder-head-sector (CHS) address, and reverse, translating a CHS address into its LBA equivalent.

```
typedef struct chs {
   int cylinder;
   int head;
   int sector;
} CHS_TYPE;

typedef long LBA_TYPE;

CHS_TYPE lba2chs(LBA_TYPE lba);

LBA_TYPE chs2lba(CHS_TYPE chs);
```

Using the algorithms, compute the location of the logical block 162656 in a hard drive with 200 cylinders, 128 sectors, and 5 two-sided platters.

Two-sided platters are served by two heads; one for each side.

Assume that a logical block is the same size as a sector and that there are no bad sectors on this perfect disk.

Your Answer:

STRUCT lba2chs(long addr)

{

STRUCT struct;

struct.cyl = addr/sectors per cylinder;

struct.head = (addr % sectors per cylinder) / sectors per track;

struct.sect = (addr % sectors per cylinder) % sectors per track;

return struct;

}

long chs2lba (STRUCT struct)

{

long logic = 0;

logic += struct.sect;

logic += struct.head * sectors per track;

logic += struct.cyl * sectors per cylinder

return logic;

}

The cylinder value is the number of tracks on one side of each platter.

200 cylinder means there 200 tracks on one side of the platter.

The number of sectors is 128 per track.

So total sector on each platter is 128*200= 25600 sectors per platter

we have to find the location of 162656 logical block (sector)

each platter can store 25600*2=51200 sectors or logical block

1st platter=51200

2nd platter =51200

3rd platter =51200

so total upto 3 platter including both side we get
51200+51200+51200=153600

we have to find the location of 162656 so it will be on the 4th platter.

Question 3

Not yet graded / 0 pts

Explain why some aspects of IO scheduling are addressed by the operating system rather than the disk controller. Give concrete examples.

What aspects of optimization would be more appropriate in the disk controller?

Your Answer:

The operating system knows about contraints that the disc controller does not, such as that demand paging takes priority over application I/O, or that writing is more important than reading if the cache is almost full.

The disk controllers can perform optimization on request ordering, coalescing, and sector numbering schemes

Question 4

Not yet graded / 0 pts

Describe:

1. advantages and disadvantages of solid state drives (SSDs) in comparison with disk hard drives,
2. what wear leveling is,
3. why is wear leveling used in the current SSDs, and
4. why wear leveling is a problem for ensuring information security.

Your Answer:

SSDs have faster starting, reading, and writing times with DRAM cache, has no noise that comes from a spinning disk, has decent mechanical reliability depending on the level type of the cells, file fragmentation is not

a problem since SSDs use direct access to any location, uses less power usually, and since problems usually happen with writes, writes can be repeated in good cells. Unfortunately they are still very expensive, have relatively low capacity, and have limited write cycles.

Wear leveling is circularly rewriting to every cell on the disk so that there is uniformal usage of all cells instead of having hotspots of rewriting. **Wear leveling** (also written as **wear levelling**) is a technique[1] (https://en.wikipedia.org/wiki/Wear_leveling#cite_note-Fundamental_Patent-1) for prolonging the **service life** (https://en.wikipedia.org/wiki/Service_life) of some kinds of erasable **computer storage** (https://en.wikipedia.org/wiki/Computer_storage) media, such as **flash memory** (https://en.wikipedia.org/wiki/Flash_memory), which is used in **solid-state drives** (https://en.wikipedia.org/wiki/Solid-state_drive) (SSDs) and **USB flash drives** (https://en.wikipedia.org/wiki/USB_flash_drive), and **phase-change memory** (https://en.wikipedia.org/wiki/Phase-change_memory). There are several wear leveling mechanisms that provide varying levels of longevity enhancement in such memory systems.

To reduce wear on current SSDs, wear leveling is used to reduce the number of hotspots.

Wear leveling causes problems in information security since you are rewriting to different cells, the old information stays in the old cell.

Question 5

Not yet graded / 0 pts

Assume that there are 256 cylinders in a disk and the current disk controller request queue is as follows:

```
67 223 45 183 233 12 238 54 29 218 156 48 22 191 78 73 224 251 2 15
184 101
```

Show the order in which the requests will be carried out for each of the following scheduling methods:

1. FCFS

2. SSTF (shortest-seek-time-first)
3. LOOK (elevator algorithm)
4. C-LOOK (circular unidirectional elevator algorithm)

NOTE: In `SCAN`, `C-SCAN`, `LOOK` and `C-LOOK`, assume that the first number is the current head position and that the head starts to move in the direction of increasing cylinder numbers.

Your Answer:

First Come First Serve (FCFS)

FCFS is the simplest **disk scheduling algorithm (https://www.geeksforgeeks.org/disk-scheduling-algorithms/)** . As the name suggests, this algorithm entertains requests in the order they arrive in the disk queue. The algorithm looks very fair and there is no starvation (all requests are serviced sequentially) but generally, it does not provide the fastest service.

Algorithm:

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. Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.
3. Increment the total seek count with this distance.
4. Currently serviced track position now becomes the new head position.
5. Go to step 2 until all tracks in request array have not been serviced.

Total number of seek operations = 2542
Seek Sequence is
223
45
183
233
12
238
54
29
218
156
48
22

191
78
73
224
251
2
15
184
101

Shortest Seek Time First (SSTF) –
Basic idea is the tracks which are closer to current disk head position should be serviced first in order to *minimise the seek operations*.

Advantages of Shortest Seek Time First (SSTF) –

1. Better performance than FCFS scheduling algorithm.
2. It provides better throughput.
3. This algorithm is used in Batch Processing system where throughput is more important.
4. It has less average response and waiting time.

Disadvantages of Shortest Seek Time First (SSTF) –

1. Starvation is possible for some requests as it favours easy to reach request and ignores the far away processes.
2. Their is lack of predictability because of high variance of response time.
3. Switching direction slows things down.

Total number of seek operations = 382
Seek Sequence is
67
73
78
101
54
48
45
29
22
15

12
2
156
183
184
191
218
223
224
233
238
251

LOOK Disk Scheduling Algorithm:
LOOK is the advanced version of **SCAN (elevator) disk scheduling algorithm (https://www.geeksforgeeks.org/scan-elevator-disk-scheduling-algorithms/)** which gives slightly better seek time than any other algorithm in the hierarchy *(FCFS->SRTF->SCAN->C-SCAN->LOOK)*. The LOOK algorithm services request similarly as SCAN algorithm meanwhile it also “looks” ahead as if there are more tracks that are needed to be serviced in the same direction. If there are no pending requests in the moving direction the head reverses the direction and start servicing requests in the opposite direction.
The main reason behind the better performance of LOOK algorithm in comparison to SCAN is because in this algorithm the head is not allowed to move till the end of the disk.

Algorithm:

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. The initial direction in which head is moving is given and it services in the same direction.
3. The head services all the requests 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 the absolute distance of the track from the head.

6. Increment the total seek count with this 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.

Initial position of head: 67
Total number of seek operations = 433
Seek Sequence is
73
78
101
156
183
184
191
218
223
224
233
238
251
54
48
45
29
22
15
12
2

C-LOOK (Circular LOOK) Disk Scheduling Algorithm:
C-LOOK is an enhanced version of both **SCAN** as well as **LOOK** disk scheduling algorithms. This algorithm also uses the idea of wrapping the tracks as a circular cylinder as C-SCAN algorithm but the seek time is better than C-SCAN algorithm. We know that C-SCAN is used to avoid starvation and services all the requests more uniformly, the same goes for

C-LOOK.
In this algorithm, the head services requests only in one direction(either left or right) until all the requests in this direction are not serviced and then jumps back to the farthest request on the other direction and service the remaining requests which gives a better uniform servicing as well as avoids wasting seek time for going till the end of the disk.

Algorithm-

1. Let Request array represents an array storing indexes of the tracks that have been requested in ascending order of their time of arrival and **head** is the position of the disk head.
2. The initial direction in which the head is moving is given and it services in the same direction.
3. The head services all the requests one by one in the direction it is moving.
4. The head continues to move in the same direction until all the requests in this direction have been serviced.
5. While moving in this direction, calculate the absolute distance of the tracks from the head.
6. Increment the total seek count with this distance.
7. Currently serviced track position now becomes the new head position.
8. Go to step 5 until we reach the last request in this direction.
9. If we reach the last request in the current direction then reverse the direction and move the head in this direction until we reach the last request that is needed to be serviced in this direction without servicing the intermediate requests.
10. Reverse the direction and go to step 3 until all the requests have not been serviced.

Initial position of head: 67
Total number of seek operations = 485
Seek Sequence is
73
78
101
156
183
184
191

218
223
224
233
238
251
2
12
15
22
29
45
48
54

SCAN (Elevator) algorithm

In SCAN disk scheduling algorithm, head starts from one end of the disk and moves towards the other end, servicing requests in between one by one and reach the other end. Then the direction of the head is reversed and the process continues as head continuously scan back and forth to access the disk. So, this algorithm works as an elevator and hence also known as the elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Total number of seek operations = 318
Seek Sequence is
54
48
45
29
22
15
12
2
0
73
78
101
156
183

184
191
218
223
224
233
238
251

What is C-SCAN (Circular Elevator) Disk Scheduling Algorithm?
Circular SCAN (C-SCAN) scheduling algorithm is a modified version of SCAN disk scheduling algorithm that deals with the inefficiency of SCAN algorithm by servicing the requests more uniformly. Like SCAN (Elevator Algorithm) C-SCAN moves the head from one end servicing all the requests to the other end. However, as soon as the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip (see chart below) and starts servicing again once reaches the beginning. This is also known as the "Circular Elevator Algorithm" as it essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

Algorithm:

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. The head services only in the right direction from 0 to size of the disk.
3. While moving in the left direction do not service any of the tracks.
4. When we reach at the beginning(left end) reverse the direction.
5. While moving in right direction it services all tracks one by one.
6. While moving in right direction calculate the absolute distance of the track from the head.
7. Increment the total seek count with this distance.
8. Currently serviced track position now becomes the new head position.
9. Go to step 6 until we reach at right end of the disk.
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.

Initial position of head: 67
Total number of seek operations = 238
Seek Sequence is
73
78
101
156
183
184
191
199
218
223
224
233
238
251
0
2
12
15
22
29
45
48
54

Question 6

Not yet graded / 0 pts

Design a data structure (or data structures) necessary to hold I/O requests for disk scheduler that can use FIFO (first-in-first-out; a.k.a.

NOOP), SSTF (shortest-seek-time-first), LOOK (elevator), and C-LOOK (circular elevator), and then write a program that implements the scheduler.

The program is given the input that specifies:

- the name of an algorithm to use, followed by
- the current head position given as a cylinder number, and finally followed by
- a sequence of I/O requests also given as corresponding cylinder numbers.

For example:

```
FIFO 67 223 45 183 233 12 238 54 29 218 156 48 22 191 78 73 224 251
2 15 184 101
```

states the program should use `FIFO` algorithm, the current head position is at cylinder `67`, and the remaining numbers are cylinder numbers associated with the I/O requests.

The output should be the sequence of requests ordered according to the specified algorithm.

For example, the sample input should generate the following output:

```
67 223 45 183 233 12 238 54 29 218 156 48 22 191 78 73 224 251 2 15
184 101
```

Your Answer:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;


// function to print request for LOOK
void LOOK(int head, vector<int> request)
{
// vector for cylinders left and right of head
vector<int> left, right;
for(int i=0; i<request.size(); i++)
{
```

```
if(request[i] < head)
left.push_back(request[i]);
if(request[i] > head)
right.push_back(request[i]);
}

// sort the 2 vectors
sort(left.begin(), left.end());
sort(right.begin(), right.end());

// start looking towards right => print right
for(int i=0; i<right.size(); i++)
cout << " " << right[i];
// look towards left => print left but in reverse order
for(int i=left.size()-1; i>=0; i--)
cout << " " << left[i];
}



// function to print request for SSTF
void SSTF(int head, vector<int> request)
{
int n = request.size();
// array to account for visited tasks
bool visited[n];
for(int i=0; i<n; i++)
visited[i] = false;

// loop for n times, find min for each run
int min = INT_MAX, idx = -1;
for(int j=1; j<=n; j++)
{
for(int i=0; i<n; i++)
{
// compare if the difference to reach the cylinder is minimum
if(!visited[i] && abs(request[i] - head) < min)
{
min = abs(head - request[i]);
idx = i;
}
}
// print min
```

```
cout << " " << request[idx];
// update head and visited
head = request[idx];
visited[idx] = true;
// clear min and idx
min = INT_MAX; idx = -1;
}
}


// function to print request for CLOOK
void CLOOK(int head, vector<int> request)
{
// vector for cylinders left and right of head
vector<int> left, right;
for(int i=0; i<request.size(); i++)
{
if(request[i] < head)
left.push_back(request[i]);
if(request[i] > head)
right.push_back(request[i]);
}

// sort the 2 vectors
sort(left.begin(), left.end());
sort(right.begin(), right.end());

// start looking towards right => print right
for(int i=0; i<right.size(); i++)
cout << " " << right[i];
// look towards left => print left (start from minimum => same order)
for(int i=0; i<left.size(); i++)
cout << " " << left[i];
}


//MAIN:

int main()
{
// take input
string type;
```

```
int head, tmp;
vector<int> request;
cin >> type;
cin >> head;
while(cin >> tmp)
{
request.push_back(tmp);
}

// print head cylinder
cout << head;

// call function according to request type
if(type == "FIFO")
{
// simply print the request in same order
for(int i=0; i<request.size(); i++)
{
cout << " " << request[i];
}
}
else if(type == "SSTF")
{
SSTF(head, request);
}
else if(type == "LOOK")
{
LOOK(head, request);
}
else if(type == "C-LOOK")
{
CLOOK(head, request);
}
cout << endl;
return 0;
}
```

Question 7

10 / 10 pts

I have submitted answers to all questions in this study set.

- (•) True
- ( ) False

Quiz Score: **10** out of 10