BUCKNELL UNIVERSITY
Computer Science

CSCI 315 Operating Systems Design

Disk Scheduling and Management

Disk Request

A request must specify:

- Whether the operation is input or output,
- The disk address for the transfer,
- The memory address for the transfer,
- The size of the data block to be transferred (number of bytes).

Disk requests are generated by processes. When the disk is busy serving a request, other incoming requests must be stored for later processing. For this purpose, the OS organizes these requests in a queue of pending requests for that disk unit. When the disk finishes serving a request, the OS much check the queue and if it is not empty, serve another request.

What policy should be used for dealing with the queue?

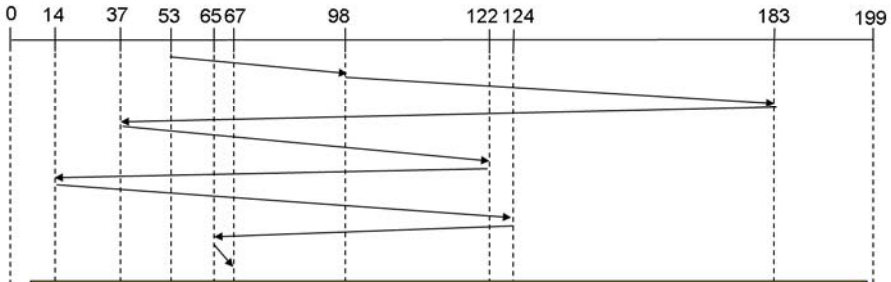
Disk Scheduling

FCFS Scheduling

Consider the following sequence of requests where each number corresponds to a disk cylinder:

98, 183, 37, 122, 14, 124, 65, 67

queue = 98, 183, 37, 122, 14, 124, 65, 67



Question: How much does the disk head travel to serve these requests?

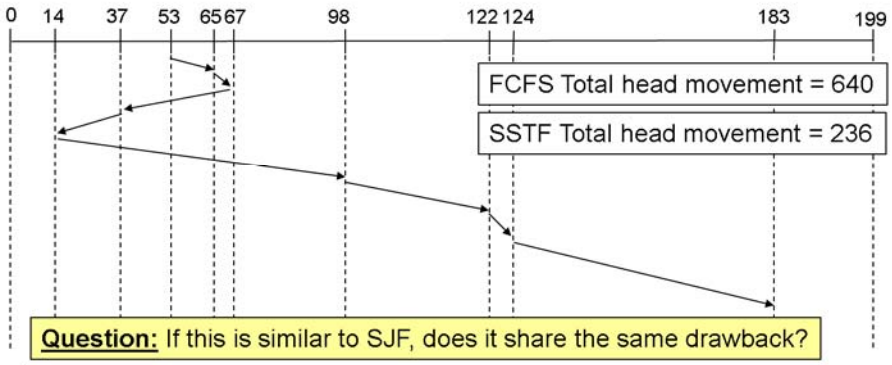
Question: What metric could one define to characterize performance here?

Question: How does this scheduling of requests affect the disk performance?

SSTF Scheduling

98, 183, 37, 122, 14, 124, 65, 67

queue = 65, 67, 37, 14, 98, 122, 124, 183



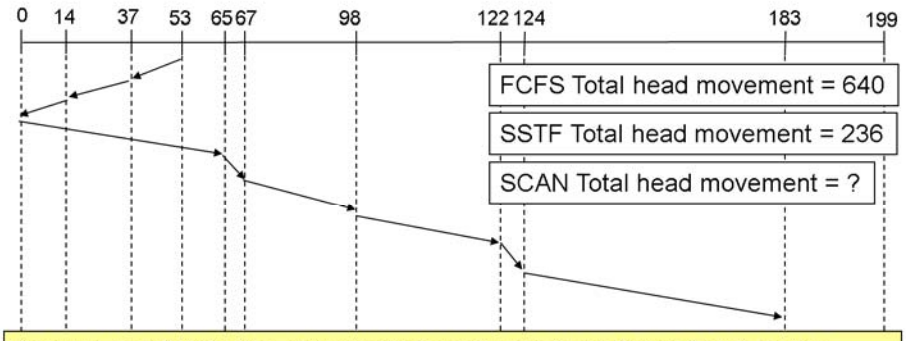
Question: If this is similar to SJF, does it share the same drawback?

Question: Is the performance of SSTF any better than that of FCFS?

Question: Is the performance of SSTF optimal?

SCAN Scheduling

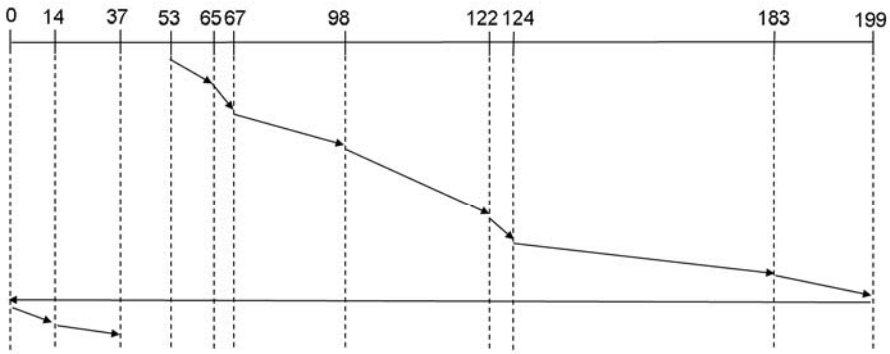
98, 183, 37, 122, 14, 124, 65, 67



Assume the distribution of requests for cylinders is uniform. Consider the density of requests when the head reaches one end and reverses direction. Hmmm… few requests will be right in front of the head… The heaviest density will be at the other end of the disk and those requests will have waited the longest. Why not just go there first?

C-SCAN Scheduling

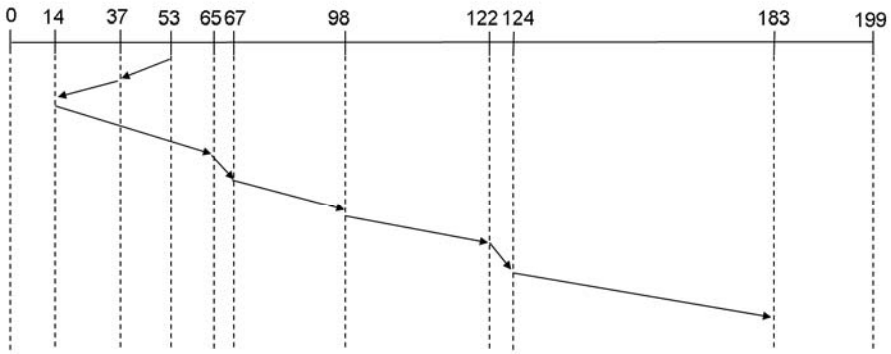
98, 183, 37, 122, 14, 124, 65, 67



When the head reaches the end of the disk, it immediately returns to cylinder 0. The algorithm essentially treats the cylinders as a **circular list** that wraps around from the final cylinder to the first one.

LOOK Scheduling

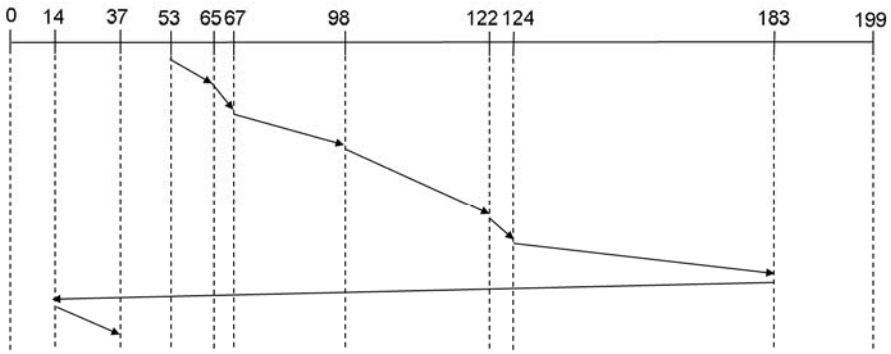
98, 183, 37, 122, 14, 124, 65, 67



Well, by now you have realized that scanning all the way to the extreme ends of the disk is perhaps wasteful. In practice, one could move the head only as far as the request that is farthest out.

C-LOOK Scheduling

98, 183, 37, 122, 14, 124, 65, 67



Choosing a Disk Scheduling Algorithm

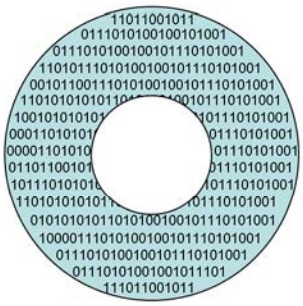
- The criteria should involve fairness and performance.
- Performance depends on the number and type of requests.
- Given a string of cylinder references, one could find the **optimal schedule** to serve them all using, for instance, *dynamic programming*. Remember, however, that **computing the optimal schedule takes time!**
- Another point to remember is that disk performance depends heavily on the file allocation method, on the location of directories and indices.
- If disk scheduling is a separate OS module, it can easily be replaced.
- Modern disk units don't disclose the physical location of disk blocks, so it can be challenging to do scheduling in the OS. Disk controllers, however, can handle disk scheduling themselves, lessening the burden on the OS.

Disk Formatting

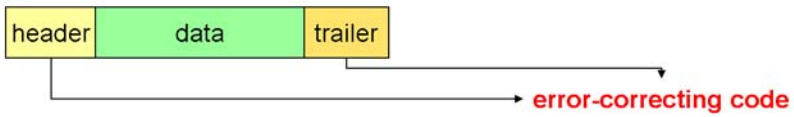
A brand new magnetic disk is a blank slate. There is no such thing as sectors. Tracks exist only as abstractions: we know they are actually created by how disk heads move over the disk surface (step motors).

Before a disk unit can be used as we've discussed, it is necessary to divide each track into sectors, what is known as ***low-level formatting***.

This formatting operation fills the disk with a special data structure for each sector:



Disk surface



Disk Formatting

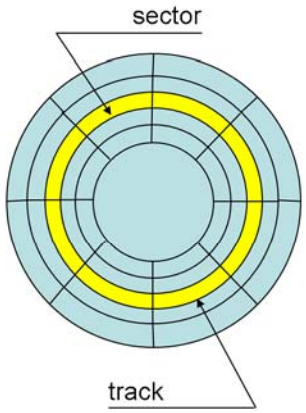
`sizeof(header)+sizeof(trailer)` = overhead

$$\text{ratio} = \frac{\text{overhead}}{\text{overhead} + sizeof(\text{data})}$$

`sizeof(data)=` { 256, 512, 1,024

Larger sectors => less disk space used for overhead.

Question: What is the potential drawback?



Organization of a disk surface

Disk Formatting

A disk can be further subdivided into ***partitions***: a contiguous group of cylinders.

Each partition is viewed by the operating system as a ***logical disk***.

Next, we'll look at the partition table from a running Linux installation.

Example: Partition Table

```
root# sbin/fdisk /dev/hda

   The number of cylinders for this disk is set to 4864.
   There is nothing wrong with that, but this is larger than 1024,
   and could in certain setups cause problems with:
   1) software that runs at boot time (e.g., old versions of LILO)
   2) booting and partitioning software from other OSs
     (e.g., DOS FDISK, OS/2 FDISK)

   Command (m for help): p

   Disk /dev/hda: 40.0 GB, 40007761920 bytes
   255 heads, 63 sectors/track, 4864 cylinders
   Units = cylinders of 16065 * 512 = 8225280 bytes

     Device Boot    Start       End    Blocks     Id    System
   /dev/hda1             1         4     32098+    de    Dell Utility
   /dev/hda2   *         5      3064  24579450     c    Win95 FAT32 (LBA)
   /dev/hda3          3065      3701   5116702+    83    Linux
   /dev/hda4          3702      4864   9341797+     f    Win95 Ext'd (LBA)
   /dev/hda5          3702      3832   1052226     82    Linux swap
   /dev/hda6          3833      4864   8289508+    83    Linux
```

Logical Formatting

Before a partition can be used, a file-system needs to be created on the partition. That means, file-system data structures are written to the disk.

Many different types of file-systems exist: **`ext2`**, **`ext3`**, **`swap`**, **`FAT`**, **`FAT32`**, **`ISO9660`**, etc. Read the man page of the `mount` command on Linux or Solaris for more information.

One may be able to use a partition as just a sequential array of logical blocks bypassing file-system data structures: this is called **raw I/O**. It also bypasses buffer caches, file locks, pre-fetching, file names, space allocation, and directories.

File System Types in Linux

Excerpt from the **`mount`** man page:

The argument following the -t is used to indicate the file system type. The file system types which are currently supported are: adfs, affs, autofs, coda, coherent, cramfs, devpts, efs, ext, ext2, ext3, hfs, hpfs, iso9660, jfs, minix, msdos, ncpfs, nfs, ntfs, proc, qnx4, ramfs, reiserfs, romfs, smbfs, sysv, tmpfs, udf, ufs, umsdos, vfat, xenix, xfs, xiafs. Note that coherent, sysv and xenix are equivalent and that xenix and coherent will be removed at some point in the future — use sysv instead. Since kernel version 2.1.21 the types ext and xiafs do not exist anymore.

For most types all the mount program has to do is issue a simple mount(2) system call, and no detailed knowledge of the filesystem type is required. For a few types however (like nfs, smbfs, ncpfs) ad hoc code is necessary. The nfs ad hoc code is built in, but smbfs and ncpfs have a separate mount program. In order to make it possible to treat all types in a uniform way, mount will execute the program /sbin/mount.TYPE (if that exists) when called with type TYPE. Since various versions of the smbmount program have different calling conventions, /sbin/mount.smb may have to be a shell script that sets up the desired call.

Boot Block

The system's primary disk unit contains a **boot block** that contains the bootstrapping program that loads the OS to memory. This program is invoked by the computer's minimal bootstrap program in ROM.

This boot block is often called the Master Boot Record (MBR).

Different operating systems treat the MBR in very different ways. Some a flexible enough to install a boot loader in the MBR, so that the disk can contain different OS in different disk partitions. The loader for each OS is then stored at the beginning of its own partition. Examples: Windows NT/2000/xp boot loader, Linux `lilo` and `grub`.

A "bootable" disk is one on which a boot block has been installed.

Bad Block Management

Certain disk blocks can't be used reliably due to problems with the magnetic media. Rather than just junk the entire disk, one can just mark the bad blocks and ignore them in the allocation procedures.

If blocks go bad after formatting, a disk check application can take care of marking them off (i.e. **`chkdsk`** in MS-DOS). The data in marked blocks is irretrievably lost.

Swap Space Management

Swap space can be allocated in two different ways:

1) From a single file in the normal file system. Normal file operations are used to manipulate this file. *External fragmentation* can become a problem requiring many seeks.

2) From a swap partition. This can be much faster than (1), but *internal fragmentation* can be a problem. This problem is relatively small considering that the lifetime of files on the swap space is small. Adding more swap, however, is not easy since it requires repartitioning the disk.