CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Files

Until now, we have focused on storage of programs and data in main memory

However, two issues with memory:

- not large enough to store everything system needs
- does not persist when computer turned off

Now discuss **secondary storage** -- large, long-term storage that persists between boots

Secondary storage also used as backing store for virtual memory

Simple example of secondary storage is hard disk

Recall that storage is a *hierarchy*

Hard disk is much larger and cheaper per bit than memory, but also much slower

How does secondary storage present to users?

We consider mechanics of actual hard disks in another lecture, but user never considers these (though OS designer might)

Main memory is treated as large array of bytes, but when was the last time you saw code that wrote to byte 50894 on disk? (Never)

OS provides *files* as abstraction for user/programmer to work with secondary storage

Definition (stolen from textbook):

File - collection of related information defined by its creator

OS allows programmer to create, read, write, delete, etc. files rather than work directly with storage on disk

Much like paged memory, OS presents convenient access to storage via files, but user has no idea how files actually stored on disk

Will discuss later different ways files can be mapped to locations on disk

OS also stores metadata about each file

- name -- human needs some way to refer to file
- unique identifier
- type
- location (where on disk is file actually stored?)
- size
- access rules
- time of last accesss/modification

All of this metadata must also be stored on disk somewhere -- needs to persist between boots

In most Linux filesystems, metadata for single file is stored in **inode**

Each file on disk gets unique inode (in a sense, having inode is what makes something a file)

We will see more about inode structure later, but inode is essentially space on disk that looks similar to

```
----------------------------------------------------------------
| owner | modified time | ... | permissions | location of file |
----------------------------------------------------------------
```

Remember -- inode is not a file. It stores information about a file.

OS generally provides support for following operations on files

- creating
- writing
- reading
- repositioning within file ("seek")
- deleting
- truncating (set size to 0 but keep other metadata)

Other operations, such as appending, can be built on these

Files can generally be opened by more than one process at once

If any processes are modifying file, this becomes critical-section problem

Essentially a reader-writer problem -- OS should allow inifinite readers but only one writer

OSes provide locks to control access to files

Shared lock can be acquired by several processes at once for reading

Exclusive lock acquired by just one process at a time for writing

Some systems do not provide shared locks

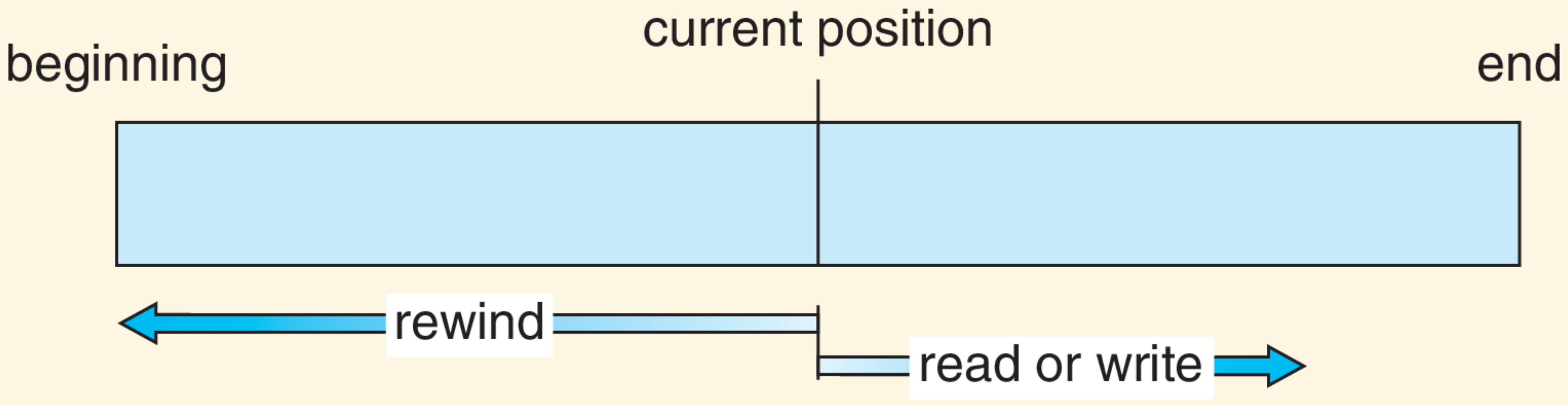
Another consideration is how files will be accessed

Sequential access -- information processed in order

Based on tape model of file

Allows `read_next` and `write_next` operations

Cannot jump to random spot in file without reading all lines between


current position
beginning
end
rewind
read or write

Direct access -- progammer can read or write to file in no particular order

Based on disk model of file

File viewed as numbered sequence of blocks or records

Sequential access can be simulated by direct access, but going the other way (sequential -> direct) is clunky

sequential access	implementation for direct access
reset	$cp = 0;$
read_next	read cp ; $cp = cp + 1;$
write_next	write cp; $cp = cp + 1;$

How files will be accessed impacts choice of how they are stored on disk