File System Interface

ECE595
Mar 29

Y. Charlie Hu

[week8] File System API

- OS provides the file system abstraction
- How do application processes access the file system?

Can we directly write() to a dir file?

- … using system calls (e.g. write)
- This is a permission question, not a privilege question
- The answer is no
- Explanation:
  - Any process independent of file system shouldn't be allowed to directly write in a directory the way they can write in files
  - Since such processes don't know how exactly the internals of directory are laid out. They should only see files, create files, and delete files.

System Calls to UNIX File Systems

- 19 system calls into 6 categories:

Return file desp.	Assign inodes	Set file attr.	Process input/ output	Change file system	Modify view of file system
open close creat pipe dup	creat link unlink	chown chmod stat fstat	read write lseek	mount umount	chdir chroot

Roadmap

- Functionality (API)
  - Basic functionality
    - Disk layout
    - File operations (open, read, write, close)
  - Directories
- Performance
  - Disk allocation
  - Buffer cache
  - File System interface
  - Disk scheduling
- Reliability
  - FS level
  - Disk level: RAID

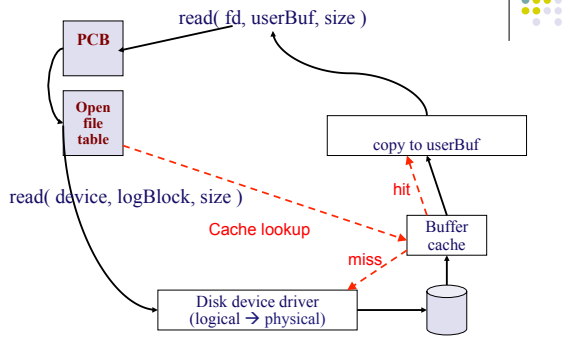
[week1] Roadmap for ECE469 lectures

Introduction to OS components
Individual components
- Process management
- Memory management
- File management
- Secondary-storage management

- (Device management)
- (shell)
- (Networking)

Hardware support for OS interspersed (before relevant topics)

Reading A Block



Modern disk drives are addressed as large one-dimensional arrays of logical blocks

File System Interface

- How do application programs typically access file data?
  - Explicit read/write operations (conventional)

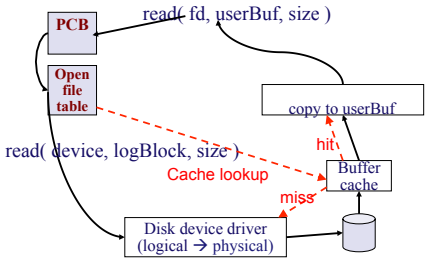
Read / Write Interface

- File data is explicitly copied between disk file and process memory
- Programs cannot directly access file data

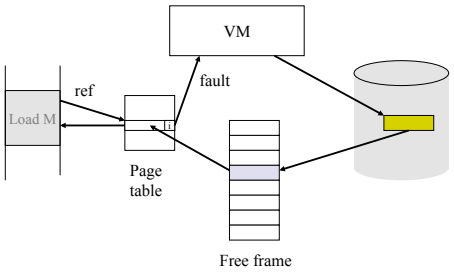
```
FileDescriptor fhandle;
int offset, length;
char *buffer = (char *) malloc(1000000);

fhandle = open("pathname");
read(fhandle, offset, buffer, length);
{read file data in buffer to do important computation};
Write(fhandle, offset, buffer, length)
close(fhandle);
```


Read / Write Interface – Problem 1

- Potential for double copies in mem

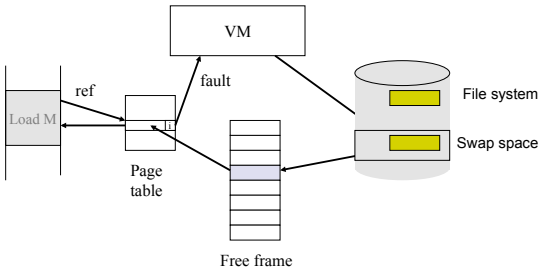


Essence of Demand Paging



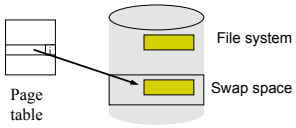
Disk is the backing store

[week6] Demand Paging, Page Fault Handling

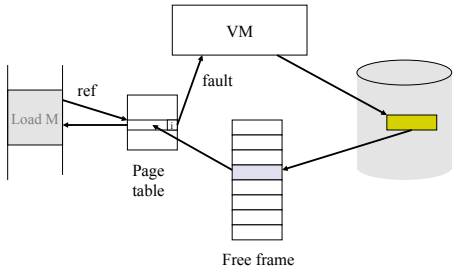


Read / Write Interface – Problem 2

- Potential for double paging on disk
  - process pages containing file data are paged out to paging space, leading to redundant copies of file data on disk

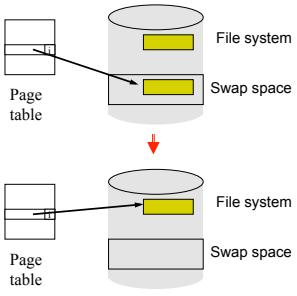


Essence of Demand Paging



Disk is the backing store

What if



Memory-mapped Files

- File is “mapped” into application’s address space
  - by initializing virtual memory so that the file (directly) serves as backing store for a region of the application’s address space

Memory-mapped Files (cont')

- Elegant integration of file system and virtual memory

```
FileDescriptor fhandle;
int offset, length;
char *address;

fhandle = open("pathname");
mmap(fhandle, offset, address, length);
{read/write file data by accessing memory range
        [address, address + length]};
munmap(address, length);
close(fhandle);
```

This is like after address = malloc()

Implementation of mmap()

Memory-mapped Files

- File is "mapped" into application's address space
  - by initializing virtual memory so that the file (directly) serves as backing store for a region of the application's address space
- File data is *demand paged* upon access to the mapped file
  - No double paging on disk
- Memory-mapped files do not go through buffer cache
  - No double copy in mem
    - Program accesses file data directly

Effects and Semantics of Memory-mapped Files

- Processes that map the same file share physical memory that caches file data
- Writes may not be immediately writen to the file on disk
  - Update periodically
  - Closing the file results in writing all to disk and removing the VM mapping

Fun with memory-mapped files

- Inter-process communication
  - Virtual addresses of diff processes mapped to the same file
- File copying as Memory copying
  - Map files to virtual addresses
  - Do memory copying

Reading

- Chapters 11-12