Operating Systems
CSE 511

Assignment 3 Directions

File Systems

Instructor: Ruslan Nikolaev

Assignment 3

- The document will be released tonight
  - You can already get started with the code https://classroom.github.com/a/uBT6S-5T
  - Academic integrity is very important! (do not use any online implementations)
    - Use our slides and official manuals!
  - You need to implement lock-based and lock-free queues and stacks
    - Use pthread_mutex for lock-based
    - Use C11 atomics for lock-free

Assignment 3

- Review pthread_mutex documentation
- Review C11 atomics (stdatomic.h)
  - atomic_compare_exchange_strong (CAS)
  - atomic_load (LOAD)
  - atomic_store (STORE)
  - You do not need atomic_compare_exchange_weak!

Assignment 3

- For ABA queues/stacks
  - Create two stacks/queues (e.g., store two head and two tail pointers for queues)
    - e.g., can create a separate structure with head/tail and then create ‘aq’ and ‘fq’ instances inside the queue object
      - Allows to use the same code for aq and fq
    - Recycle elements through ‘fq’

Assignment 3

- [Hint] Stacks: **need tagging only for 'top'**
  - *Not needed for 'next' (think why)*
- [Hint] Queues: **need tagging everywhere (head, tail, next)**
- When you need tagging, create a separate structure like this

```
typedef struct aba_ptr {
    void *ptr; // or any other pointer type
    size_t tag;
} aba_ptr;
```

Assignment 3

- Just pass pointers/values of aba_ptr directly to atomic_compare_exchange_strong, atomic_load, atomic_store like you pass pointers/values to regular pointers
- Use _Atomic(…) to encapsulate custom atomic types, e.g., _Atomic(queue_node *) head or _Atomic(aba_ptr) aba_head.

Filesystem interface

- What is a file? Named storage
- File operations:
  - read(fd, buffer, length)
  - write(fd, buffer, length)
  - fd = open(pathname, flags)
  - close(fd)

syscalls (unbuffered)

  - fread(buffer, size, num, file)
  - fwrite(buffer, size, num, file)
  - file = fopen(pathname, mode)
  - fclose(file)

Buffered C library

File descriptor (fd)

- Purpose: identify file being accessed
- What actually is a file descriptor?
  - 0-indexed number – index of file descriptor table

File descriptor	Referencing
0 (stdin)	Console
1 (stdout)	Console
2 (stderr)	Console
3	My_open_file.txt
…	…

- Important points:
  - Each process has its own table
  - There is a limit (default 1024)

Filesystems

- Purpose: making storage usable

1. Tracking file information (metadata)
2. Mapping files to blocks (i.e., LBN)
3. Tracking/allocating free blocks

File metadata

- Metadata: data that's not the actual data

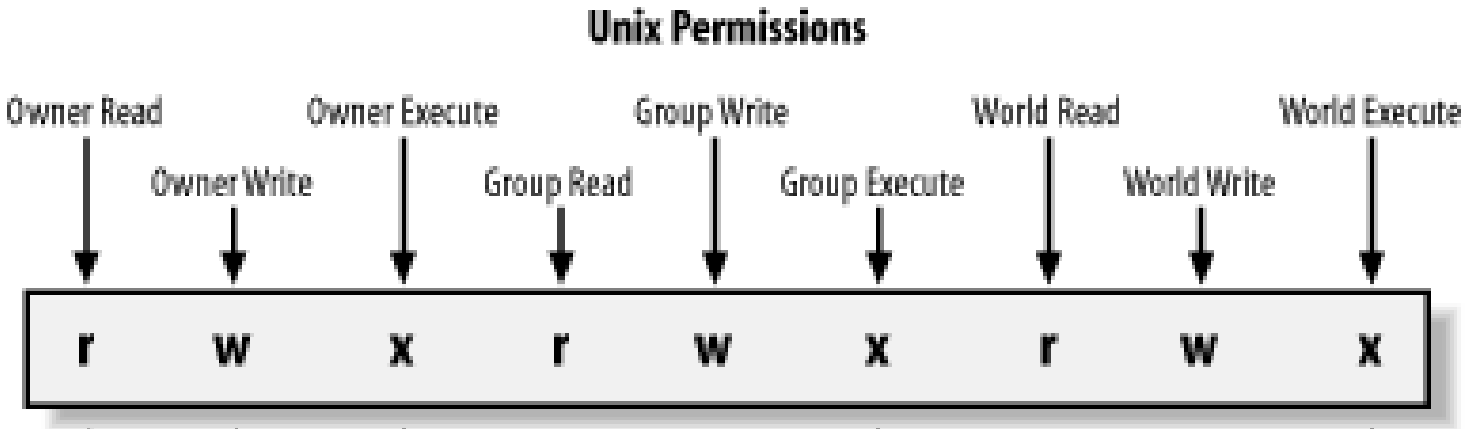
- Examples:
  - Filename

  - Size of file
  - Last access timestamp
  - Last modification timestamp
  - File owner
  - Permission
  - Location of data

inode

File permission

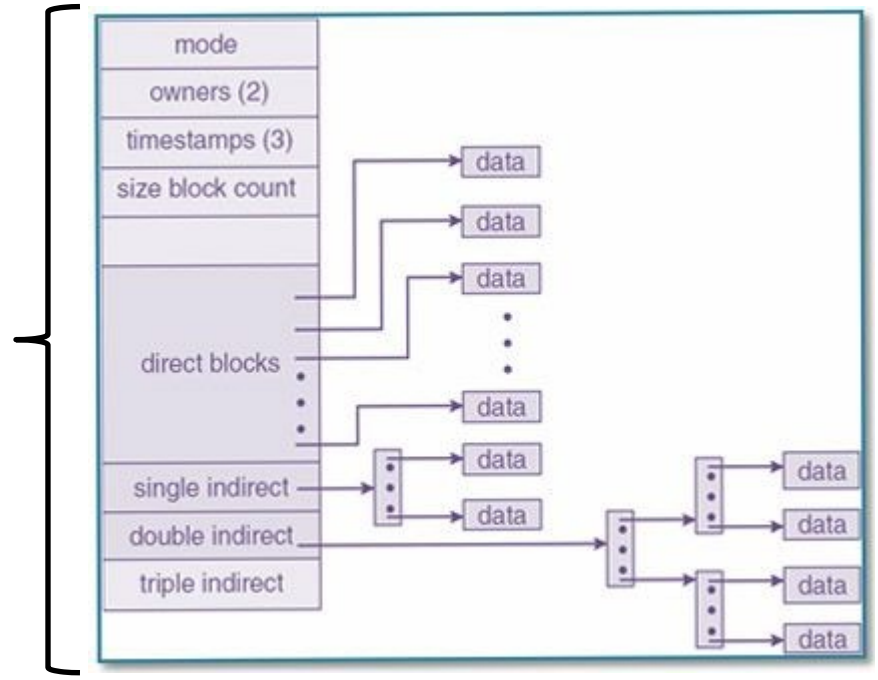
- Files have owners/groups
  - Use chown to modify owner/group
- File permission
  - Use chmod to modify permission



Mapping files to blocks (i.e., LBN)

- Simple: linked-list of data blocks (FAT)
- Better: indexed table (ext2/ext3)
- Problem: how big of a table?
- Solution: multi-level index

ext4 uses an "extent tree" instead



Tracking/allocating free blocks

- Simple: free list (slow)
- Better: bitmap (easy to scan)
- Complex: list of extents (i.e., ranges of free blocks)