Project 03

Virtual Memory & File system

Due date
2025. 06. 18 23:59

Overview

- Copy-on-Write Fork
  - Memory management optimization through lazy allocation
- Large Files
  - File system extension with doubly-indirect blocks
- Symbolic Links
  - Advanced file referencing and path resolution

Project 03

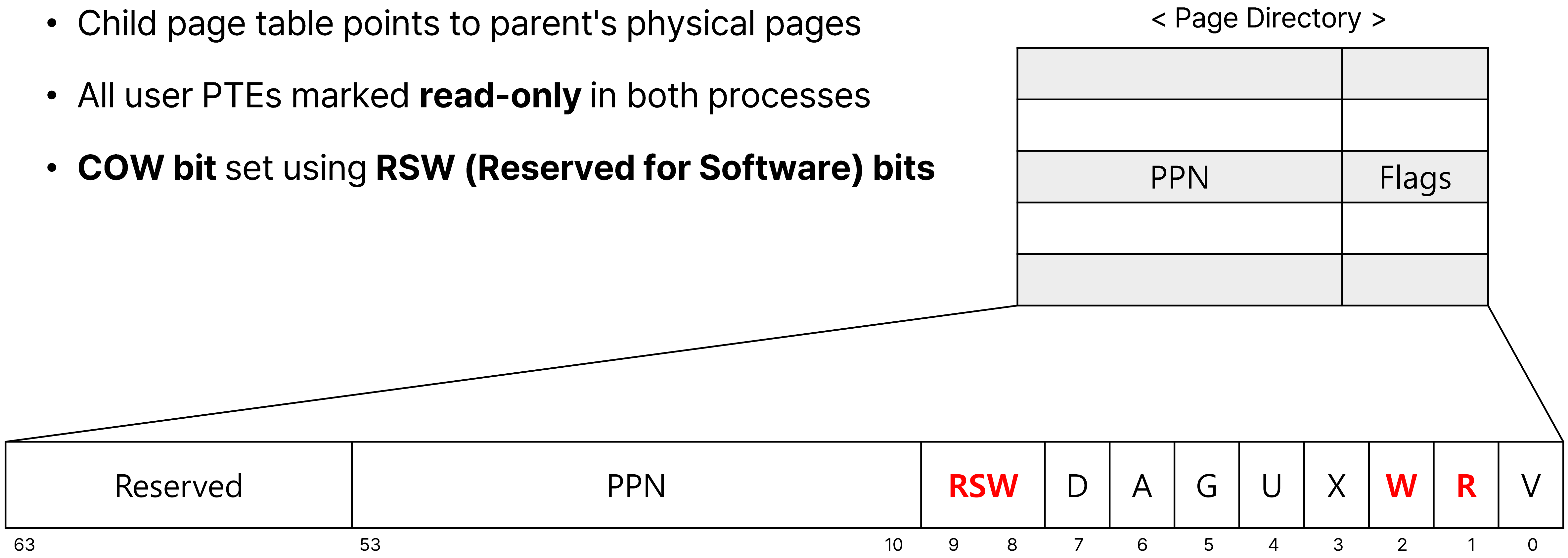
Copy-on-Write Fork

Overview

- Problem: Inefficient memory copying in fork()
- Current fork() copies all parent memory immediately
  - Wasteful when followed by exec()
  - Can fail due to insufficient memory
- Solution: Copy-on-Write (COW)
  - Share pages between parent and child initially
  - Copy pages only when modified

COW Fork Architecture

- Initial State:
  - Child page table points to parent's physical pages
  - All user PTEs marked **read-only** in both processes
  - **COW bit** set using **RSW (Reserved for Software) bits**



COW Fork Architecture

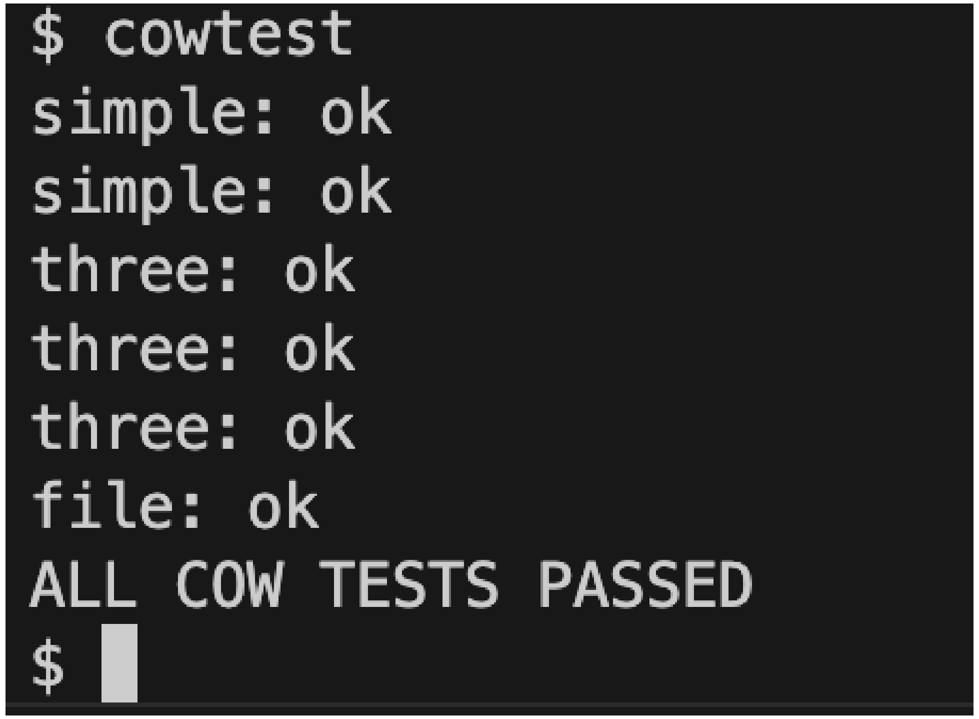
- Write Access:
  - **Page fault** triggered on write attempt
  - Kernel allocates new physical page
  - Original page copied to new page
  - PTE updated with write permission
- Reference Counting:
  - Track number of processes sharing each page
  - Free pages only when reference count reaches zero

Implementation Tasks

- Core Functions to Modify:
  - `uvmcopy()`: Share pages instead of copying
  - `usertrap()`: Handle COW page faults
  - `copyout()`: Apply COW logic to kernel operations
  - `kalloc()/kfree()`: Implement reference counting
- Key Challenges:
  - Distinguishing COW faults from other page faults
  - Proper synchronization for reference counts
  - Handling out-of-memory conditions
  - Maintaining compatibility with existing code

Cow Test Result

- simpletest() - Basic COW Memory Allocation and Behavior Verification
- threetest() - Multi-Process COW Stress Testing
- filetest() - System Call Integration with COW (copyout() Compatibility)



```
$ cowtest
simple: ok
simple: ok
three: ok
three: ok
three: ok
file: ok
ALL COW TESTS PASSED
$ 
```

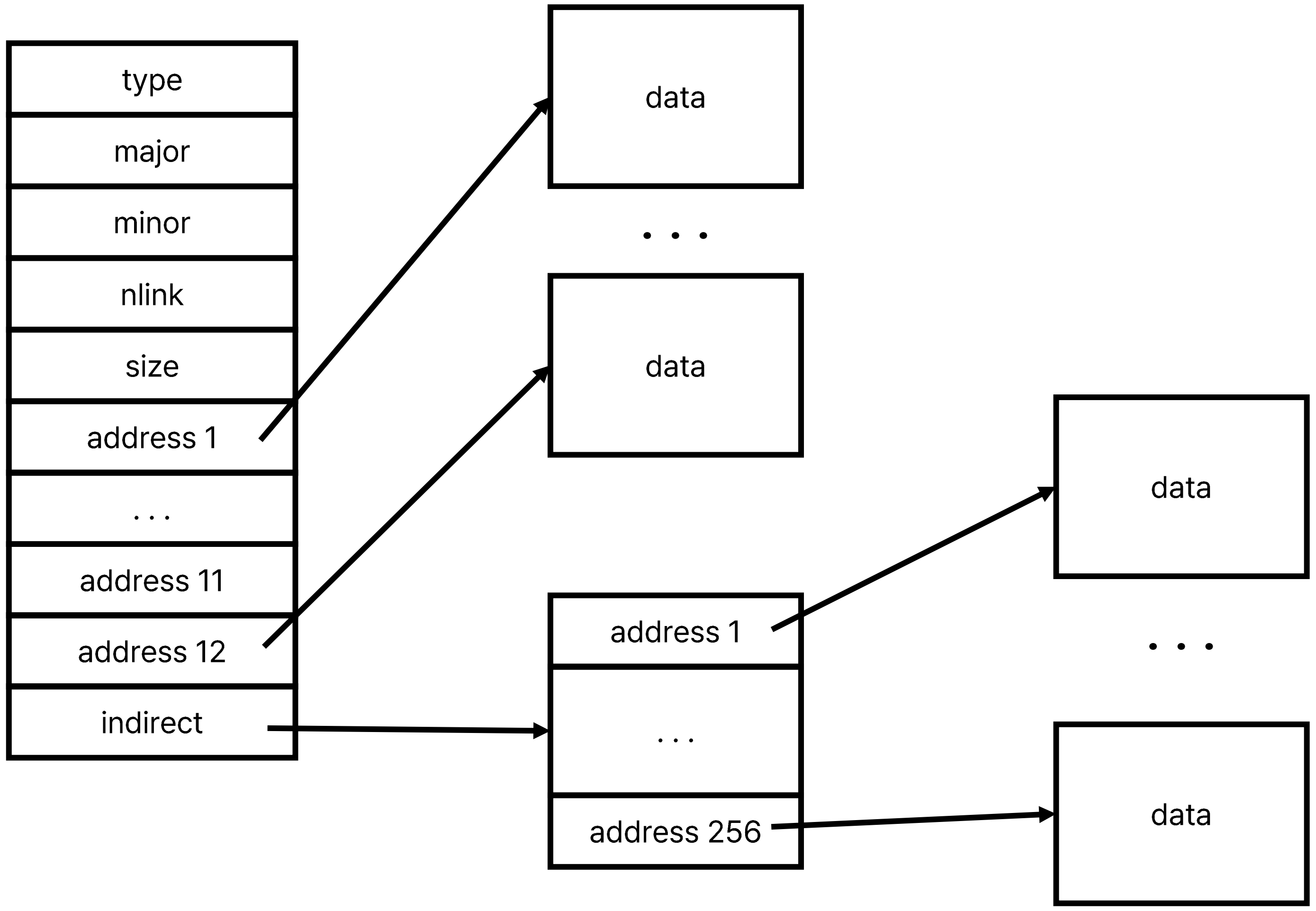
Project 03

Large files

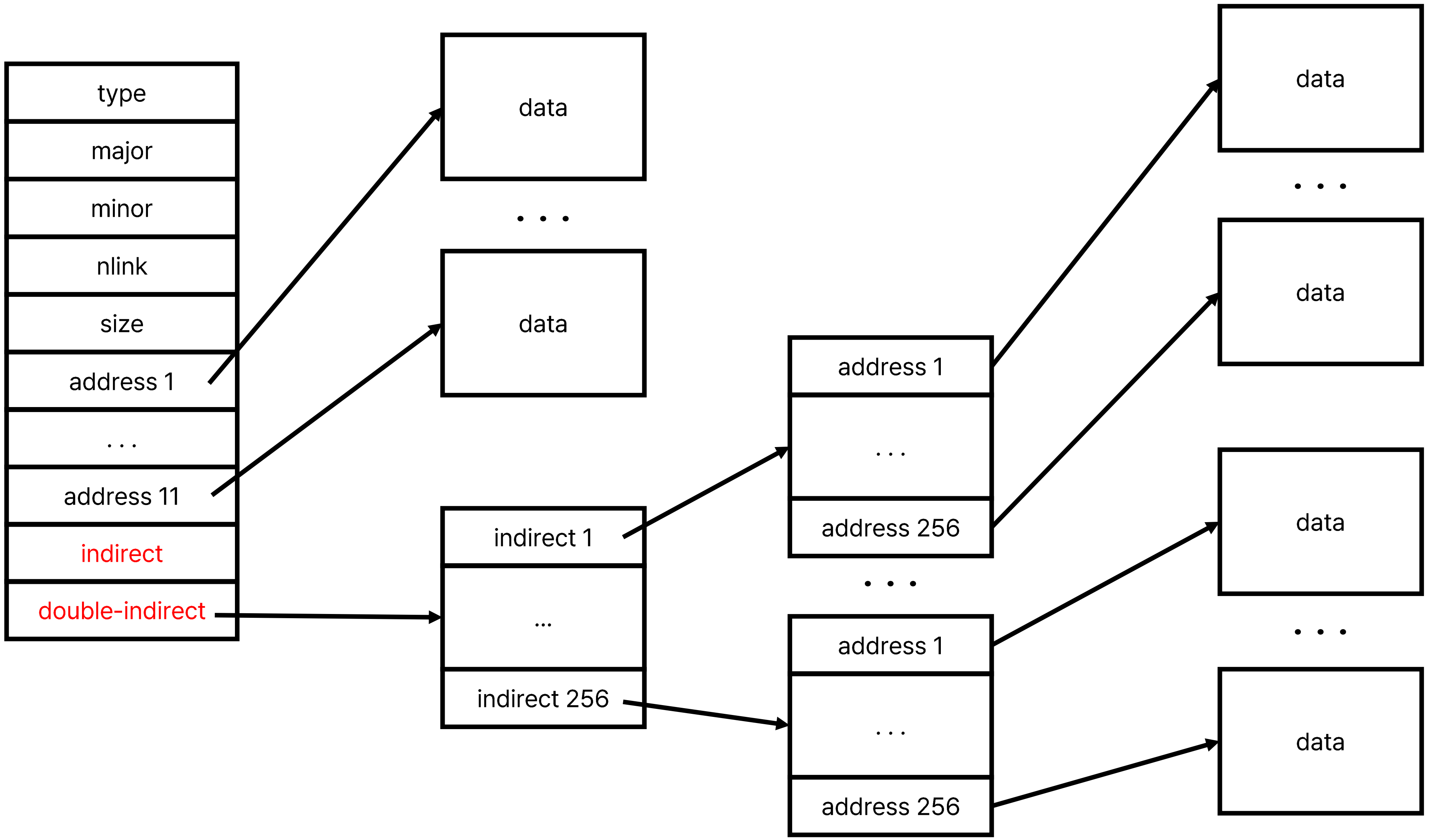
Overview

- Problem: Limited file size in xv6
  - Current limit: 268 blocks (268 KB)
  - Structure: 12 direct + 1 singly-indirect
  - Insufficient for modern applications
- Solution: **Doubly-Indirect Blocks**
  - Support files up to 65,803 blocks (~64 MB)
  - Structure: 11 direct + 1 singly-indirect + 1 doubly-indirect
  - Maintains backward compatibility

Current Block Addressing Hierarchy



Block Addressing Hierarchy

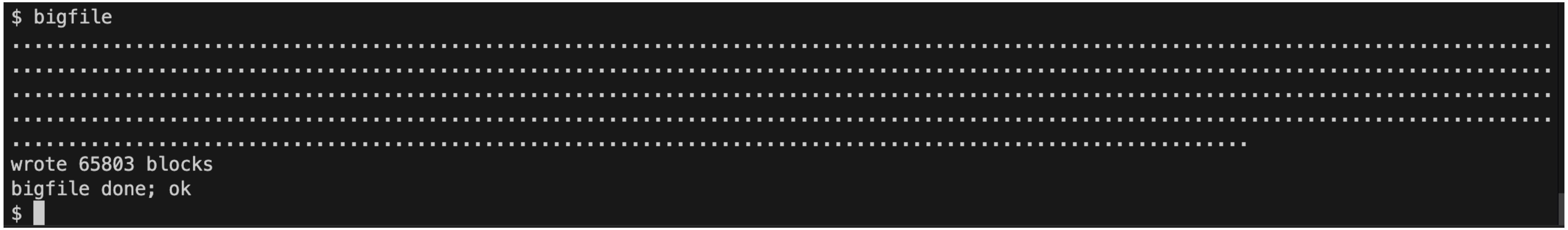


Implementation Tasks

- Constants and Structures:
  - Change `FSSIZE` from 2000 to 200000
  - Change `NDIRECT` from 12 to 11
  - Update `MAXFILE` calculation
  - Modify `addrs[]` array in dinode and inode structures
- Core Functions:
  - `bmap()`: Implement doubly-indirect address translation
  - `itrunc()`: Free doubly-indirect blocks properly
  - `create()`: Support new inode structure Critical

Largefile Test Result

- Large File Creation and Sequential Writing
- File Size Validation
- Data Integrity Verification



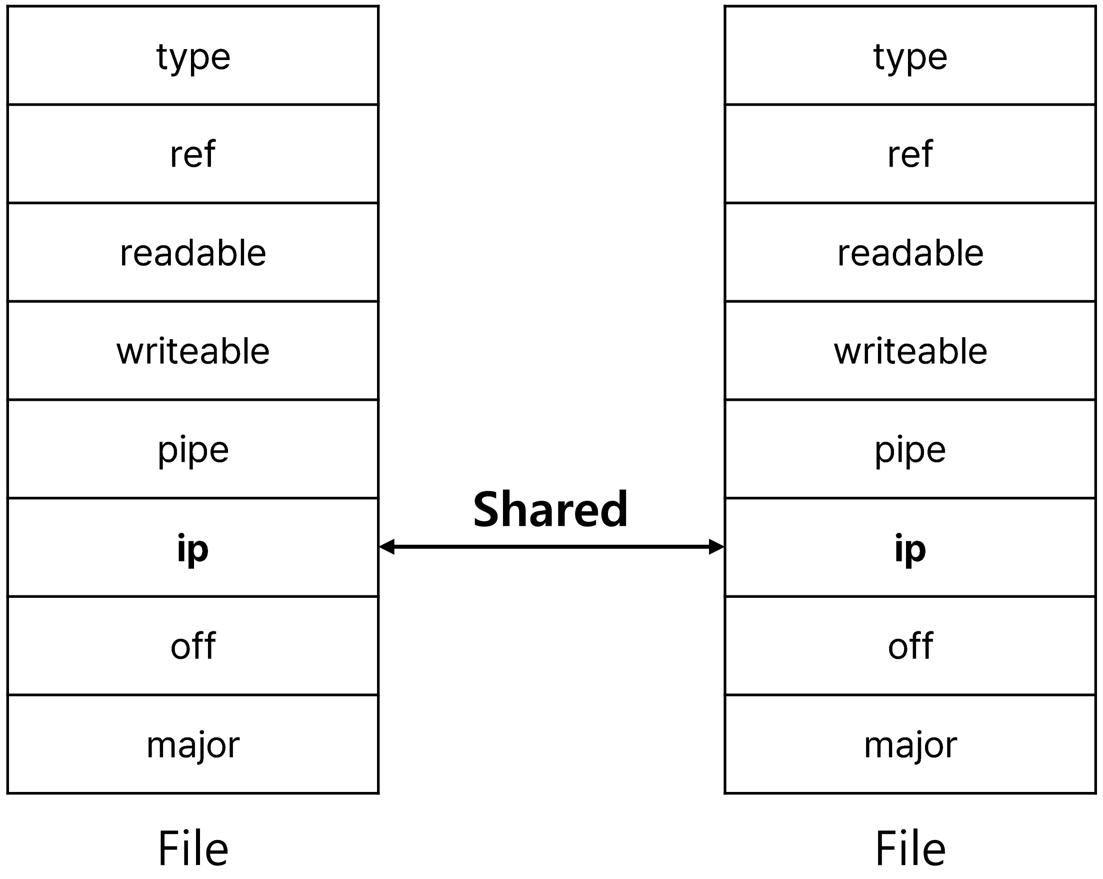
```
$ bigfile
.........................................................................................................................................
.........................................................................................................................................
.........................................................................................................................................
.........................................................................................................................................
..............................................................................................................
wrote 65803 blocks
bigfile done; ok
$ 
```

Project 03

Symbolic links

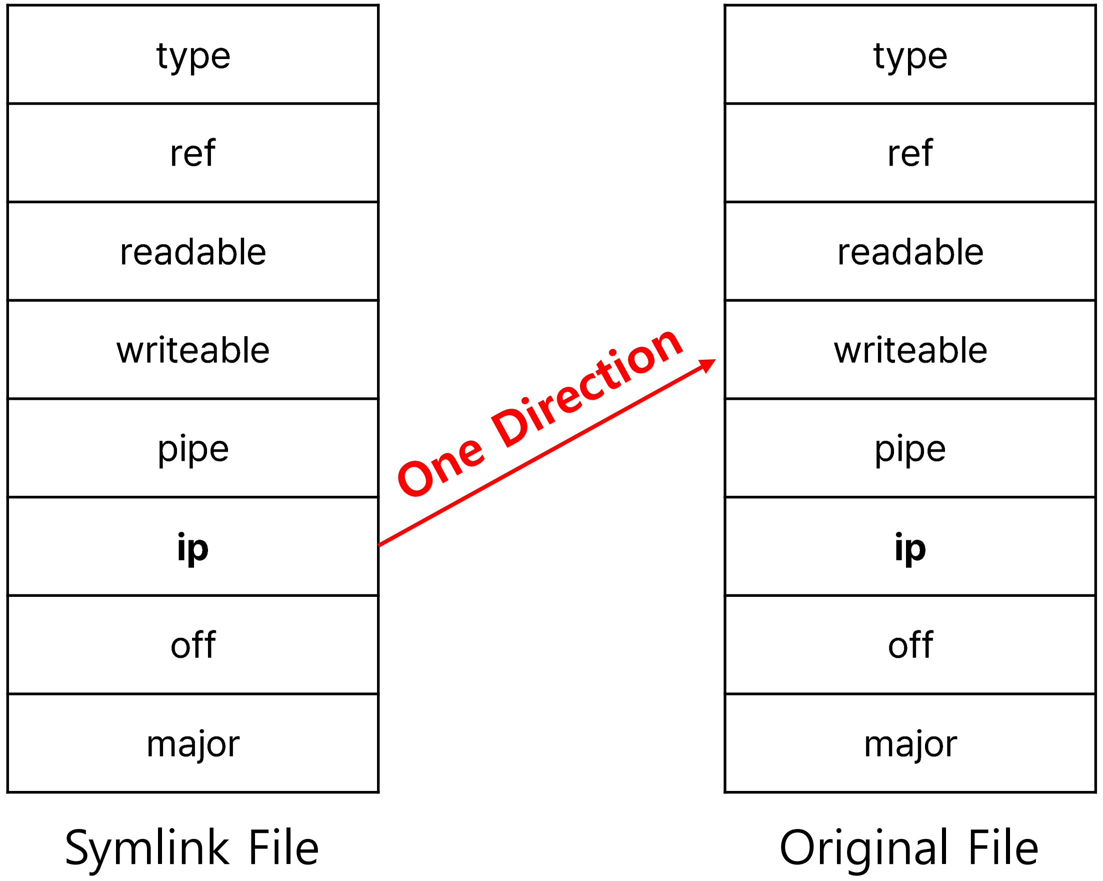
Overview

- Problem: Rigid file referencing in xv6
  - Only hard links available (inode-based)
  - Cannot reference across file systems
  - No support for broken or flexible links



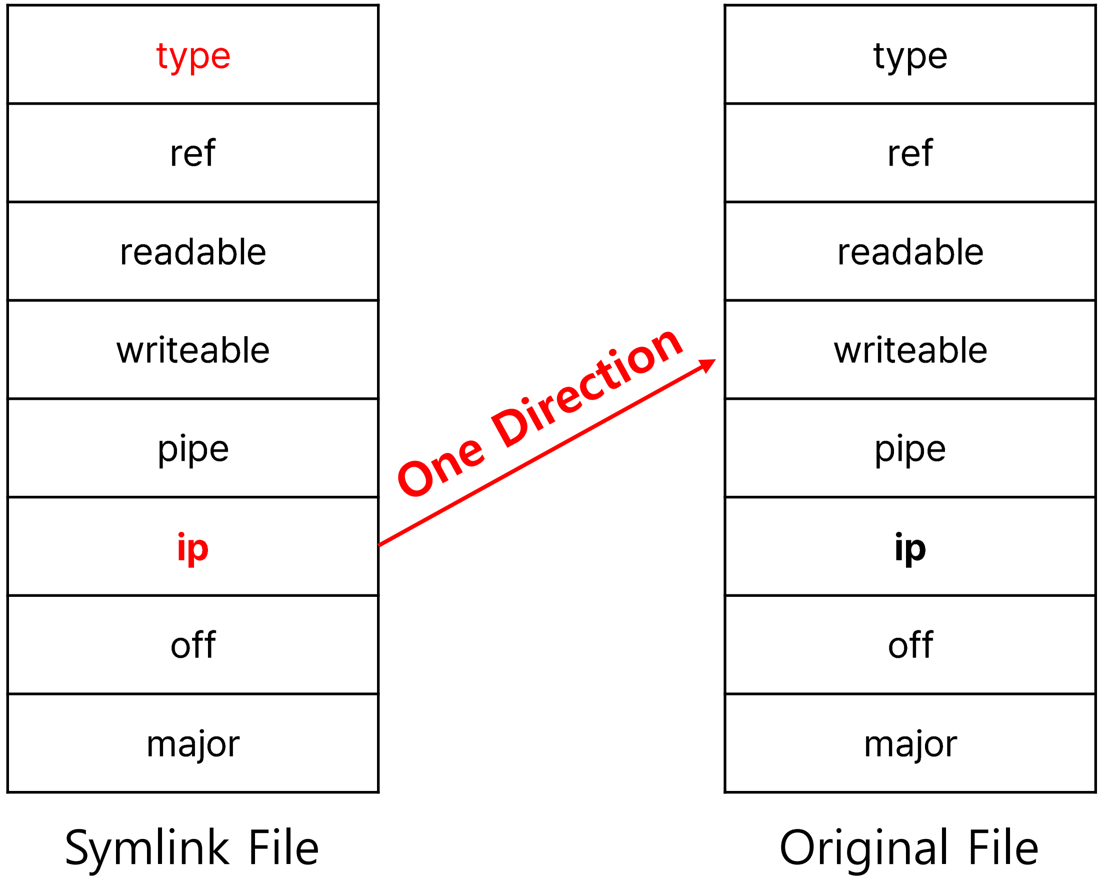
Overview

- Problem: Rigid file referencing in xv6
  - Only hard links available (inode-based)
  - Cannot reference across file systems
  - No support for broken or flexible links
- Solution: **Symbolic Links**
  - Pathname-based file references
  - Can point to non-existent files
  - Support for cross-filesystem references
  - Enhanced file system flexibility



Symbolic Links Architecture

- File Type Extension:
  - New `T_SYMLINK` file type
  - Target path stored in symlink's data blocks
  - Distinguished from regular files and directories
- System Call Interface:
  - `symlink(target, path)`: Create symbolic link
  - Modified `open()`: Follow or access links directly
  - `O_NOFOLLOW` flag: Open link itself, not target
- Path Resolution:
  - Recursive link following (max depth: 10)
  - Cycle detection to prevent infinite loops
  - Proper error handling for broken links

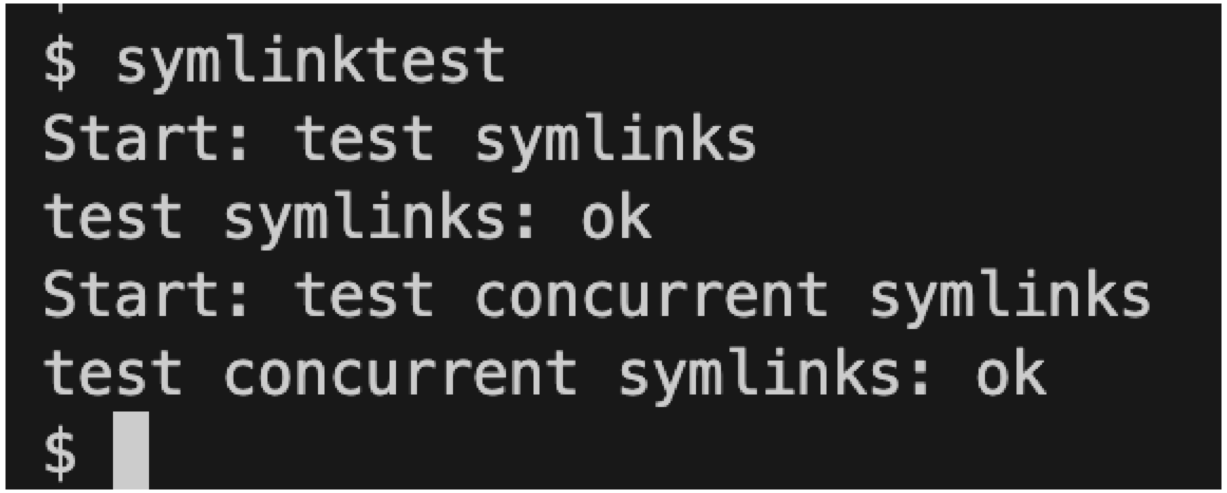


Implementation Tasks

- New Components:
  - `T_SYMLINK` file type in kernel/stat.h
  - `O_NOFOLLOW` flag in kernel/fcntl.h
  - `symlink()` system call implementation
- Modified Functions:
  - `sys_open()`: Implement link resolution logic
  - `create()`: Support symbolic link creation

Symlink Test Result

- testsymlink() - Core Functionality Testing
  - Basic Link Creation and Access
  - Broken Link Handling
  - Circular Reference Detection
  - Non-existent Target Linking
  - Chain Link Resolution
- concur() - Concurrent Testing



```
$ symlinktest
Start: test symlinks
test symlinks: ok
Start: test concurrent symlinks
test concurrent symlinks: ok
$
```

Evaluation

- **Completeness** The xv6 operating system must function correctly according to the specification requirements.
- **Wiki & Comment** Grading will be based on the wiki documentation, so the wiki should be written in as much detail as possible.
- **Deadline** The submission deadline must be strictly observed. After the deadline, your GitHub writing permissions will be revoked.
- **DO NOT SHARE AND COPY!!**

Wiki

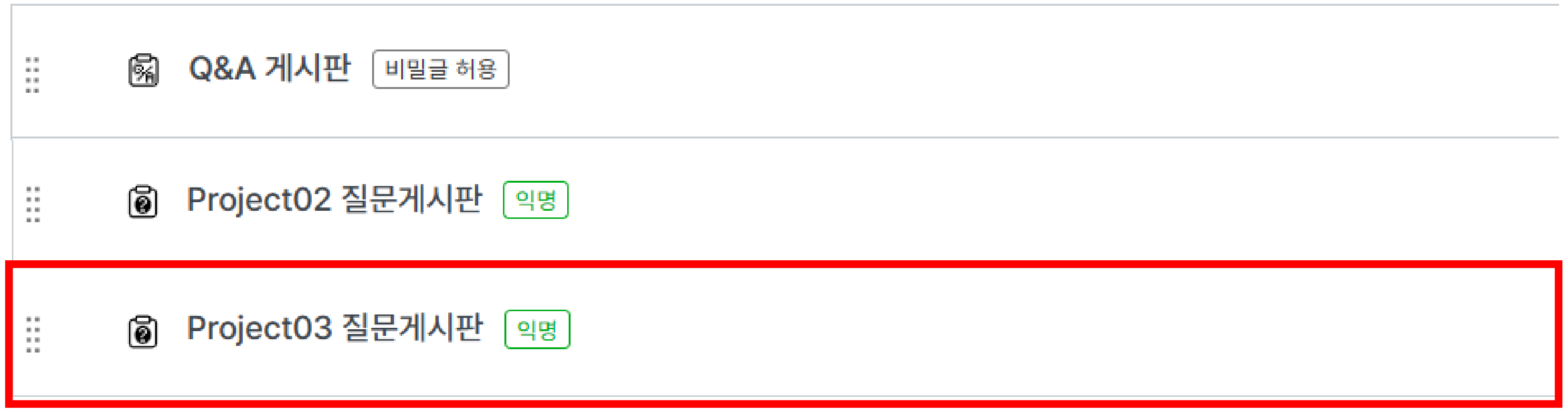
- **Design** Outline your implementation approach for meeting the project requirements
- **Implementation** Explain key code modifications and their purpose, focusing on changes from the original code.
- **Results** Show evidence of successful implementation with:
  Compilation process, Screenshots of working code, Explanation of program flow
- **Troubleshooting** Describe any problems encountered, solutions applied, and any unresolved issues.
- Additional content may be included if relevant.

Submission

- Submit your implemented code and wiki through GitHub.
  - **Refer to the announcement and create a new repository.**
  - Rename the repository to “**project03-[student ID]**”
- The wiki file should be named "**OS_project03_[class number]_[student ID].pdf**".
- Submission deadline: **June 18, 2025, 23:59**
  - Late submissions will be accepted via **email** until **June 19, 2025, 23:59**, but will only receive **50%** of the possible score.

Q&A

- For questions related to the project, please use the question board (Project 03 Question Board) on the LMS.
- Questions sent by email will **not** be answered.
- For questions not related to the project, please use the Q&A board or send an email.



Q & A