CIS3110 Lecture 13 Summary - March 25, 2025

Part 1: Assignment 4 Overview - FAT12 File System (File-System Structure 14.1-14.6)

- Assignment 4 involves reading blocks of data from a Microsoft FAT12 file system
- The process is similar to assignment 1 where a block of memory had to be interpreted

Key File System Components:

- **Boot block**: Contains information about the file system structure
- **FAT blocks**: The File Allocation Table that tracks block allocation
- **Root directory blocks**: Contains directory entries
- **Data blocks**: Where actual file data is stored

Working with the File System:

- Reading a file requires finding a directory entry, locating the first block, then "chasing" through the FAT
- The instructor provides a data structure that aligns with Microsoft's directory entry format
- Students must only modify the fat12.c file, not the header file with predefined structures

Testing Utilities:

- **mtools**: Provides utilities like mdir (view directory), mcopy (copy files), mdel (delete files)
- **Sample file systems provided**:
  - small_files.fd0: Contains some sample files
  - blank.fd0: A formatted but empty file system
  - bad.fd0: Intentionally broken file system (FAT loops on itself)

Debugging Tools:

- **hexdump**: Shows file contents as hex values and ASCII
- **od (octal dump):** Alternative tool that can display data in different formats (long, short)

FAT12 Format:

- The "12" in FAT12 indicates 12 bits per FAT entry
- Demonstrates efficient bit packing (12-bit entries packed into 3 bytes)

- Suitable for smaller file systems; larger file systems use FAT16 or FAT32

Part 2: File System Buffer Caches (File-System Internals 15.5, 15.6, 15.8)

Two Types of Buffer Caches:

- **Physical block buffer cache**: Contains metadata specific to file system type
  - For FAT: Contains FAT table and root directory
  - For Unix: Contains inodes, indirect blocks, etc.
- **Logical block buffer cache**: Contains actual file data blocks
  - These represent logical parts of files regardless of physical layout
  - Used for both local and network file systems

Reading Process:

1. Calculate logical block number from file offset
2. Check if block is in logical buffer cache
3. If not found:
   - Allocate space in buffer cache
   - Use metadata to determine physical block location
   - If block is a "hole", fill with zeros
   - Otherwise, issue disk read to controller
4. Process blocks while waiting for I/O to complete
5. When interrupt happens, copy data to user program
6. Program continues execution

Writing Process:

1. Calculate logical block number from file offset
2. Check if block is in logical buffer cache
3. If not found:
   - Allocate buffer
   - If past end of file or hole, fill with zeros
   - If partial write to existing block, read the block first
4. Copy user data to logical block buffer
5. Mark buffer as modified

6. Schedule write to disk
7. Update metadata if needed

I/O Performance Optimization Techniques (I/O Systems 12.2-12.5):

- **Synchronous I/O**: Process waits until data is safely on disk (safer but slower)
- **Asynchronous I/O**: Program continues after data is in buffer cache (faster but riskier)
- **Read-ahead**: Pre-populate buffer cache with blocks program will likely need next
- **Delayed write/write-behind**:
  - Combine multiple writes to same block
  - Implement with block boundary detection and timers
  - Reduces metadata updates (like modification times)
  - Saves SSD lifespan by reducing write frequency

Part 3: Introduction to Security (Security 16.1, 16.2, 16.3)

Security Concerns in Operating Systems:

- Protecting the system from users and users from each other
- Protecting against network-based attacks
- Limiting access to resources to prevent damage

Types of Malicious Software:

- **Virus**: Malicious code attached to legitimate software
  - Detected by fingerprinting known byte patterns
  - Prevention through education and virus scanners
- **Trojan**: Custom-created malicious program
  - Often distributed through unofficial sources
  - App stores help screen for these
- **Worm**: Self-replicating software that spreads without user intervention
  - Example: 1988 Morris Worm that crashed the early internet
  - Typically exploits buffer overflows in network-facing programs

Buffer Overflow Exploits:

- How they work:
  - Program reads data into fixed-size buffer without bounds checking

    - Excess data overflows into return address on stack
    - When function returns, execution jumps to attacker's code
- Countermeasures:
    - Using safe functions (fgets instead of gets)
    - Using heap memory (malloc) instead of stack for buffers
    - Address Space Layout Randomization (ASLR): randomizing memory segments
    - Stack execution controls
    - Proper bounds checking

User-Based Protection:

- Identify users by unique user IDs
- Implement privilege levels (at least root/administrator and regular users)
- Run potentially vulnerable programs (like web servers) as restricted users
- Log important system changes
- Implement resource quotas
- Require proper authentication

The instructor mentioned that next lectures will cover authentication and password encryption techniques. The final week of the term will be dedicated to review.