Multi-Client FTP System

A comprehensive file transfer protocol system designed for concurrent client handling and robust file operations.

AVIK MANDAL
ANKIT KUMAR
NAVODIT VERMA
KAVY VAGHELA
UTKARSH KUMAR

Agenda

- Project Overview
- Core Principles & Architecture
- Technical Implementation
- Features & Demonstration
- Applications & Future Enhancements

Project Overview

Multi-client FTP system

Client-server architecture

Enhanced Synchronization mechanisms

Concurrent file operations

Command line Interface

Core Principles

Robust File Transfer

Chunk-based protocol with error detection and retry mechanisms for failed transfers.

Concurrent Client Handling

POSIX threads for simultaneous connections and scalable design for multiple clients

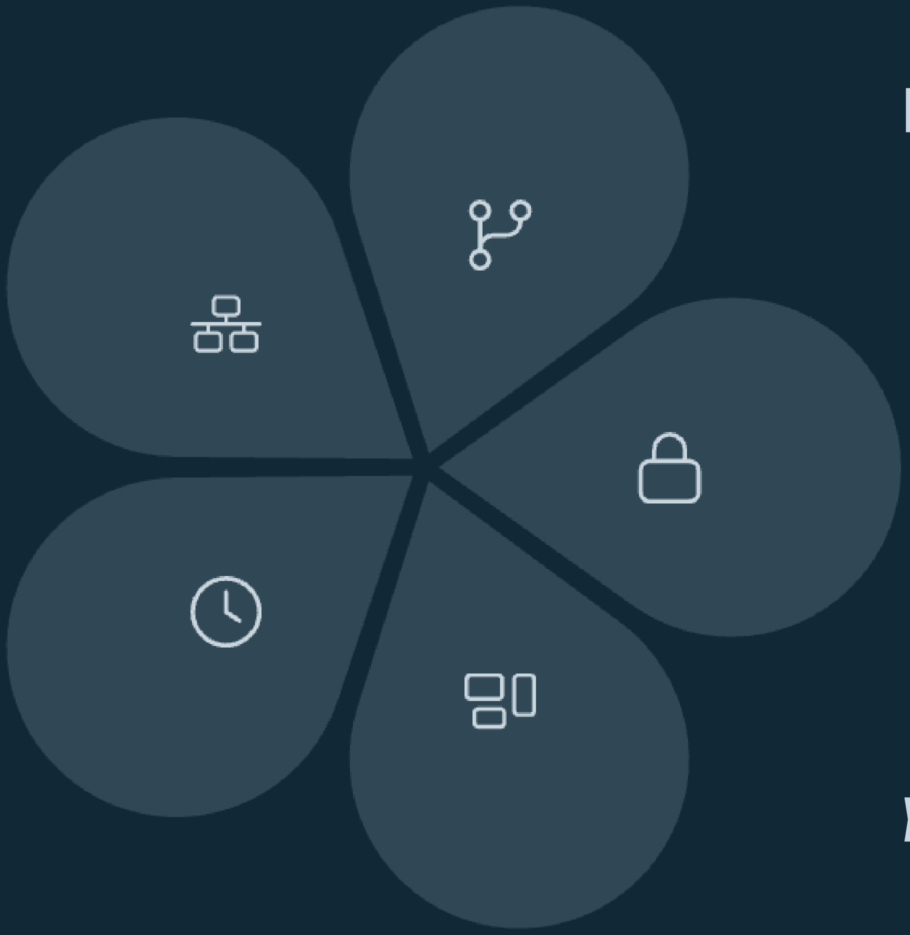
Command-based Interface

Simple text-based protocol supporting both local and remote operations.

Resource Management

Reader-Writer locks with deadlock prevention and fine grained control over file access

Technologies Used



Socket Programming

Multi-threading

Mutexes & Condition Variables

Lock Timeout Mechanisms

Exponential Backoff

Architecture Overview

Client Component

- Command interface
- Local file operations
- Server communication

Network Layer

- Data transmission
- Protocol handling

Server Component

- Multi-threaded connection handling
- File operation processing
- Lock management

Synchronization Mechanism

Timeout Mechanisms

Prevents indefinite waiting

Deadlock Prevention

Detection and resolution strategies

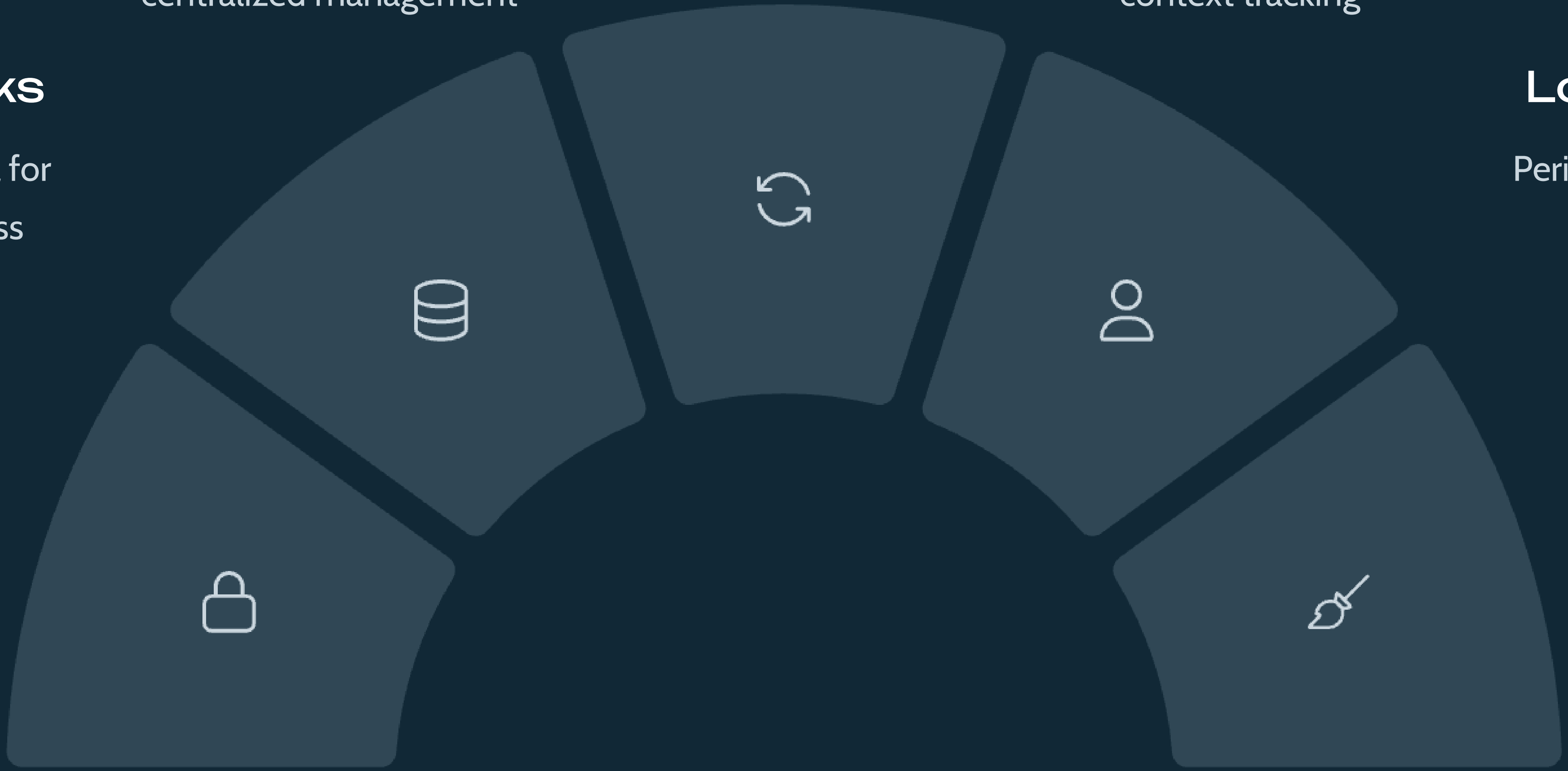
Priority Management

Prevents reader/writer starvation

Reader-Writer Lock System

Multiple readers, exclusive writers

Thread Safety Mechanisms



Per-File Locks

Fine-grained control for individual file access

Global Lock Registry

Protected by mutex for centralized management

Atomic Counters

For statistics without contention

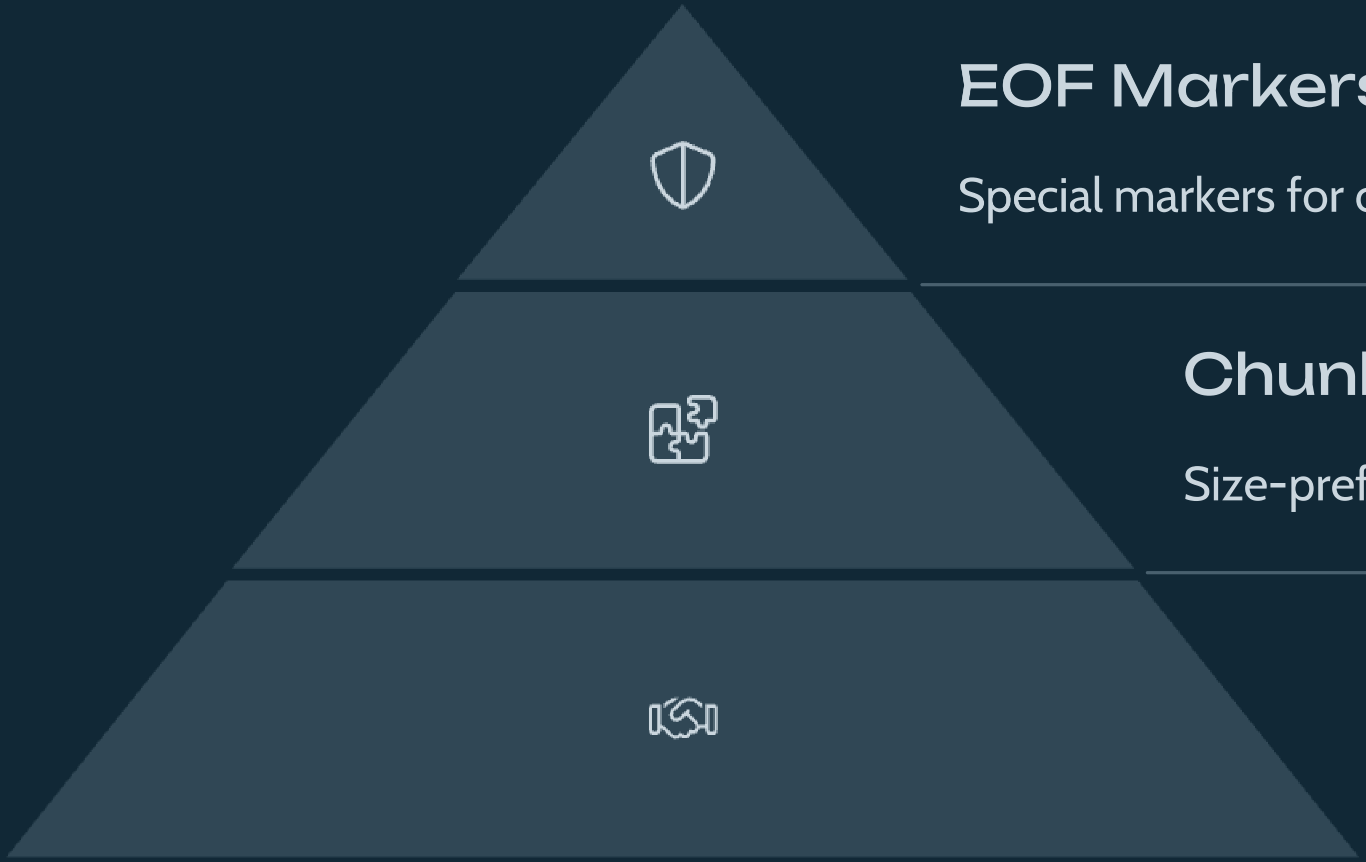
Thread Local Information

For debugging and context tracking

Lock Cleanup

Periodic maintenance of abandoned locks

Socket Communication



EOF Markers

Special markers for command output termination

Chunked Data Protocol

Size-prefixed chunks for file transfers

Reliable Data Transfer

Custom send_all/recv_all functions with guarantees

Our socket communication layer ensures data integrity through custom send_all and recv_all functions that guarantee complete data transmission even under network instability. The chunked data protocol efficiently handles file transfers by breaking data into manageable size-prefixed chunks. Special EOF markers clearly indicate command output termination, preventing ambiguity in communication streams.

Technical Implementation Details

```cpp
// Server-side
void *handle_client(void *client_socket){
    int sock = *(int *)client_socket;
    string client_directory = ".";
    // Handle Client code
}

// File Transfer
bool send_all(int sock, const void* buf, size_t len) {
    const char* p = static_cast<const char*>(buf);
    while (len > 0) {
        int bytes = send(sock, p, len, 0);
        if (bytes <= 0) return false;
        p += bytes;
        len -= bytes;
    }
    return true;
}
```

```cpp
// Client-side
void send_command(int sock, string command){
    char buffer[BUFFER_SIZE];
    send(sock, command.c_str(), command.size(), 0);
    // Send command code
}

// File Transfer
bool recv_all(int sock, void* buf, size_t len) {
    char* p = static_cast<char*>(buf);
    while (len > 0) {
        int bytes = recv(sock, p, len, 0);
        if (bytes <= 0) return false;
        p += bytes;
        len -= bytes;
    }
    return true;
}
```

Advanced Synchronization Features

Reader-Writer Fairness

Our system implements a balanced approach between readers and writers to prevent starvation, ensuring all operations get fair access to resources.

Deadlock Prevention

We employ timeout-based detection and global statistics monitoring to identify and prevent potential deadlocks before they occur.

Exponential Backoff

Smart retry mechanisms with increasing delays and jitter provide efficient resource utilization during high contention periods.

Lock Implementation Details

```
struct rwlock_t {
    pthread_mutex_t mutex;          // Basic lock for the structure
    pthread_cond_t readers_cv;      // For readers to wait
    pthread_cond_t writers_cv;      // For writers to wait
    int readers_count;              // Number of active readers
    int writers_count;              // Number of active writers
    int waiting_writers;            // Number of waiting writers
    bool writer_active;             // Is there an active writer?
    unordered_map<string, bool> locked_files; // Track which files are currently being written to
    unordered_map<string, int> file_readers;  // Track number of readers per file
};
```

Server Implementation

Socket Creation and Binding

The server initializes by creating a socket and binding it to a specified port, preparing to accept incoming client connections.

Client Connection Handling

When clients connect, the server spawns dedicated threads to handle each connection independently, enabling multi-client support.

Command Processing

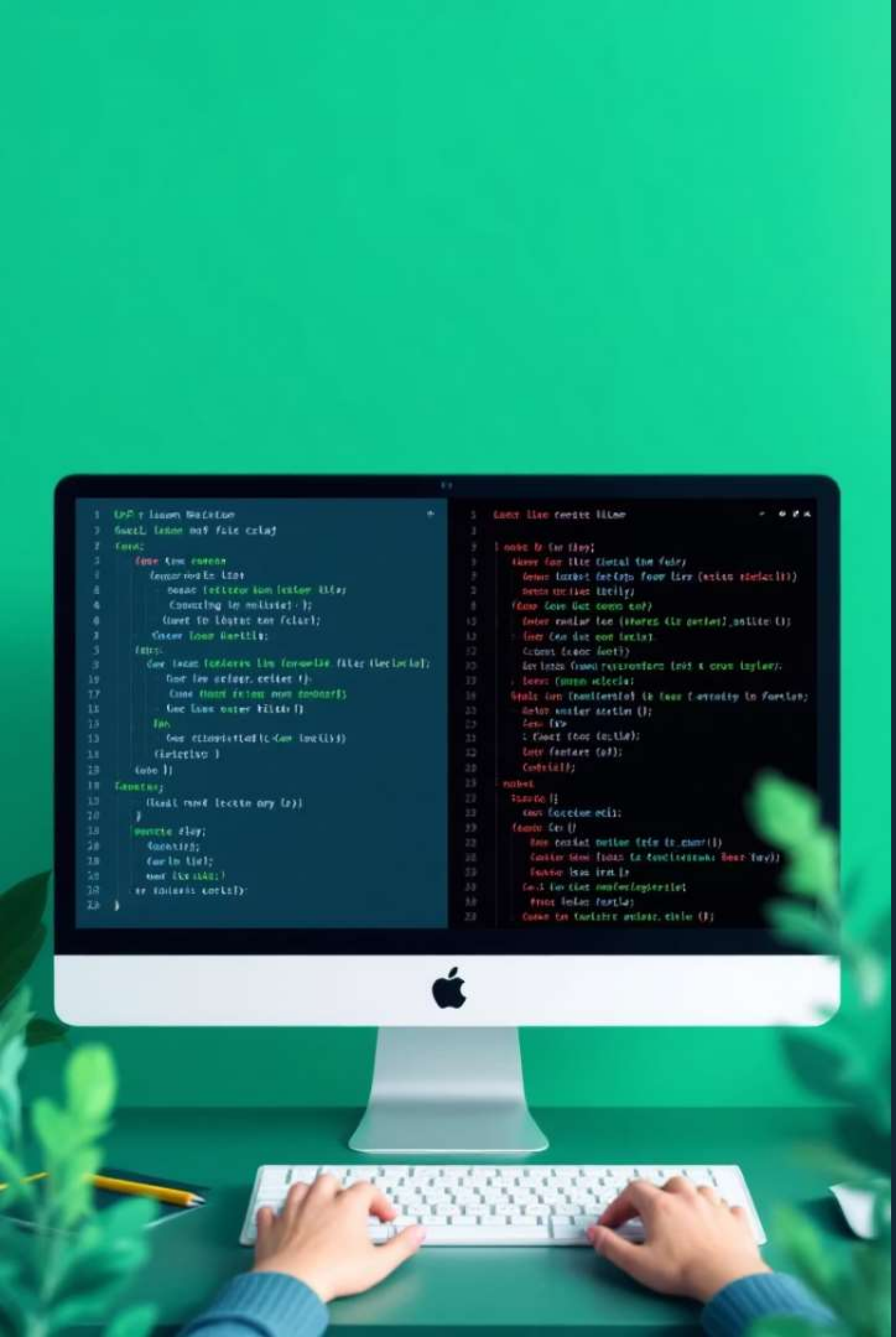
The server parses and executes commands received from clients, including file operations and directory navigation.

File Operation Handling

Implements directory listing (ls), permission changes (chmod), directory navigation (cd, pwd), and file transfers (put, get).

Lock Management

Coordinates access to shared resources through a sophisticated locking system to prevent conflicts.



Client Implementation

Server Connection

The client establishes a connection to the server using IP address and port number, creating a secure communication channel.

Command Interface

A user-friendly interface parses input and executes corresponding commands, translating user intentions into protocol operations.

Local File Operations

The client handles local operations (lls, lcd, lpwd, lchmod) without server interaction, improving efficiency for local-only tasks.

File Transfer

Implements put and get operations with sophisticated retry mechanisms to ensure reliable file transfers even under unstable conditions.

Core Features

Directory Navigation

Users can seamlessly navigate both local and remote directory structures with intuitive commands like cd, pwd, lcd, and lpwd.

File Management

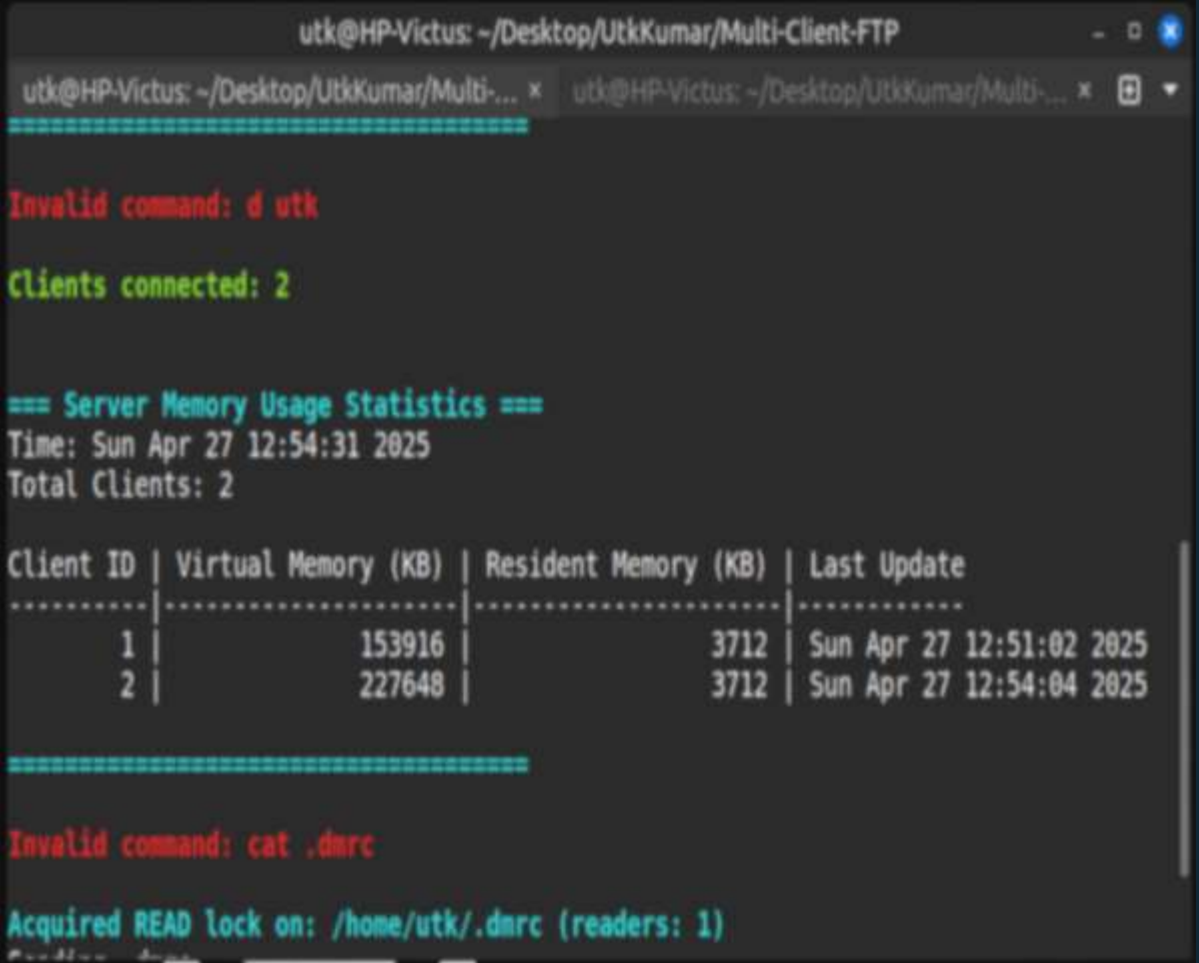
Comprehensive file listing with format options and permission management provide complete control over file resources.

File Transfer

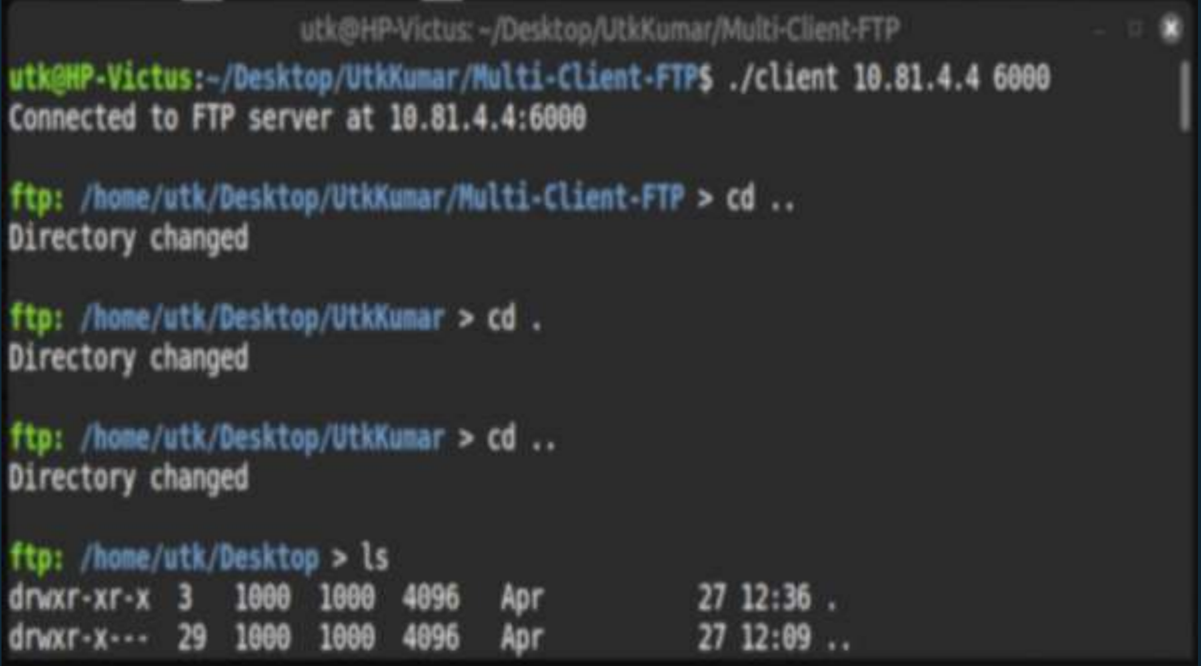
Robust upload and download capabilities with error checking and recovery mechanisms ensure data integrity during transfers.

Multi-Client Support

The server handles multiple simultaneous client connections, each in its own thread, allowing concurrent operations without interference.


utk@HP-Victus: ~/Desktop/UtkKumar/Multi-Client-FTP
utk@HP-Victus: ~/Desktop/UtkKumar/Multi-...
utk@HP-Victus: ~/Desktop/UtkKumar/Multi-...

```
========================================

Invalid command: d utk

Clients connected: 2


=== Server Memory Usage Statistics ===
Time: Sun Apr 27 12:54:31 2025
Total Clients: 2

Client ID | Virtual Memory (KB) | Resident Memory (KB) | Last Update
----------|---------------------|----------------------|------------
        1 |              153916 |                 3712 | Sun Apr 27 12:51:02 2025
        2 |              227648 |                 3712 | Sun Apr 27 12:54:04 2025

========================================

Invalid command: cat .dmrc

Acquired READ lock on: /home/utk/.dmrc (readers: 1)
```



```
utk@HP-Victus:~/Desktop/UtkKumar/Multi-Client-FTP$ ./client 10.81.4.4 6000
Connected to FTP server at 10.81.4.4:6000

ftp: /home/utk/Desktop/UtkKumar/Multi-Client-FTP > cd ..
Directory changed

ftp: /home/utk/Desktop/UtkKumar > cd .
Directory changed

ftp: /home/utk/Desktop/UtkKumar > cd ..
Directory changed

ftp: /home/utk/Desktop > ls
drwxr-xr-x  3  1000  1000  4096   Apr          27 12:36 .
drwxr-x---  29 1000  1000  4096   Apr          27 12:09 ..
```

User Experience Features

Colorized Output

Enhanced readability through color-coded terminal output helps users quickly identify different types of information and status messages.

Detailed Error Messages

Clear, informative error messages guide users to understand and resolve issues quickly without requiring technical expertise.

Local and Remote Commands

Intuitive command prefixing distinguishes between local and remote operations, providing a seamless experience across environments.

Help System

Comprehensive documentation accessible directly from the command line offers immediate assistance for all available commands.

Demo: System in Action

Server Startup

```
$ ./server 8080
FTP Server started at 192.168.1.100:8080 ...
```

Client Connection

```
$ ./client 192.168.1.100 8080
Connected to FTP server at 192.168.1.100:8080
```

Remote Directory Listing

```
ftp: /home/user/server > ls
drwxr-xr-x 4 user group 4.0 KB Jun 10 14:30 .
drwxr-xr-x 8 user group 4.0 KB Jun 10 14:15 ..
-rw-r--r-- 1 user group 10.5 KB Jun 10 14:20 file1.txt
-rw-r--r-- 1 user group 2.3 MB Jun 10 14:25 file2.jpg
```

Local Directory Listing

```
ftp: /home/user/server > lls
drwxr-xr-x 3 user group 4.0 KB Jun 10 14:40 .
drwxr-xr-x 8 user group 4.0 KB Jun 10 14:15 ..
-rw-r--r-- 1 user group 5.2 KB Jun 10 14:35 local_file.txt
```

File Transfer Demos

1

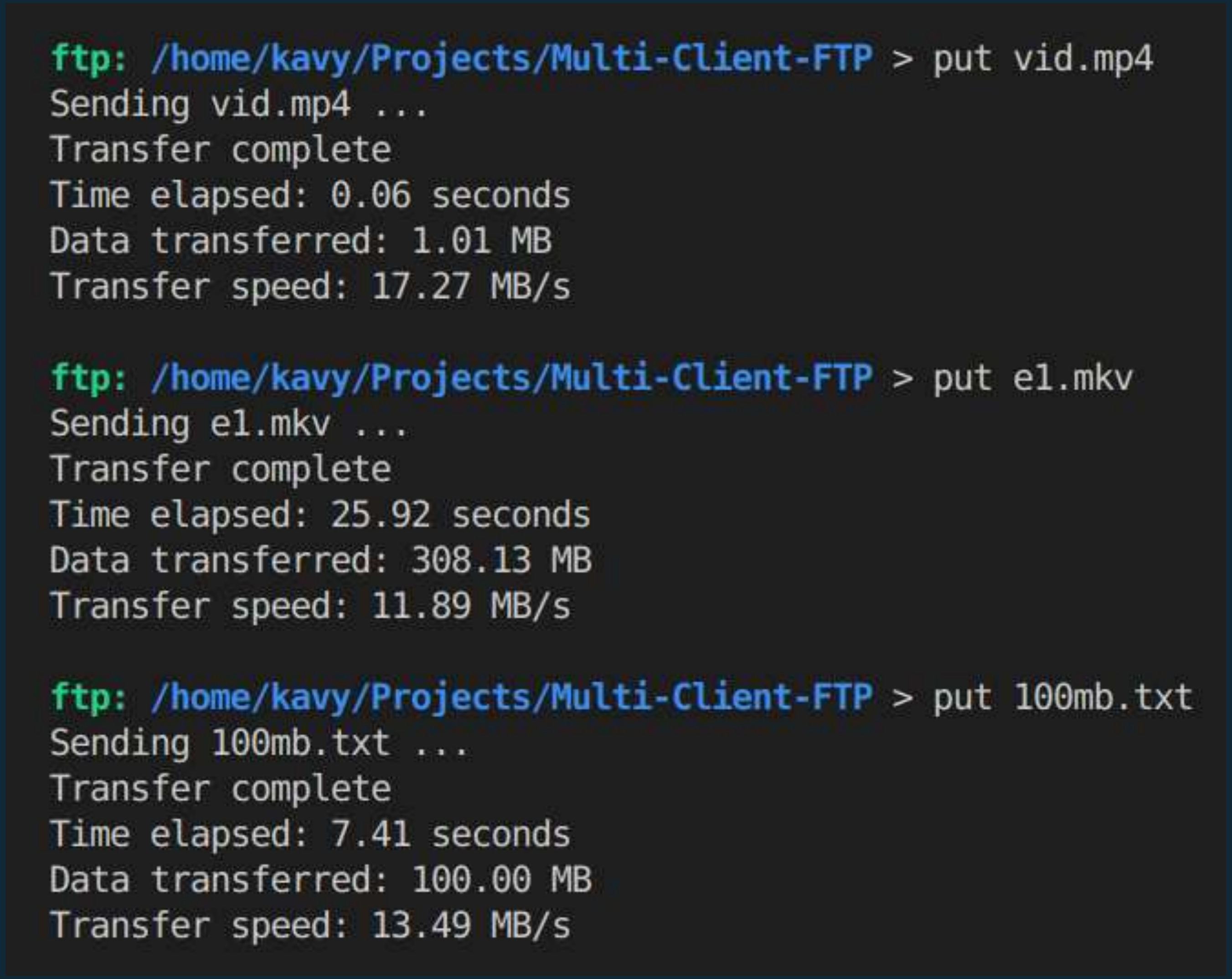
Uploading a File

Use **put file.txt** command to send files to server.

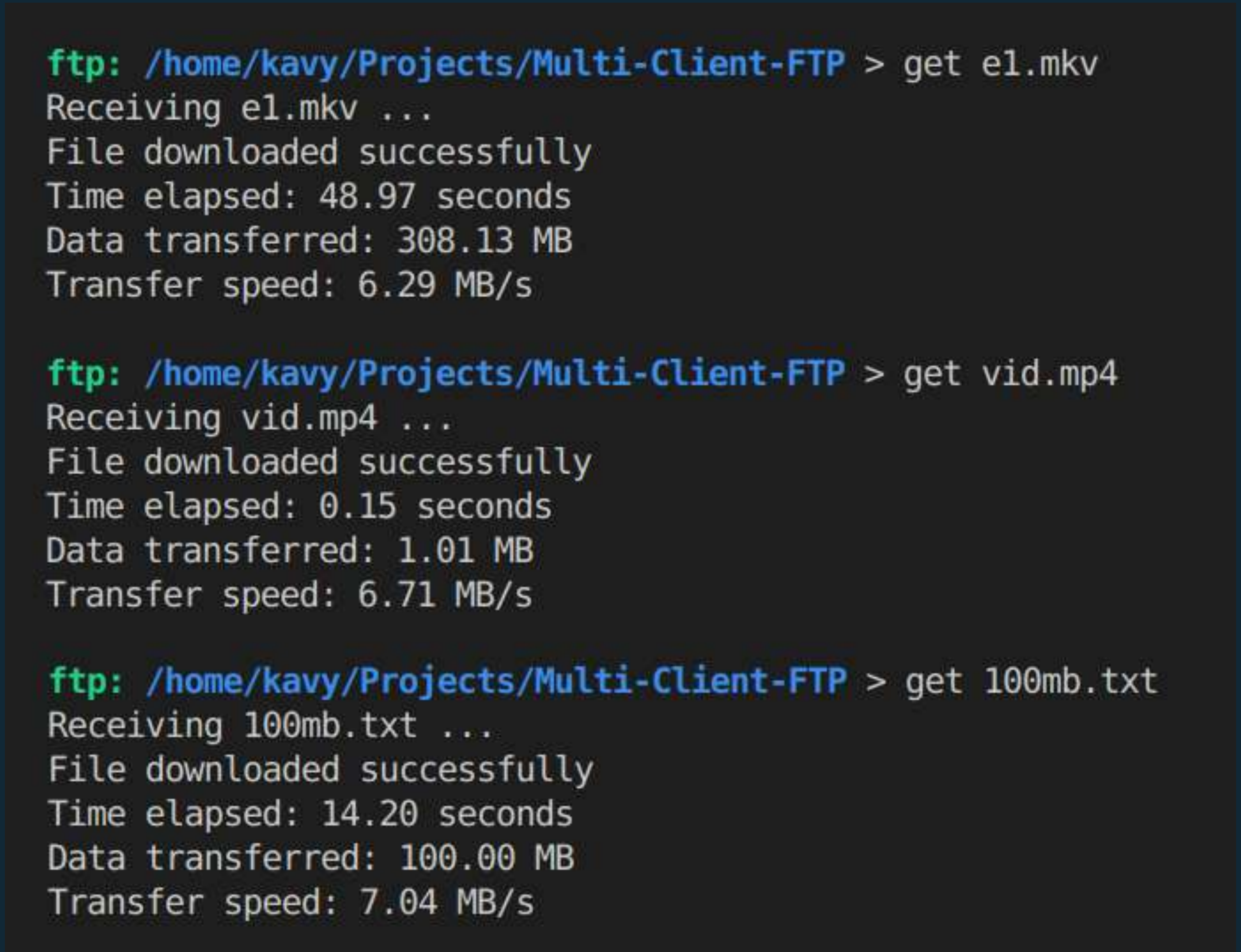
2

Downloading a File

Use **get file.txt** to retrieve files from server.

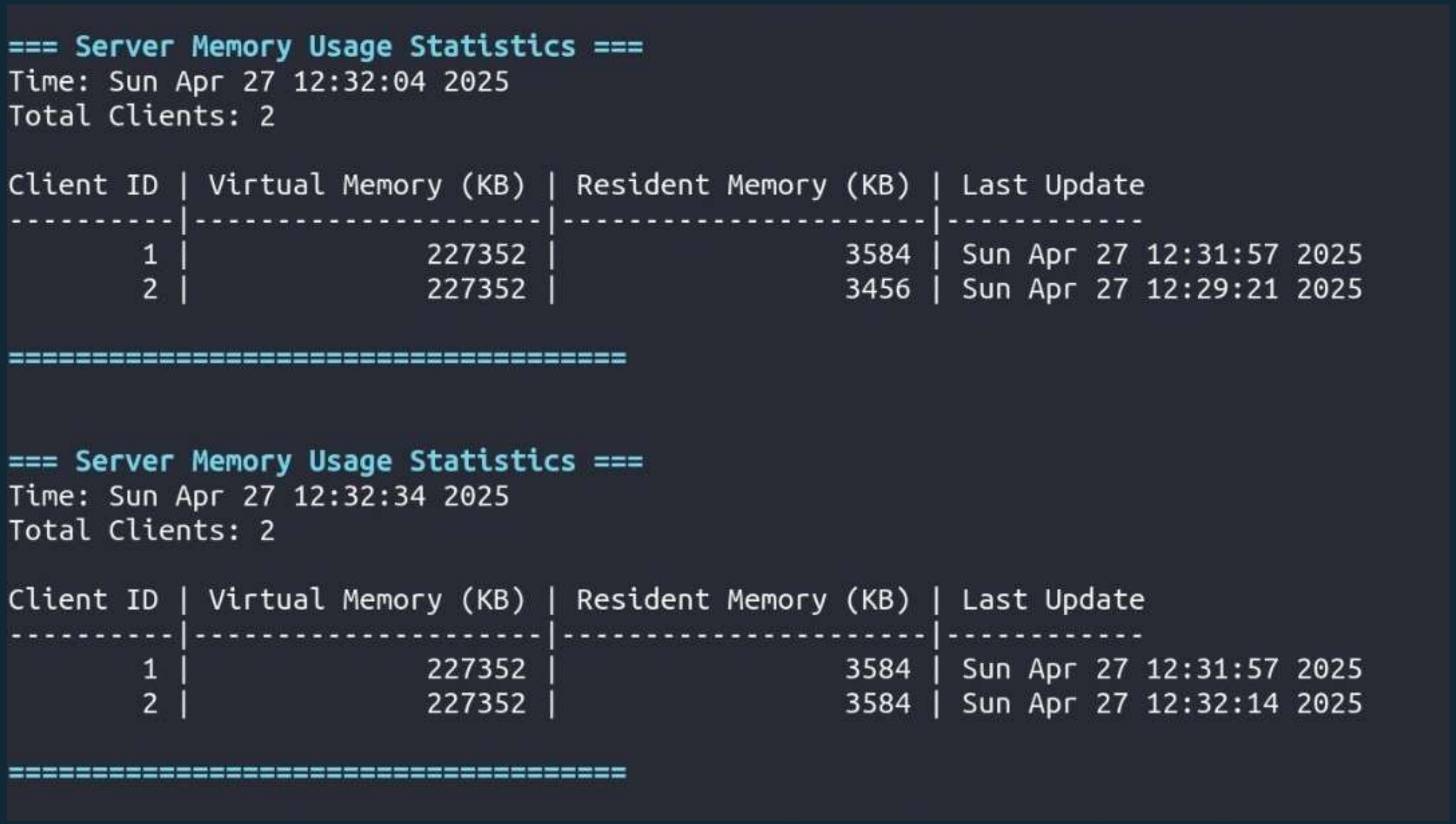


```
ftp: /home/kavy/Projects/Multi-Client-FTP > put vid.mp4
Sending vid.mp4 ...
Transfer complete
Time elapsed: 0.06 seconds
Data transferred: 1.01 MB
Transfer speed: 17.27 MB/s

ftp: /home/kavy/Projects/Multi-Client-FTP > put e1.mkv
Sending e1.mkv ...
Transfer complete
Time elapsed: 25.92 seconds
Data transferred: 308.13 MB
Transfer speed: 11.89 MB/s

ftp: /home/kavy/Projects/Multi-Client-FTP > put 100mb.txt
Sending 100mb.txt ...
Transfer complete
Time elapsed: 7.41 seconds
Data transferred: 100.00 MB
Transfer speed: 13.49 MB/s
```



```
ftp: /home/kavy/Projects/Multi-Client-FTP > get e1.mkv
Receiving e1.mkv ...
File downloaded successfully
Time elapsed: 48.97 seconds
Data transferred: 308.13 MB
Transfer speed: 6.29 MB/s

ftp: /home/kavy/Projects/Multi-Client-FTP > get vid.mp4
Receiving vid.mp4 ...
File downloaded successfully
Time elapsed: 0.15 seconds
Data transferred: 1.01 MB
Transfer speed: 6.71 MB/s

ftp: /home/kavy/Projects/Multi-Client-FTP > get 100mb.txt
Receiving 100mb.txt ...
File downloaded successfully
Time elapsed: 14.20 seconds
Data transferred: 100.00 MB
Transfer speed: 7.04 MB/s
```

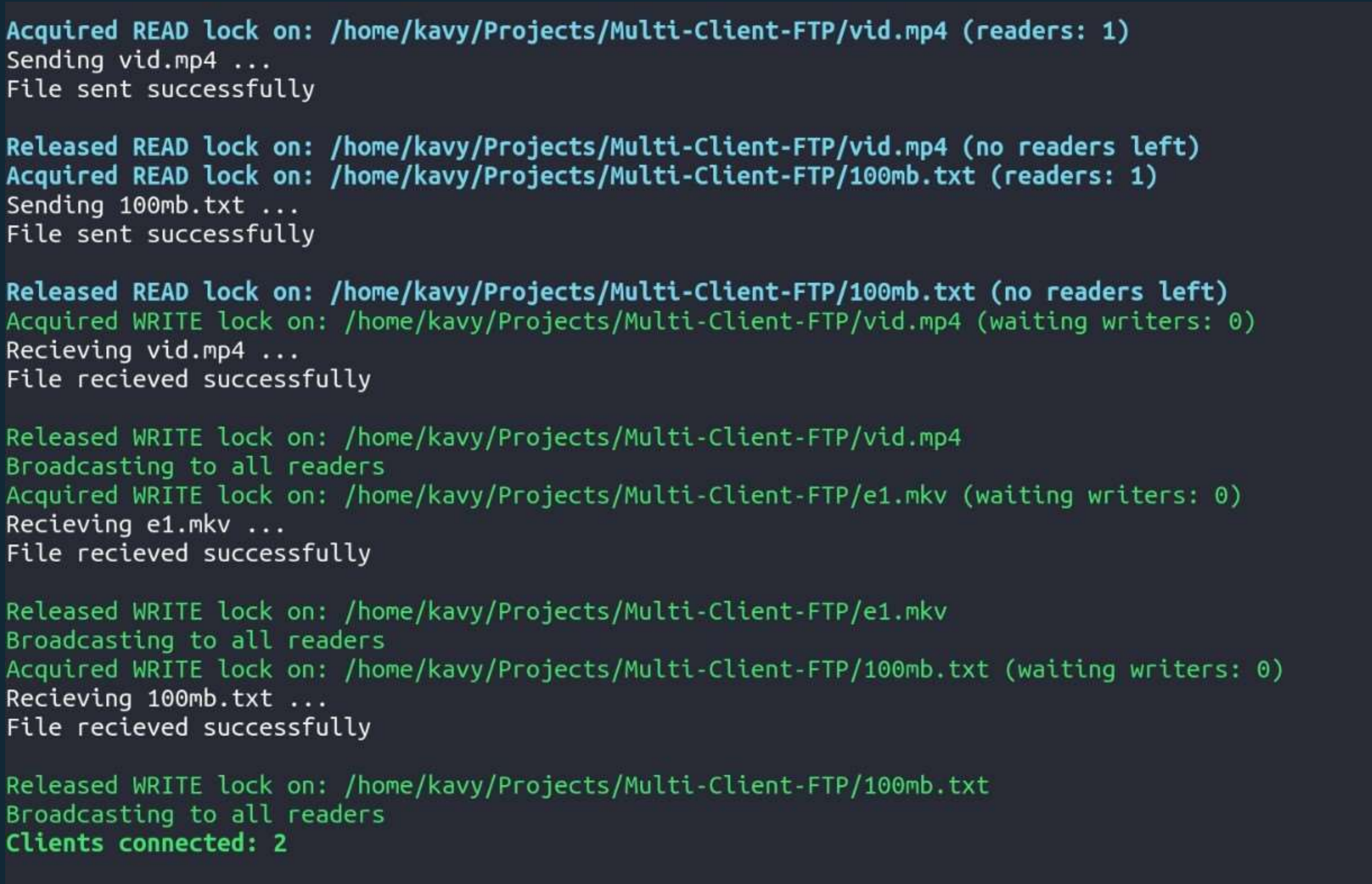
Server Snapshot



```
=== Server Memory Usage Statistics ===
Time: Sun Apr 27 12:32:04 2025
Total Clients: 2

Client ID | Virtual Memory (KB) | Resident Memory (KB) | Last Update
----------|---------------------|----------------------|------------
        1 |              227352 |                 3584 | Sun Apr 27 12:31:57 2025
        2 |              227352 |                 3456 | Sun Apr 27 12:29:21 2025

=====================================


=== Server Memory Usage Statistics ===
Time: Sun Apr 27 12:32:34 2025
Total Clients: 2

Client ID | Virtual Memory (KB) | Resident Memory (KB) | Last Update
----------|---------------------|----------------------|------------
        1 |              227352 |                 3584 | Sun Apr 27 12:31:57 2025
        2 |              227352 |                 3584 | Sun Apr 27 12:32:14 2025

=====================================
```

Memory Usage Stats

Reader Writer Lock



```
Acquired READ lock on: /home/kavy/Projects/Multi-Client-FTP/vid.mp4 (readers: 1)
Sending vid.mp4 ...
File sent successfully

Released READ lock on: /home/kavy/Projects/Multi-Client-FTP/vid.mp4 (no readers left)
Acquired READ lock on: /home/kavy/Projects/Multi-Client-FTP/100mb.txt (readers: 1)
Sending 100mb.txt ...
File sent successfully

Released READ lock on: /home/kavy/Projects/Multi-Client-FTP/100mb.txt (no readers left)
Acquired WRITE lock on: /home/kavy/Projects/Multi-Client-FTP/vid.mp4 (waiting writers: 0)
Recieving vid.mp4 ...
File recieved successfully

Released WRITE lock on: /home/kavy/Projects/Multi-Client-FTP/vid.mp4
Broadcasting to all readers
Acquired WRITE lock on: /home/kavy/Projects/Multi-Client-FTP/e1.mkv (waiting writers: 0)
Recieving e1.mkv ...
File recieved successfully

Released WRITE lock on: /home/kavy/Projects/Multi-Client-FTP/e1.mkv
Broadcasting to all readers
Acquired WRITE lock on: /home/kavy/Projects/Multi-Client-FTP/100mb.txt (waiting writers: 0)
Recieving 100mb.txt ...
File recieved successfully

Released WRITE lock on: /home/kavy/Projects/Multi-Client-FTP/100mb.txt
Broadcasting to all readers
Clients connected: 2
```

Real-World FTP Applications

Local Network Sharing

Simple file sharing in small office or home networks.

Backup Solutions

Reliable file transfers with retry for data safety.

Development Collaboration

Shared asset repository with file locking for teams.

Home Media Server

Media sharing across multiple devices effortlessly.

Educational Use

Assignment submission system for students and teachers.

Planned Future Enhancements

Security Enhancements

- User authentication and authorization
- Encrypted transfers (SSL/TLS)

Performance Improvements

- File caching for faster access
- Parallel file transfers
- Optimized lock management

Feature Extensions

- File append operations
- Directory synchronization
- Web-based management interface

Current System Limitations

No Authentication

Users lack identity verification, reducing security.

Limited Encryption

Data transfers are mostly unencrypted, vulnerable to attacks.

Local Network Focus

Designed for LAN use, limiting internet scalability.

Basic File Operations

Missing advanced operations like append or sync.

Resource Intensive

Performance suffers with many simultaneous clients.

Manual Configuration

Setup requires administrator hands-on work.

Conclusion

- Robust multi-client FTP with sync
- Demonstrated network and concurrency concepts
- Implemented reader-writer locks and deadlock prevention
- Comprehensive and extensible command interface
- Foundation for future security and feature improvements

THANK YOU