Software Requirements Specification (SRS)

for

Distributed Content Discovery & Inspection Utility (DCDIU)

Version 0.1

Prepared by

Group – 02

GROUP MEMBERS
SAGNIK DEY
TILAK KUMAR S
MERSHIKA JAYABAL
RAKSHANA BABU S

Table of Contents

DCDIU Client Server Implementation

1.Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document defines the functional and non-functional requirements for the Distributed Content Discovery & Inspection Utility (DCDIU) implemented using a client–server architecture.

The purpose of this document is to provide a clear and structured description of the system's behavior, interfaces, constraints, and operating environment. It is intended for:

- Developers implementing the client and server components
- System architects designing the client–server interaction
- Instructors and evaluators assessing the project
- Testers responsible for validation and verification

This document serves as a reference throughout the design, implementation, testing, and maintenance phases of the system lifecycle.

1.2 Document Conventions

The document uses the "Times New Roman" font with bold for the heading. The main heading size is 18. The Sub-heading size is 14. The content under the is of 12 points size. All the spacing are normal/default spacing of MS Word.

1.3 Intended Audience and Reading Suggestions

This document is intended for:

- Software Architects
- Software Developers
- System Engineers
- Quality Assurance Engineers
- Project Managers and Evaluators

Readers are expected to have a basic understanding of:

- Client-Server Architecture
- Networking Fundamentals
- Operating Systems
- File Handling in C++
- Multi-threaded Programming

1.4 Project Scope

The Distributed Content Discovery & Inspection Utility (DCDIU) is a command-line based client–server system designed to perform remote directory traversal, content discovery, and file inspection on a server-hosted file system.

In this system:

- The client provides a menu-driven Command Line Interface (CLI) for user interaction
- The server performs all filesystem-related operations, including directory traversal, file scanning, and file inspection
- Communication between client and server is achieved using POSIX socket-based networking

The system enables users to:

- Recursively traverse directories located on a remote server
- Perform deep-content pattern matching across discovered files
- Retrieve and inspect file contents using absolute paths
- Receive descriptive error messages and diagnostic logs

DCDIU operates entirely in **user space** and strictly adheres to **POSIX standards**, ensuring portability across Unix-like operating systems. The modular design allows future enhancements such as multi-client support, parallel processing, advanced pattern matching, and security extensions.

1.5 References

The following authoritative references have been used to support the architectural design, client-server communication, file system operations, error handling, and security concepts defined in the Directory and Content Detection and Inspection Utility (DCDIU) system.

1.POSIX Standard – The Open Group Base Specifications

https://pubs.opengroup.org/onlinepubs/9699919799/

(Official POSIX documentation covering directory handling, file I/O, and system calls like opendir(), readdir(), stat(), open(), read().)

2. Linux Manual Pages (man7.org)

https://man7.org/linux/man-pages/

(Authoritative Linux reference for system calls like open(), read(), stat(), and socket APIs.)

3. C++ Reference – File and String Handling

https://en.cppreference.com/w/cpp/filesystem

(Official C++ reference for file systems and string operations like std::string::find().)

4. Beej's Guide to Network Programming

https://beej.us/guide/bgnet/

(Clear explanation of TCP socket programming used in your client-server communication.)

5. Linux System Programming – Michael Kerrisk

https://man7.org/tlpi/

(Advanced reference for low-level Linux programming, including file descriptors and system calls.)

2. Overall Description

2.1 Product Perspective

The Distributed Content Discovery & Inspection Utility (DCDIU) is a new, standalone client server system.

The system is logically divided into two major components:

- **Client Component**
    - Runs on the user's machine
    - Provides a menu-driven Command Line Interface (CLI)
    - Sends user requests to the server over a TCP connection
    - Displays responses received from the server
- **Server Component**
    - Runs on a remote machine
    - Hosts the target filesystem
    - Performs directory traversal, content scanning, and file inspection
    - Handles logging and exception management
    - Communicates with the client using POSIX sockets

All filesystem operations are strictly executed on the server side, ensuring true remote directory traversal.

2.2 System Architecture

The system follows a **two-tier client–server architecture**:

User → Client (CLI) → Server → File System ← Response ←

Architectural Characteristics:

- Communication protocol: **TCP**
- Data exchange: **Text-based command protocol**
- Execution environment: **User space**
- Operating system compatibility: **POSIX-compliant systems**

The client and server operate as **separate processes**, potentially on different machines, connected via a network.

2.3 Product Functions

At a high level, the DCDIU system provides the following functions:

- Accept user commands through a CLI on the client side
- Establish and maintain a TCP connection with the server
- Perform recursive directory traversal on the server filesystem
- Conduct deep-content discovery by scanning file contents for user-defined patterns
- Allow file inspection by retrieving and streaming file contents to the client
- Handle errors gracefully with descriptive system messages
- Generate diagnostic logs with standardized severity levels

2.4 User Characteristics

The intended users of the system include:

- Students and developers learning system programming
- System administrators auditing files on remote machines
- Users familiar with basic command-line operations

User Assumptions:

- Users understand basic CLI usage
- Users can provide valid directory paths and search patterns
- Users are aware that paths correspond to the **server's filesystem**, not the client's

No advanced technical knowledge of networking or POSIX APIs is required to operate the system.

2.5 Operating Environment

The DCDIU system operates under the following environment:

- **Client Operating System:** Any Unix-like, POSIX-compliant OS
- **Server Operating System:** Linux or Unix-based system
- **Programming Language:** C++
- **Networking:** POSIX TCP sockets
- **Execution Mode:** Terminal-based execution
- **Privileges:** Standard user privileges (root not required)

2.6 Design and Implementation Constraints

The system is subject to the following constraints:

- Only **POSIX APIs** may be used for filesystem and networking operations
- No graphical user interface (GUI) is allowed
- Communication must be implemented using low-level socket APIs (no HTTP/REST)
- The server must be capable of handling malformed or invalid client requests
- The system must run entirely in **user space**

2.7 Assumptions and Dependencies

Assumptions:

- The server is running and reachable before the client starts
- Network connectivity is stable
- The server filesystem is accessible with sufficient permissions

Dependencies:

- POSIX-compliant C++ compiler
- Standard C and C++ libraries
- TCP/IP networking support on the host OS

2.8 Future Enhancements

The modular client–server architecture allows future extensions such as:

• End-to-end encrypted communication channels (SSL/TLS)
• Advanced pattern matching using regular expressions
• Web-based client interface
• Distributed server clusters for load balancing

• Cloud-based deployment and monitoring
• AI-based content analysis and threat detection

3. System Features (Functional Requirements)

This section describes the **major system features** provided by the DCDIU system. Each feature is presented with a clear description, inputs, processing behavior, and outputs. All features are implemented using a **client–server architecture**, where the client initiates requests and the server performs processing.

3.1 Client–Server Communication Feature

Description

This feature enables communication between the client and the server using **TCP sockets**. It forms the foundation of all other system features.

Functional Requirements

- The server shall listen for incoming client connections.
- The server shall prompt the client for authentication credentials (username and password).
- The client shall provide valid login credentials to the server.
- The server shall authenticate the client using secure hashed password verification.
- Only authenticated clients shall be allowed to execute commands.
- The client shall send text-based commands to the server after successful authentication.
- The server shall process commands and send responses back to the client.
- The connection shall remain active until the client issues an exit command..
- The connection shall remain active until the client issues an exit command.

Inputs

- Client command strings (e.g., TRAVERSE, SEARCH, INSPECT, EXIT)

Processing

- Socket creation and connection establishment
- Command transmission and reception
- Request parsing on the server side

Outputs

- Acknowledgement messages
- Operation results
- Error messages

3.2 Remote Directory Traversal Feature

Description

This feature allows the client to request **recursive traversal of a directory located on the server's filesystem.**

Functional Requirements

- The client shall send a directory path to the server.
- The server shall validate the directory path.
- The server shall recursively traverse the directory and its subdirectories.
- The server shall collect all regular file paths discovered.
- The server shall return the total number of files discovered to the client.

Inputs

- Directory path (string) relative to the server's filesystem

Processing

- Directory access using POSIX APIs
- Recursive traversal logic
- File path collection

Outputs

- File discovery count
- Status or error messages

3.3 Content Discovery and Pattern Matching Feature

Description

This feature allows the client to search for specific strings or patterns within files discovered during traversal.

Functional Requirements

- The client shall provide a search pattern.
- The server shall scan all previously discovered files.
- The server shall read file contents using POSIX file APIs.
- The server shall identify files containing the specified pattern.
- The server shall return a list of matching file paths to the client.

Inputs

- Directory Path(string)
- Search pattern (string)

Processing

- Sequential file reading
- Pattern matching within file contents
- Match result aggregation

Outputs

- List of files containing the pattern
- Informational log messages

3.4 File Inspection Feature

Description

This feature allows the client to inspect the **full contents of a specific file located on the server**.

Functional Requirements

- The client shall provide an absolute file path.
- The server shall validate file existence and permissions.
- The server shall read the file in chunks.
- The server shall stream the file contents to the client.
- The client shall display the file contents to the user.

Inputs

- Absolute file path (string)

Processing

- File opening and reading
- Chunk-based data transmission
- End-of-message detection

Outputs

- Full file content
- Error messages if inspection fails

3.5 Exception Handling Feature

Description

This feature ensures that system failures are handled gracefully without crashing either the client or the server.

Functional Requirements

- The server shall detect filesystem access errors.
- The server shall generate descriptive error messages.
- The system shall continue execution after recoverable errors.
- Fatal errors shall terminate execution safely.

Inputs

- Invalid paths
- Permission-denied files
- Failed system calls

Processing

- Error detection using errno
- Error classification (recoverable vs fatal)

Outputs

- Warning or fatal messages sent to the client
- Diagnostic logs

3.6 Diagnostic Logging Feature

Description

This feature records system activity and errors using a standardized logging mechanism.

Functional Requirements

- The system shall support four log levels: FATAL, INFO, WARNING, DEBUG.
- Logs shall include timestamps.
- Logging shall be centralized on the server.
- Fatal logs shall terminate the server process.

Inputs

- System events
- Error conditions

Processing

- Log message formatting
- Severity-based handling

Outputs

- Console log messages
- Termination for fatal conditions

3.7 Command Termination Feature

Description

This feature allows the client to terminate the session cleanly.

Functional Requirements

- The client shall send an exit command to the server.
- The server shall close the active client connection.
- System resources shall be released properly.

Inputs

- Exit command

Processing

- Socket closure
- Resource cleanup

Outputs

- Confirmation of session termination

4. External Interface Requirements

This section describes how the **DCDIU client–server system interacts with external entities**, including users, hardware, software, and communication interfaces.

4.1 User Interfaces

Description

The DCDIU system provides a **command-line based user interface (CLI)** on the **client side**. No graphical user interface is used.

Interface Characteristics

- Menu-driven text interface
- Keyboard-based input
- Console-based output

User Interface Elements

- Menu options:
- 1. Traverse Directory
- 2. Search Content
- 3. Inspect File
- 4. Exit
- Input prompts for:
    - Directory paths
    - Search patterns
    - Absolute file paths

Output Display

- Operation results
- File lists
- File contents
- Error and status messages

Constraints

- The interface must remain responsive
- Invalid inputs must not crash the client
- All outputs must be human-readable

4.2 Hardware Interfaces

Description

The DCDIU system does not require any specialized hardware.

Minimum Hardware Requirements

- Standard x86 or ARM-based processor
- Keyboard for user input
- Network interface card (NIC) for client–server communication

Notes

- Client and server may run on the same machine or different machines
- No hardware acceleration or special peripherals are required

4.3 Software Interfaces

Operating System Interface

- POSIX-compliant operating systems (Linux/Unix)
- Uses POSIX system calls for:

  File handling

  Directory traversal

  Process and socket management

Programming Language Interface

- Implemented in **C++**
- Uses standard C++ libraries
- Uses POSIX C libraries

Filesystem Interface

- Server directly accesses the local filesystem
- Client has no direct filesystem access
- File operations performed using:
  - open()
  - read()
  - opendir()
  - readdir()
  - stat()

4.4 Communication Interfaces

Description

Communication between client and server is achieved using **TCP sockets**, ensuring reliable data transfer.

Protocol Details

- Transport Protocol: TCP
- Communication Model: Request–Response

- Data Format: Plain text commands and responses
- Message Termination: End-of-message marker (<<END>>)

Supported Commands

Command	Description
TRAVERSE <path>	Request recursive directory traversal
SEARCH <pattern>	Search for pattern in discovered files
INSPECT <file>	Retrieve file contents
EXIT	Terminate session

4.5 Network Interface Requirements

- IPv4 networking support
- Server listens on a fixed port
- Client connects using server IP and port
- Connection maintained until exit

4.6 Error and Message Interfaces

Error Handling Interface

- Descriptive error messages returned to the client
- POSIX errno used for error context
- Errors categorized as:
    - Warning
    - Fatal

Logging Interface

- Logs generated on server side
- Four severity levels supported:
    - FATAL
    - INFO
    - WARNING
    - DEBUG

4.7 Interface Constraints

- No GUI or web interface allowed
- No third-party networking libraries permitted
- Communication must be synchronous

5. Non-Functional Requirements

These requirements focus on how the system performs rather than what it does. This section defines the quality attributes and constraints of the DCDIU client–server system.

5.1 Performance Requirements .

- The system shall respond to client requests within an acceptable time under normal load conditions.
- Directory traversal and content scanning operations shall be performed sequentially to ensure correctness.
- File inspection shall stream data in chunks to avoid excessive memory usage.
- Network communication shall use TCP to ensure reliable data transfer.

5.2 Reliability Requirements

- The server shall continue running even if a client disconnects unexpectedly.
- The system shall handle recoverable errors without crashing.
- Invalid inputs from the client shall not cause undefined behavior.
- The system shall ensure clean shutdown of sockets and file descriptors.

5.3 Availability Requirements

- The server shall remain available as long as it is running.
- The client shall be able to reconnect after a previous session ends.
- Temporary failures (e.g., permission issues) shall not stop the server.

5.4 Security Requirements

- The client shall not directly access the server filesystem.
- All filesystem operations shall be executed only on the server side.
- The server shall validate all client commands before processing.
- The system shall not require root privileges to operate.

Future enhancement: Authentication and encrypted communication may be added.

5.5 Portability Requirements

- The system shall be portable across all POSIX-compliant operating systems.
- No OS-specific or platform-dependent libraries shall be used.
- The system shall compile using standard POSIX-compatible C++ compilers.

5.6 Maintainability Requirements

- The system shall follow a modular design.
- Each module shall have a single, well-defined responsibility.
- Code shall be readable, well-commented, and easy to extend.
- New features (e.g., regex scanning, multithreading) can be added without major redesign.

5.7 Scalability Requirements

- The architecture shall support future multi-client extensions.
- The server design shall allow integration of concurrency mechanisms.
- File processing modules shall be reusable across different contexts.

5.8 Usability Requirements

- The client interface shall be menu-driven and intuitive.
- Error messages shall be descriptive and user-friendly.
- The system shall guide users through valid input formats.

5.9 Compliance Requirements

- The system shall comply with POSIX standards.
- The system shall adhere to IEEE SRS documentation guidelines.
- The system shall run entirely in user space.

5.10 Safety Requirements

- The system shall not modify or delete any files.
- The system shall operate in read-only mode for all file operations.
- Unexpected termination shall not corrupt system resources.

6. Other Requirements

This section describes **additional requirements**, **future enhancements**, and **assumptions** related to the DCDIU client–server system that do not fit strictly into functional or non-functional categories.

6.1 Error Handling Requirements

- The server shall detect and handle all system call failures.
- Error messages shall include meaningful descriptions using POSIX error codes.
- Recoverable errors shall not terminate the server.
- Fatal errors shall result in a controlled shutdown.
- The client shall display server error messages clearly to the user.

6.2 Logging and Monitoring Requirements

- All critical operations shall be logged on the server.
- The logging system shall support four severity levels:
    - FATAL
    - INFO
    - WARNING
    - DEBUG
- Log messages shall include timestamps.
- Logging shall assist in debugging, auditing, and system monitoring.

6.3 Data Handling Requirements

- The system shall operate strictly in **read-only mode**.
- No file creation, modification, or deletion shall be performed.
- All data transmitted between client and server shall be textual.
- Large file data shall be transferred in chunks.

6.4 Network and Protocol Requirements

- Communication shall use a persistent TCP connection.
- The system shall implement an application-level protocol.
- Each server response shall terminate with an end marker (<<END>>).
- The client shall read responses until the end marker is detected.

6.5 Assumptions

- The server filesystem structure remains stable during execution.
- Users provide correct server-side paths.
- Network connectivity is reliable during operations.
- The client and server are trusted environments.

6.6 Dependencies

- POSIX-compliant operating system
- C++ compiler supporting standard libraries
- TCP/IP networking stack
- Standard C and C++ runtime libraries

6.7 Future Enhancements

The system architecture allows for the following future improvements:

- Multi-client handling using threading or multiprocessing
- Secure authentication mechanisms
- Encrypted communication using TLS
- Advanced pattern matching using regular expressions
- Distributed server clusters for load balancing
- Graphical user interface (GUI) client

6.8 Limitations

- The current system supports a single client at a time.
- No authentication or authorization is implemented.
- Communication is unencrypted.
- Performance depends on network and filesystem size.

6.9 Legal and Regulatory Requirements

- The system does not process personal or sensitive data.
- No licensing restrictions apply to the implementation.
- The system complies with academic usage guidelines.

6.10 Document Completion

This SRS document provides a complete and structured specification for the Distributed Content Discovery & Inspection Utility (DCDIU) implemented using a client–server

architecture. It serves as a definitive reference for system development, testing, evaluation, and future enhancement.

Appendix A: Glossary:

DCDIU – Distributed Content Delivery And Inspection Utility

CLI – Command Line Interface

POSIX – Portable Operating System Interface

TCP – Transmission Control Protocol

DFS – Distributed File System

API – Application Programming Interface

UT – Unit Testing

IT – Integration Testing