Karol Korszun

Operating Systems Project

Introduction

This project implements Phase 1 of a distributed system for file processing, establishing the foundational framework with core components and their basic interactions. The system follows a client-server architecture with specialized worker processes, where the client interface manages user interactions, the central server coordinates operations, and worker processes handle specific types of file processing tasks.

Core Components

1. Client (Fleck)

   1. Command-line interface for user interaction
   2. Configuration management
   3. Basic shell implementation with commands:
      - 3.1. CONNECT: Initiates server connection
      - 3.2. LIST TEXT/MEDIA: File listing functionality
      - 3.3. DISTORTED: File processing request
      - 3.4. CHECK STATUS: Operation monitoring
      - 3.5. LOGOUT: Connection termination

2. Central Server (Gotham)

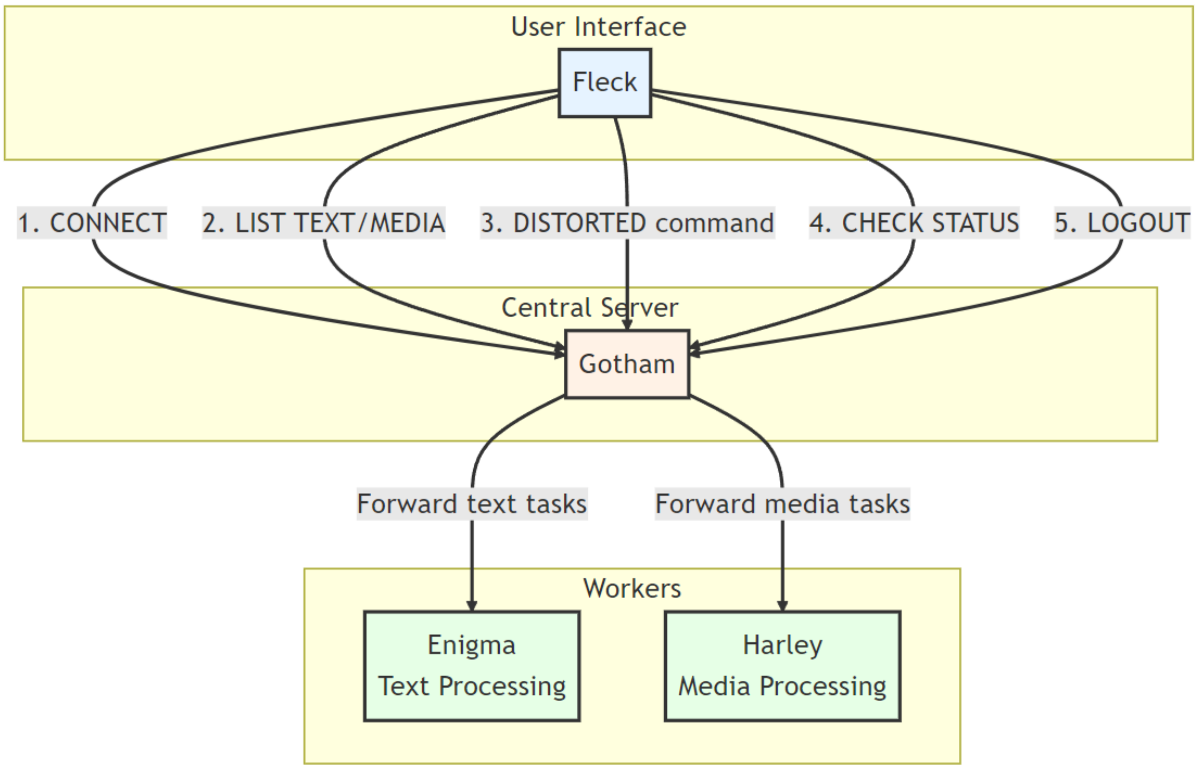
   1. Main coordination server
   2. Handles component registration
   3. Configuration management for network settings

3. Worker Processes

   1. **Enigma**: Specialized for text file processing
   2. **Harley**: Specialized for media file processing

3. Both implement configuration loading and directory verification

The structure of the program may be seen on the diagram below.



Implementation Details

File Operations

The system implements robust file handling through careful management of file descriptors, particularly in configuration file operations. Each component verifies directory existence before performing any operations. String operations are implemented with bounds checking using strncpy, preventing buffer overflows.

Signal Handling

Signal management is implemented through a custom SIGINT handler, registered within each system component. This ensures graceful handling of interruption signals, preventing abrupt termination during critical operations.

Command Processing (Fleck)

Commands are processed case-insensitively to improve user experience. Each command undergoes input validation before execution, with an error message system providing clear feedback for invalid inputs. The implementation provides a structured approach to command handling.

Configuration Management

Configuration handling is implemented through a systematic line-by-line reading approach. Each component validates its specific parameters during initialization. The system verifies IP addresses, port numbers, and file paths, ensuring all configuration parameters are valid before proceeding with operations. String copying is implemented with memory safety considerations.

Error Management

The error handling system implements standardized error reporting across all components. Input validation occurs at multiple stages of operation, ensuring system stability. When errors occur, the system performs proper resource cleanup and provides descriptive error messages to users. Exit status handling is implemented to maintain system reliability.

Memory Management

Memory handling is implemented with attention to dynamic allocation and deallocation. The system tracks memory usage through cleanups, preventing memory leaks. Buffer overflow prevention is implemented through size checking and bounded operations.

Data Structures

The system uses three main structures to manage component configurations:

FleckConfig

```
  char username[MAX_USERNAME_LENGTH]     // User identification
  char folder_path[MAX_PATH_LENGTH]      // Working directory path
  char gotham_ip[MAX_IP_LENGTH]          // Server IP address
  char gotham_port[MAX_PORT_LENGTH]      // Server port number
```

GothamConfig

```
  char fleck_ip[MAX_IP_LENGTH]           // Client IP address
  char fleck_port[MAX_PORT_LENGTH]       // Client port number
  char worker_ip[MAX_IP_LENGTH]          // Worker IP address
  char worker_port[MAX_PORT_LENGTH]      // Worker port number
```

WorkerConfig

```
  char gotham_ip[MAX_IP_LENGTH]          // Server IP address
  char gotham_port[MAX_PORT_LENGTH]      // Server port number
  char fleck_ip[MAX_IP_LENGTH]           // Client IP address
  char fleck_port[MAX_PORT_LENGTH]       // Client port number
  char save_folder[MAX_PATH_LENGTH]      // Output directory path
  char worker_type[MAX_TYPE_LENGTH]      // Worker specialization type
```

Justification:

- Fixed-size arrays are used for string storage to prevent buffer overflows
- Separate structures for each component type to maintain clear separation of concerns
- Common maximum lengths defined as constants for consistency
- Structure members ordered to optimize memory alignment

Running The Project

In order to run the project, we may simply run the bash script called run_project.sh after compiling the code using command make or copy the commands from the bash script and manually introduce them in the command line.