WIFI-filesystem

Academic project for the course of Operating Systems

1st Maksym Mykhasiuta
Ukrainian Catholic University
Computer Science
Lviv, Ukraine
maksym.mykhasiuta@ucu.edu.ua

2nd Fedir Zhurba
Ukrainian Catholic University
Computer Science
Lviv, Ukraine
fedir.zhurba@ucu.edu.ua

3rd Bohdan Pelekh
Ukrainian Catholic University
Computer Science
Lviv, Ukraine
bohdan.pelekh@ucu.edu.ua

I. Introduction

This educational project aspires to develop a device designed to facilitate the remote operations with files through wireless connectivity, thereby providing an access to a remote file system. The project aims to establish a user interface that enables seamless interaction with files situated in remote storage, leveraging the capabilities of WiFi technology to simulate conventional processes associated with local file systems.

II. Components

A. Portable device

We use one ESP32-S3 board as a flash drive emulating device. The device connects to the computer and provides access to the remote file system as a pen drive. Using a WiFi connection, the board accesses the server as a client, requests information about the status of the file system, its fullness, and can perform read-write operations.

B. Remote servers

The servers facilitate communication with remote clients over a TCP/IP socket. Clients can create requests like initialization requests, write requests, and get requests to interact with the FAT32 image file.

The communication protocol is established using sockets, by binding the servers to listen for incoming connections on a specified port. Upon connection, the servers processes the client's requests.

Servers abstract essential file system operations, providing a platform to interact with file data remotely, emulating it as plugged-in USB flash device. The overall logic focuses on creating a seamless bridge between clients and remote FAT32 image files, providing a robust interface for interaction.

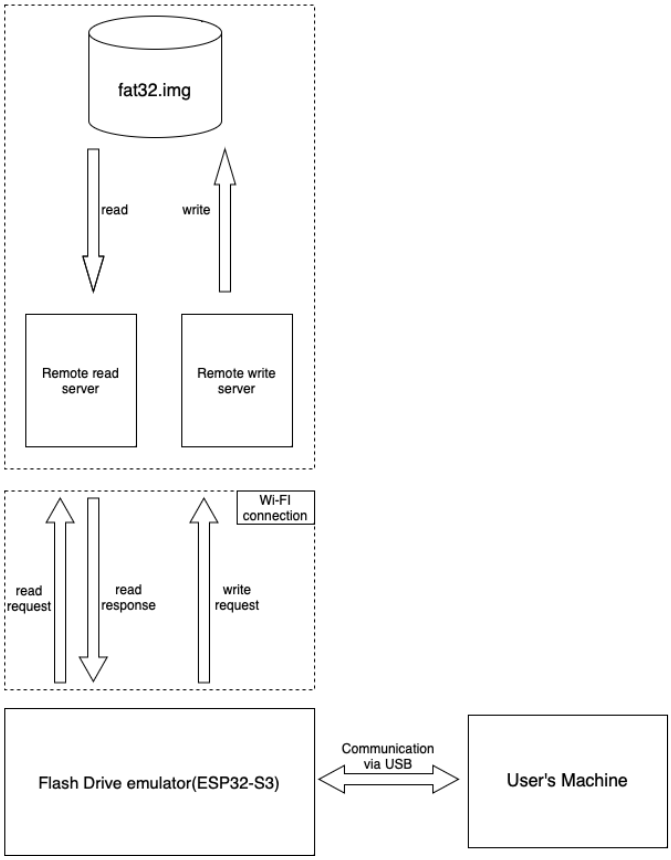

Figure 1. System data flow.

III. Data Flow

The comprehensive data flow within the system is encapsulated by four principal entities: the user's computing device, the portable flash storage device, the remote servers, and the underlying file system. The interaction initiates as the user's computer engages with the portable flash device, establishing a communication channel for data exchange. Through a series of read-and-write operations, the user's computer facilitates the transfer of data to and from the remote read and write servers. The servers acts like a central hubs, receiving specialised user requests and data payloads. Having been provided access to the remote FAT32 image, the servers tries to execute data operations and returns a response to the portable flash device (success or failure).

IV. Technical details

A. Portable device

Upon connection to a computer, the ESP32 is configured to function as a USB Mass Storage device. This allows users to interact with the ESP32 as an external drive, facilitating seamless data transfer between the ESP32 and the connected computer.

Upon connecting to the remote server, the ESP32 dynamically initializes its USB Mass Storage drive based on information received from the server. This dynamic initialization includes crucial details such as the number of sectors, clusters, and available files.

Continuous monitoring of the connection to the remote server is a key feature. In the event of a connection loss, the system attempts to re-establish the connection, ensuring robustness and reliability in varying network conditions.

Events related to the USB Mass Storage interface, such as start/stop events, are handled appropriately. For instance, the system responds to USB connection plug and unplug events.

B. Read server

The server initializes a TCP socket, binds it to a specified port, and starts listening for incoming connections. Upon accepting a client connection, it sets up TCP keepalive and reuse address options for the accepted socket.

The server processes an init structure, which is sent by ESP at the beginning of the workflow. This is required to correctly initialise the filesystem, according to the provided number of sectors and the size of one sector.

Filesystem is emulated using library tinyusb, which provides a possibility to create a FAT32 image using the information in the init request.

Following read requests specify which data to read. According to those requests server provides a buffer with requested data as a response.

C. Write server

When this server receives a write request, it opens the specified filesystem and writes the data from the receive buffer to the file at the appropriate position.

This server is tailored for a specific purpose – acting as an intermediary for filesystem interactions with a remote client.

The main difference from the write server is the type of received and sent requests. For write requests server does not send a response back. Also, write request must contain a buffer with the information we want to write. Having this two-server system allows to avoid some transition errors and provides more robust interface for error handling and USB-server interactions.

D. Finite state machine

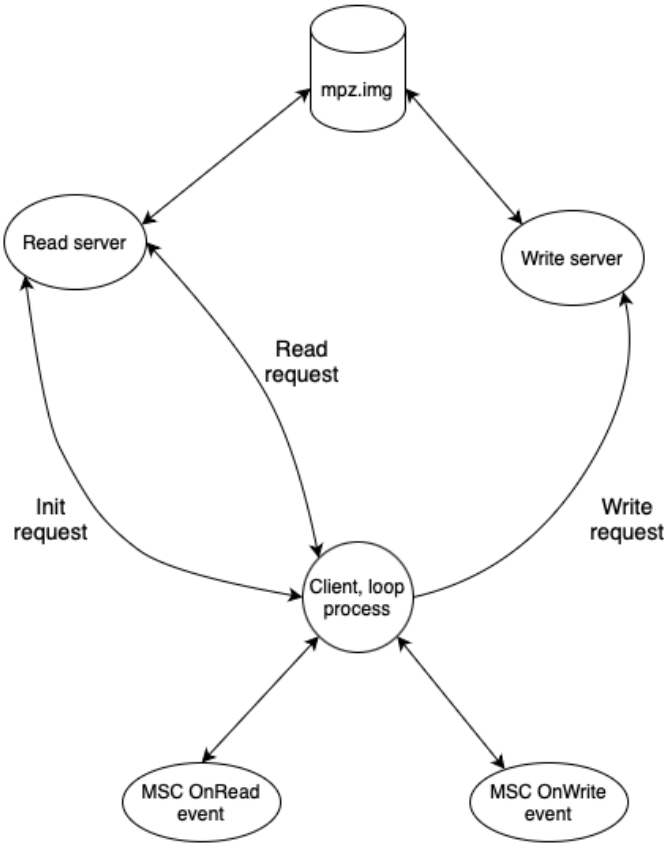

Figure 2. System FSM.

References

[1] Documentation of the FAT Filesystem. http://elm-chan.org/docs/fat_e.html#fsinfo, May 24, 2009.

[2] CodeName33, "Моя попытка сделать Wi-Fi-флешку и что из этого получилось (а что нет)" https://habr.com/en/articles/685768/, September 5, 2022.

[3] Espressif Systems (Shanghai) Co., "ESP32-S3 Series Datasheet". https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf, November 24, 2023.

[4] Ha Thach Co., "TinyUSB". https://www.tinyusb.org/, 2021.