NetworkDevicesMonitor

Gabriel Dancă

December 2023

1 Introduction

This report was designed to demonstrate the implementation of a multithreaded and concurrent server for the "NetworkDevicesMonitor" project. The purpose of the project is to have an application that can monitor the network infrastructure, regardless of where each actor is, with the user being able to monitor them. The user will have a customizable dashboard available, with numerous statistics and metrics, alongside filters to retrieve logs from the central server.

2 Applied Technologies

2.1 TCP - Transmission Control Protocol

The TCP protocol was chosen in this project to ensure reliability in the transmission of data between clients and the server, because the information transmitted is vital for the user's dashboard.
TCP is a connection-oriented transport protocol, ensuring lossless data delivery. It aims to provide maximum quality of service, integrating mechanisms for connection establishment and release, as well as controlling the flow of data.

2.2 SQLite

This technology was choosen because the user requests some information about the logs from the central server, so the server must be able to filter quickly the relevant data. SQLite is a lightweight, serverless, and self-contained relational database management system (RDBMS). It is known for its simplicity, efficiency, and ease of integration into various applications.

2.3 Systemd-journal-remote

Is a component of the systemd system and service manager on Linux systems. Specifically, it is a part of the journal subsystem, which manages system and service log data.
In the project, The systemd-journal-remote tool is used for remote logging from the agents in the central server.

3 Application Structure

3.1 Client Structure

• The client invokes the ‘socket()‘ function, saves the returned descriptor in a variable, and connects to the server using the ‘connect()‘ primitive.
• After successfully connecting to the server, the client’s operating system, which is a distribution of Linux, sends all the logs of the system.
• The user receives relevant statistics and metrics about his system from the server, such as authentication attempts or security breaches. He can also retrieve them manually from the given dashboard.

3.2 Server Structure

• The server invokes the ‘socket()‘ function, saves the returned descriptor in a variable, populates the data structure used by the server, and associates the socket with the corresponding information by calling the ‘bind()‘ function.
• It listens for incoming client connections using the ‘listen()‘ primitive.
• In an infinite loop, it accepts a connection with a client through the ‘accept()‘ function, simultaneously saving the returned descriptor.
• It creates a thread with the descriptor returned by ‘accept()‘ through a call to the ‘pthread create()‘ function.
• It stores the logs from the user’s actors in an SQLite database and sends the statistics and metrics back to the user.

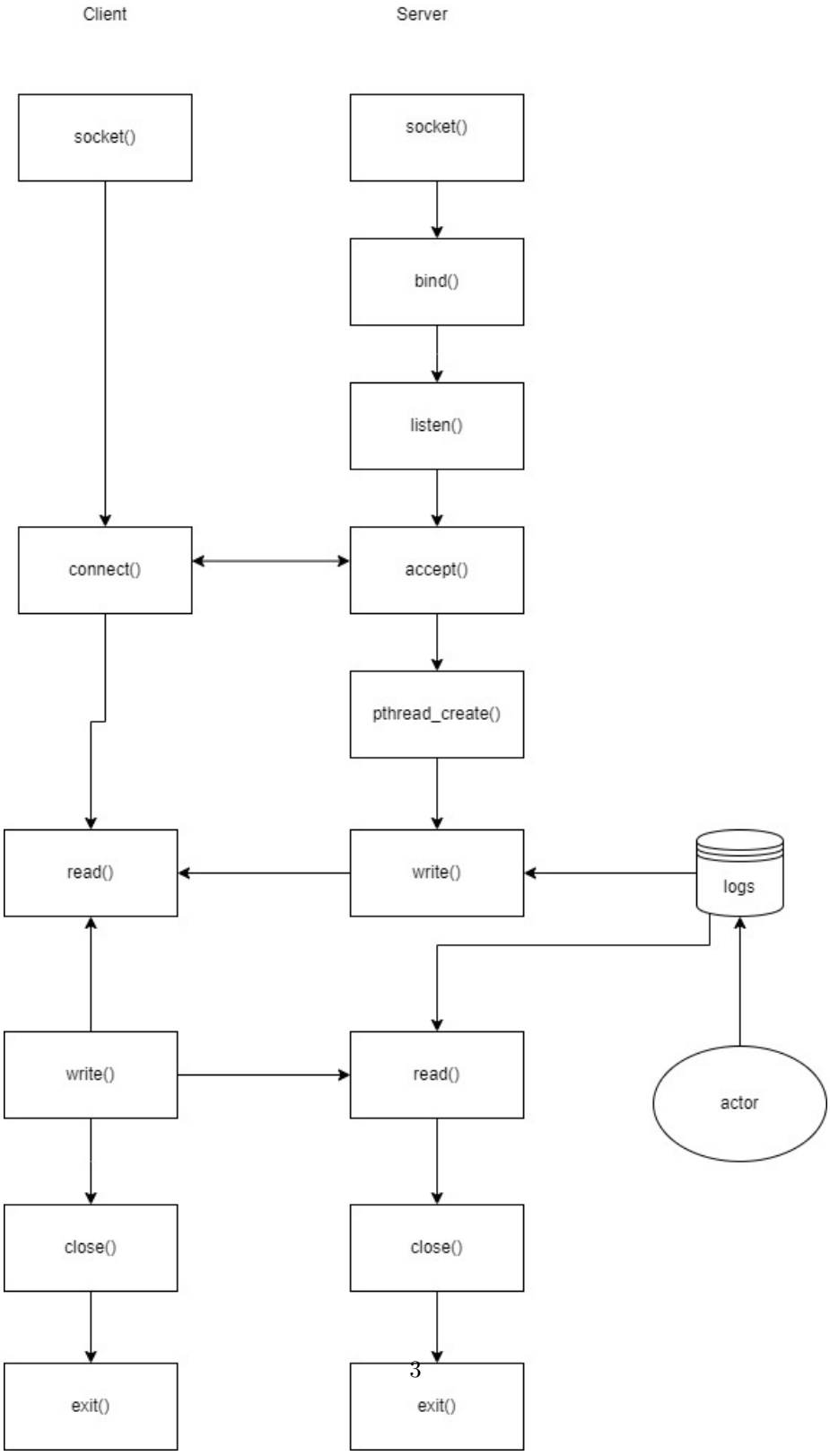

Figure 1: Diagram of the application

4 Implementation Aspects

For the project to even function, each user need to follow some steps to link the device(s) to the central server. This steps are:

4.1 Client upload

4.1.1 Install systemd-journal-remote

• sudo apt-get install systemd-journal-remote

4.1.2 Edit /etc/systemd/journal-upload.conf

•[Upload]
URL=http://10.0.0.1:19532

The user must modify the IP adress with the one given in the dashboard

4.1.3 To make sure journal-upload auto start on boot

• sudo systemctl enable systemd-journal-upload.service

4.1.4 Restart journal-upload after configuration

• sudo systemctl restart systemd-journal-upload.service

4.2 Client connection to server

In the client implementation, the `socket()` primitive will be used as follows:

```
if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("Error at socket().\n");
    return errno;
}
```

After the `socket()` primitive is called, the server data structure is populated with the provided IP address and port obtained during the client's execution launch. Subsequently, the client connects to the server using the `connect()` function:

```
server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr(argv[1]);
server.sin_port = htons(port);

if (connect(sd, (struct sockaddr *) &server, sizeof(struct sockaddr)) == -1) {
    perror("Error at connect().\n");
    return errno;
}
```

4.3 Server

In the case of the server, the call of the `socket()` primitive and the population of the server data structure are done in a similar manner. However, it additionally calls two more primitives before establishing the connection with the client - `bind()` and `listen()`:

```
if (bind(sd, (struct sockaddr *) &server, sizeof(struct sockaddr)) == -1) {
    perror("Error at bind().\n");
    return errno;
}

if (listen(sd, 5) == -1) {
    perror("Error at listen().\n");
    return errno;
}
```

The actual acceptance of clients will take place in an infinite loop, and it will be achieved by calling the `accept()` primitive. This will be followed by the creation of a corresponding thread for the current client, making sure that the server serves clients concurrently:

```
while (1) {
    if ((client = accept(sd, (struct sockaddr *) &from, &length)) < 0) {
        perror("Error at accept().\n");
        continue;
    }

    td = (struct thData *) malloc(sizeof(struct thData));
    td->idThread = thNumber++;
    td->cl = client;
    pthread_create(&th[thNumber], NULL, &treat, td);
}
```

5 Conclusions

The purpose of the project is to implement a Network monitoring application that can be used in a real life enviroment.
Some improvements could be the tranzition from the protocol http to https for an secure transmision of the logs from the actor the the central server. Another one could regard the application being opened in a browser,for simplicity.

6 Bibliographic References

- https://profs.info.uaic.ro/ computernetworks
- https://serverfault.com/questions/758244/how-to-configure-systemd-journal-

remote

- https://www.freedesktop.org/software/systemd/man/systemd-journal-remote.service.html