buyer seller Design

WO1 Clayton E. Williams

December 2023

Project Overview

`buyer_seller` is a two-part project, consisting of a buyer (client), and seller (server).The client reads a data file of ~50000 entries, consisting of an account number and order or payment, and sends it to the server through the use of Unix Domain Sockets. The server receives these transactions, and through the use of multi-threading, consolidates data from all 10 clients into a single account structure.

Project Structure

Project structure breakdown:

```
.
├── client_src
│   ├── client.c
│   └── client_helper.c
├── data
│   ├── seller10.dat
│   ├── seller1.dat
│   ├── seller2.dat
│   ├── seller3.dat
│   ├── seller4.dat
│   ├── seller5.dat
│   ├── seller6.dat
│   ├── seller7.dat
│   ├── seller8.dat
│   └── seller9.dat
├── doc
│   ├── design.pdf
│   ├── rubric.pdf
│   ├── testplan.pdf
│   └── writeup.pdf
├── include
│   ├── client_helper.h
│   ├── server_helper.h
│   └── shared.h
├── Makefile
├── output.txt
├── README.md
├── server_src
│   ├── server.c
│   └── server_helper.c
└── test
    ├── test_client_helper.c
    └── test_data
    │   └── 0_file
```

`client_src` - C source code files for the client program.
`data` - 10 unique data files to be used by `client`.
`doc` - Documentation for the project, design plan, writeup, testplan, and rubric.
`include` - Custom header files for both `client` and `server`.
`server_src` - C source code files for the server program.
`test` - C source code files for unit testing.

Data structures

There is one main data structure that will be used by both `client` and `server` for the purposes of maintaining account information:

```
typedef struct account_t {
    int amt_owed;
    uint32_t num_orders;
    uint32_t num_payments;
} account_t;
```

In addition, a second struct will be used by `server` to hold the 'package' details to be passed to the threads:

```
typedef struct pkg_t {
    account_t *client_accounts;
    ssize_t sockfd;
} pkg_t;
```

On the `client` side, one struct will be initialized by each client to track their transactions, while the `server` has one instance of the struct and is updated from transactions from all clients. **Code to manipulate the struct is protected by mutex lock to prevent undesired behavior or modification.**

Project Flow

Client

1. On startup, do validation against argv and argc. argc should only equal 2, the program and valid file option. Run the file argument, `argv[1]` through custom `validate_file()` function.
2. Initialize the `account_t` struct to be used by the client.
3. Create client socket, set path and family, and connect.
4. Read each line from the data file provided inside while loop using `getline`.
5. On each line read, serialize data into a 5 byte byte array for the purpose of having a custom protocol to send to server. This serialization makes it easy on the server side to confirm data transmission, as all transmissions, except for EOF are 5 bytes in length.

6. With each valid transmission, update `account_t`.
7. After file has been parsed and sent, print the values of each account.
8. Cleanup/free memory and exit. **The while loop has a conditional checking against SIGINT to exit the loop when system interrupt signal is sent to the client.**

Server

1. On startup, register the signal handler for SIGINT.
2. Initialize the `account_t` structure to hold account information for all client connections (up to 10).
3. Initialize an empty thread list of size 10, identified by `NUM_MAX_CLIENTS` defined in `shared.h`.
4. Create `sockaddr_un` struct for both server and client, and create socket. `bind`, then `listen` on that server socket.
5. Inside a while loop with exit condition of number of connections, (initialized to 0) < `NUM_MAX_ClIENTS`, server `accept` system call will wait for requests from clients. If the server socket returend is -1 and errno is 4, indicates SIGINT on the server side, continue loop to beginning where `SIGINT_FLAG` is evaluated and break loop to cleanup and exit.
6. If `accept` returns a good value, calloc the `pkg_t` struct for that client, setting the sockfd to fd returned from `accept`, and setting the `account_t` struct.
7. Create a new thread, hadning it a void pointer to the `pkg_t` struct, and `thread_func()`.
8. Inside `thread_func`, the `account_t` logic is the same as `client`, however the lines of code manipulating the member variables are wrapped in mutex lock.
9. After a successful thread creation, number of connections is incremented and loop restarts.
10. When loop exit condition is met, join all threads from the thread list.
11. After threads have been joined, print the information for each account.
12. Close the server socket and exit.