SYSTEM PROGRAMMING

MIDTERM PROJECT REPORT

STUDENT: OĞUZ ALP KELEŞ

STUDENT NUMBER: 220104004834

Contents

1- General Architecture And Working

1.1- Program Flow (High Level):

1-Client reads a client file.

2-Creates a server connection request and sends it to server.

3-Server accepts it and creates a teller.

4-Teller reads actual request information from the client which includes details such as operation type bank id etc.

5-Teller makes integrity checks if request is applicable or not.

6-Teller sends request information to server for database to be updated.

7-Server Updates the database.

8-Teller sends response to client.

9-Signal SIGTERM is delivered to server process in order to end the program.

10-After that, updated database and log file including the information related to requests done can be seen.

1.2- Client Flow:

1-Client process, reads a client file that is given as argument.

2-For every line in the client file (for every client/request), create a child inside a loop without waiting. The aim is to make it concurrent where clients request service from the server in an unordered way.

3-Using specific pid of the child create a client_fifo_%d specific to each client request which will then be used in communication with the teller.

4-Each of the children (requests) sends a “connection request” to the server that only includes information about client_fifo_%d that wants to connect. (Not the details such as type of operation etc.)

5-Using client_fifo_%d, send actual request to teller which contains detailed information about operation, amount and other necessary information.

7-Read the response from the teller.

8-Wait for all child processes to ensure proper clean up of resources.

Notes: In case of termination signal, fifos are cleaned via handler function called “cleanup_fifo”.

1.3- Server Flow:

1-Program sets the signal handlers to ensure proper cleanup when SIGTERM is delivered. (Note that the only way to stop the server is to deliver CTRL+C (SIGTERM) signal from the terminal.

2-Creates necessary SERVER_FIFO and shared memory to be used between tellers and server.

3-Loads the database from database.txt file. (This file should exist in order for program to run).

4-Initialize the semaphore that will protect the shared data.

5-Server forks a handler that will deal with updating the database. (It is added to help server support concurrency and to be consistent with what the homework document asks. By this way handler can listen for incoming requests and leave update to that handler)

6-This handler reads valid requests from the teller and make updates to database ensuring that no one except the server make changes to database as required in the homework document.

7-Server enters into main server loop where it listens for incoming requests and then passes these requests to Tellers.

8-Teller function creates a child process (actual teller) which executes “func“ (function given as its argument) as desired in the homework document and validation job is done for the request.

9-These tellers send response message regarding the result of the requests to the client.

10-At the end, all the child processes are waited using “waitTeller” function stated in the homework document.

11-When SIGTERM (CTRL+C) signal is arrived, server makes all the clean up and exit. (In this handler function, updated database is written to the database file, more on design choices section of this document, please read.)

2- My Design Choices

2.1- Related To Database:

-First of all, database file should exist in order for program to run.

-The file name is **database.txt**.

-Format of the database file is as follows; bank id followed by a space and amount that exists in the account. If you want to change it please obey to this format.

-**Example of a line** inside the database.txt file ; **BankID_X Y where X denotes a number like 01 and Y denotes an integer representing the amount. (BankID_01 2500)**

-Program **supports at most 100 accounts.** If you want to change it **you can change the constant macro MAX_ACCOUNTS** in server.c file.

! In my program database information is stored inside an Account (a structure) array . As a design choice, server loads the database from the file and stores it inside that array. I make the updates to the database on that array and write the current database (information inside that array) to the file only when the server execution is stopped/ ended by a signal (SIGTERM). WHY ? Because it seems logical to me that when each time server runs it means a server session and making updates to database file directly would cause overhead of file I/O operations. That is why when server session ends, I write the data to the database file for later use of it when server runs again.

2.2- Related To Log File:

-First of all, log file should exist in order for program to run.

-The file name is **AdaBank.bankLog .**

-Format of the log file is as follows; bank id followed by a space , operation (D for deposit, W for withdraw), and amount.

-**Example of a line** inside the log file ; **BankID_01 D 300.** Meaning that account BankID_01 deposited 300.

! Every time server runs again, log file is written from scratch. WHY ? Because it seems logical to me that log is written for a server session. Also note that log file stores only the successful operations since errors are printed via terminal. I did not want to have duplicate information.

2.3- Additional Notes:

-Please note that some outputs **may not** look ordered, it is because concurrency. If you look at what is written in both server and client terminals, you can validate that operations are successfully conducted.

-To say a few words on resource clean up; **client** waits all of its child processes and in case it receives a signal it has a handler that cleans up the fifo created. **Server** has a signal handler that cleans up all the resources including the shared memory created when delivered a signal. Server also waits for child processes to avoid orphaned child processes.

3- Detailed Explanations Of The Code And Functions

3.1- Functions In server.c:

1-init_log_file(): This function creates a new log file named AdaBank.bankLog at the beginning of the server execution. It writes a timestamp message at the top of the file to indicate when the bank server started logging. This provides a clear historical starting point for all further transaction logs.

2-log_transaction(const char *account_id, const char *operation, int amount): This function appends a transaction record to the AdaBank.bankLog file. Each transaction records the account ID, the operation (deposit or withdraw), and the transaction amount. It ensures that every operation performed on the bank database is persistently logged for auditing.

3-finalize_log_file(): When the server is terminating, this function appends the line representing the end of the log file to the log file. It signals the logical end of all recorded transactions in that session, ensuring the log is properly closed. This provides a neat closure for the server's transaction history.

4-load_database_from_file(): This function reads the contents of the database.txt file and loads the account information into the server's shared memory. Each line in the file represents an account ID and its corresponding balance. It populates the server's in-memory database so that it can continue from where it left off across runs.

5-save_database_to_file(int sig): This function writes the current state of the bank database from shared memory back into database.txt. It is also the signal handler for SIGINT, ensuring the database is saved safely when the server is shut down. Additionally, it kills all active child processes (handlers and tellers) and properly frees all system resources like FIFOs, semaphores, and shared memory.

6-update_database(const char *account_id, const char *operation, int amount, int possible_request, char *response): This function updates the shared memory bank database according to the request received. It handles creating new accounts, making deposits, performing withdrawals, and removing accounts when their balance reaches zero. If a request cannot be fulfilled (e.g., insufficient funds or invalid account), it returns appropriate error messages for logging and client communication.

7-void *func(void *arg): This function represents the work of a teller process that serves an individual client. It reads the client request, validates if the operation is possible by checking

the database (protected with semaphore), sends the validated request to the server via a pipe, and finally responds to the client. Each teller process operates independently and ensures safe concurrent access to shared data.

8-pid_t Teller(void *func, void *arg_func): This function forks a new child process to act as a teller. The newly created teller process executes the provided func function (handling one client's transaction). It allows the server to serve multiple clients concurrently by offloading individual client operations to child processes.

9-int waitTeller(pid_t pid, int *status): This function waits for the termination of a child teller process identified by its PID. It ensures that all child tellers cleanly terminate after finishing their tasks, avoiding the creation of zombie processes. This function is essential for correct resource management in the server.

10-void setup_sigaction(int signum, void (*handler)(int)): This utility function sets up signal handling behavior using the more reliable sigaction() system call. It allows the server to correctly handle signals like SIGINT and SIGTERM, ensuring graceful shutdown and resource cleanup. Compared to the older signal() function, sigaction() provides more control and robustness.

3.2- Functions In client.c:

1-void printMsg(char operations[][10], int amounts[], int client_num): This function prints a friendly status message when a client connects to the bank server. It informs the user which client is connecting and which operation (deposit or withdrawal) they are attempting with how much amount. This adds clarity to the client's console output and helps with debugging.

2-void cleanup_fifo(int sig): This function acts as a signal handler to clean up the client's private FIFO when the client is interrupted (via signals like SIGINT or SIGTERM). It ensures that no leftover FIFO files remain in the filesystem, which could otherwise cause resource leakage or program malfunctions in later runs. After cleaning, it terminates the client process with an exit status of 1.

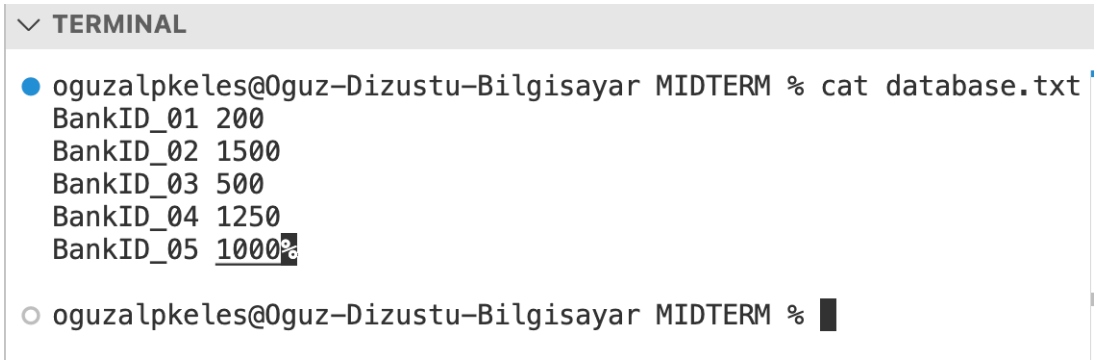
3-void setup_sigaction(int signum, void (*handler)(int)): The same as in server.c

Notes: Main() functions are not explained since they are covered in section 1 of this report mentioning the program flow.

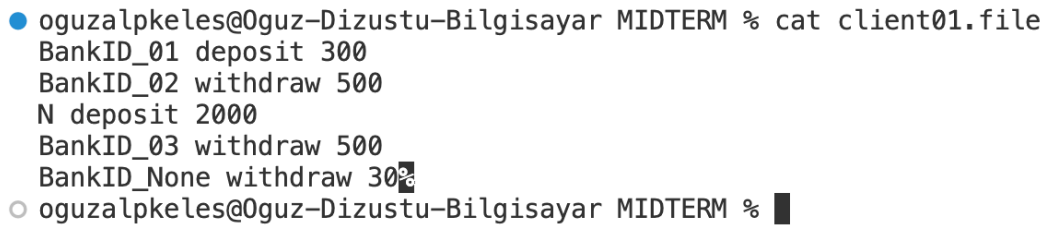
4- Testing/Outputs And How To Reproduce The Results

-In the initial setup, contents of the files are as follows;

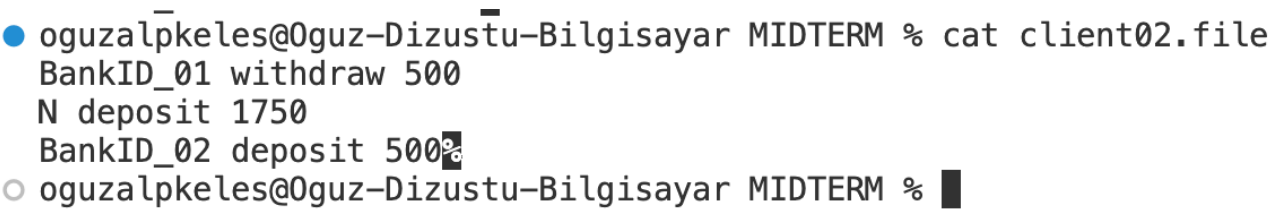
> cat database.txt



> cat client01.file



> cat client02.file



In order to produce the test results please run the following commands (Note that test cases include all the scenarios, successful deposit, withdrawal, account creation, deletion and no valid bank id (BankID_None));

> make or make all

> ./server (in server terminal)

> ./client client01.file (in client terminal)

> ./client client02.file (in client terminal)

> CTRL+C (in server terminal, at this stage you can check AdaBank.bankLog and database.txt files to ensure results are correct)

> ./client client01.file (in client terminal, this represents the situation where server is not active)

OUTPUTS WHEN RUN THE COMMANDS IN THIS ORDER ARE AS FOLLOWS ;

Output Of Server Terminal

```
TERMINAL
oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM % make clean
rm -f server client
oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM % make all
gcc  server.c -o server
gcc  client.c -o client
oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM % ./server
Creating the bank database...
Adabank is active... Waiting for clients at server_fifo

Teller PID70453 is active serving the client.
BankID_01 Deposit successful. New balance: 500

Teller PID70456 is active serving the client.
Withdrawal successful. Account BankID_03 removed.

Teller PID70457 is active serving the client.
New account BankID_06 created with balance 2000

Teller PID70458 is active serving the client.
BankID_02 Withdrawal successful. Remaining balance: 1000

Teller PID70459 is active serving the client.
Invalid request for BankID_None

Teller PID70682 is active serving the client.
Withdrawal successful. Account BankID_01 removed.

Teller PID70683 is active serving the client.
New account BankID_07 created with balance 1750

Teller PID70684 is active serving the client.
BankID_02 Deposit successful. New balance: 1500
^C
Signal received, cleaning up and closing active tellers.
Adabank says “Bye”...
oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM %
```

Output Of Client Terminal

```
oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM %
./client client01.file
Reading clients.file... Found 5 requests.
Client00 connected..depositing 300 credits
Client03 connected..withdrawing 500 credits
Client02 connected..depositing 2000 credits
Client01 connected..withdrawing 500 credits
Response from server for request 1: Request accepted and being processed.
Client04 connected..withdrawing 30 credits
Response from server for request 4: Request accepted and being processed.
Response from server for request 3: Request accepted and being processed.
Response from server for request 2: Request accepted and being processed.
Response from server for request 5: Insufficient balance or invalid account.
oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM %

oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM %

oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM %
./client client02.file
Reading clients.file... Found 3 requests.
Client00 connected..withdrawing 500 credits
Client01 connected..depositing 1750 credits
Client02 connected..depositing 500 credits
Response from server for request 1: Request accepted and being processed.
Response from server for request 2: Request accepted and being processed.
Response from server for request 3: Request accepted and being processed.
```

After These, Content Of The Files;

> cat database.txt

```
oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM % cat database.txt
BankID_02 1500
BankID_04 1250
BankID_05 1000
BankID_06 2000
BankID_07 1750
oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM %
```

> cat AdaBank.bankLog

```
oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM % cat AdaBank.bankLog
# Adabank Log file updated @18:07 April 27 2025
BankID_01 D 300
BankID_03 W 500
BankID_06 D 2000
BankID_02 W 500
BankID_01 W 500
BankID_07 D 1750
BankID_02 D 500
## end of log.
oguzalpkeles@Oguz-Dizustu-Bilgisayar MIDTERM %
```

Please note that after the operations given these client files, outputs are correct and reflect the expected behavior such as accounts with ids 1 and 3 are removed since their amount became zero and new accounts are created and given new ids by the server. You can try with different inputs but please be careful that the format of the files match as my files.