NAME: VICTOR OMONDI ARWA

REG. NUMBER: SCT221-0112/2019

UNIT CODE: ICS 2305

UNIT NAME: SYSTEMS PROGRAMMING

TASK: ASSIGNMENT

TASK: MAKE A FINANCIAL SYSTEM THAT EXCHANGES TRANSACTIONS USING THE CONCEPT OF SOCKET PROGRAMMING

This project implements a simple financial transaction system using C socket programming, where a client interacts with a server to perform basic banking operations. The server acts as a bank, maintaining an account balance and processing client requests such as checking the balance, depositing funds, and withdrawing money. The client connects to the server using TCP sockets, ensuring reliable and ordered message delivery. When a user enters a command, the client sends it to the server, which interprets it, updates the account data if necessary, and sends back a response. The server listens for incoming connections, accepts a client, and continuously exchanges data until the client requests termination. Both server and client use standard socket APIs like socket(), bind(), listen(), accept(), connect(), send(), and recv() to handle network communication.

The system is a client-server application where; the server acts as a bank that holds and manages an account balance and the client represents a user performing financial transactions such as checking balance, depositing, or withdrawing money.

Communication is done using TCP sockets**,** ensuring reliable and ordered delivery of messages between client and server.

IMPLEMENTATION

1. SERVER-SIDE PROCESS

The server-side process is as follows in the implementation:

The server uses socket(AF_INET, SOCK_STREAM, 0) functionto create a TCP socket. It then calls bind()system call to attach to a specific IP address and port.
Calls listen() to prepare for incoming connections. Waits for a client with accept(), which returns a new socket for communication.
Reads client commands (BALANCE, DEPOSIT <amount>, WITHDRAW <amount>, EXIT) using recv() or read().
Parses the command with functions like strncmp() and sscanf() to determine the action. Updates an internal balance variable and sends responses using send().

2. CLIENT_SIDE PROCESS

Creates its own TCP socket with socket(). Uses connect() system call to establish a link to the server's IP and port.
Takes user input (e.g., "DEPOSIT 100") and sends it to the server with send(). Waits for the server's response and displays it to the user.
Repeats until "EXIT" is sent, after which the connection is closed.

SERVER PROGRAM (Server.c)

```
// server.c — Simple bank server using TCP sockets (POSIX)

// Build: gcc -O2 -Wall -Wextra -o server server.c

// Run:   ./server

// Default: 127.0.0.1:5000

#include <arpa/inet.h>

#include <errno.h>

#include <netinet/in.h>

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>

#define PORT 5000
```

```
#define BACKLOG 1

#define BUFSZ 1024

typedef struct {

    char username[32];

    double balance;

} Account;

static Account acct = { "user1", 1000.0 };

static void die(const char *msg) {

    perror(msg);

    exit(EXIT_FAILURE);

}

static void trim_newline(char *s) {

    size_t n = strlen(s);

    while (n > 0 && (s[n-1] == '\n' || s[n-1] == '\r')) {

        s[n-1] = '\0';

        n--;

    }

}

static bool parse_amount(const char *s, double *out) {

    if (!s || !*s) return false;
```

```
    char *end = NULL;
    errno = 0;
    double v = strtod(s, &end);
    if (errno != 0 || end == s) return false;
    // Ensure no junk after number (allow trailing spaces)
    while (*end == ' ') end++;
    if (*end != '\0') return false;
    *out = v;
    return v >= 0.0; // disallow negative amounts
}
static void to_upper(char *s) {
    for (; *s; ++s) {
        if (*s >= 'a' && *s <= 'z') *s = (char)(*s - 'a' + 'A');
    }
}
static void process_request(const char *req_line, char *resp, size_t resp_sz) {
    // Work on a modifiable copy
    char buf[BUFSZ];
    snprintf(buf, sizeof(buf), "%s", req_line);
    trim_newline(buf);
```

```
    // Tokenize: COMMAND [ARG]
    char *cmd = strtok(buf, " ");
    if (!cmd) {
        snprintf(resp, resp_sz, "ERR Invalid command\n");
        return;
    }
    to_upper(cmd);
    if (strcmp(cmd, "BALANCE") == 0) {
        snprintf(resp, resp_sz, "OK Balance: %.2f\n", acct.balance);
        return;
    }
    if (strcmp(cmd, "DEPOSIT") == 0) {
        char *arg = strtok(NULL, "");
        if (!arg) { snprintf(resp, resp_sz, "ERR Usage: DEPOSIT <amount>\n"); return;
}
        while (*arg == ' ') arg++;
        double amt;
        if (!parse_amount(arg, &amt)) {
            snprintf(resp, resp_sz, "ERR Invalid amount\n");
            return;
        }
```

```
        acct.balance += amt;
        snprintf(resp, resp_sz, "OK Deposited %.2f. New balance: %.2f\n", amt,
acct.balance);
        return;
    }
    if (strcmp(cmd, "WITHDRAW") == 0) {
        char *arg = strtok(NULL, "");
        if (!arg) { snprintf(resp, resp_sz, "ERR Usage: WITHDRAW <amount>\n");
return; }
        while (*arg == ' ') arg++;
        double amt;
        if (!parse_amount(arg, &amt)) {
            snprintf(resp, resp_sz, "ERR Invalid amount\n");
            return;
        }
        if (amt > acct.balance) {
            snprintf(resp, resp_sz, "ERR Insufficient funds\n");
            return;
        }
        acct.balance -= amt;
```

```
        snprintf(resp, resp_sz, "OK Withdrew %.2f. New balance: %.2f\n", amt,
acct.balance);
        return;
    }
    if (strcmp(cmd, "EXIT") == 0) {
        snprintf(resp, resp_sz, "OK Bye\n");
        return;
    }
    snprintf(resp, resp_sz, "ERR Unknown command\n");
}
int main(void) {
    int srv = socket(AF_INET, SOCK_STREAM, 0);
    if (srv < 0) die("socket");
    int opt = 1;
    if (setsockopt(srv, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) < 0)
        die("setsockopt");
    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT);
    addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK); // 127.0.0.1
```

```
if (bind(srv, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    die("bind");
if (listen(srv, BACKLOG) < 0)
    die("listen");
printf("Bank server listening on 127.0.0.1:%d ...\n", PORT);
for (;;) {
    struct sockaddr_in cli;
    socklen_t clen = sizeof(cli);
    int cfd = accept(srv, (struct sockaddr *)&cli, &clen);
    if (cfd < 0) die("accept");
    char ip[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &cli.sin_addr, ip, sizeof(ip));
    printf("Connected: %s:%d\n", ip, ntohs(cli.sin_port));
    char line[BUFSZ];
    ssize_t n;
    for (;;) {
        n = recv(cfd, line, sizeof(line) - 1, 0);
        if (n <= 0) {
            if (n < 0) perror("recv");
            break;
```

```
}
line[n] = '\0';
// handle possibly multiple lines in one recv; process line-by-line
char *p = line;
while (p && *p) {
    char *nl = strchr(p, '\n');
    char one[BUFSZ];
    if (nl) {
        size_t len = (size_t)(nl - p);
        if (len >= sizeof(one)) len = sizeof(one) - 1;
        memcpy(one, p, len);
        one[len] = '\0';
        p = nl + 1;
    } else {
        // if no newline, treat whole buffer as one command
        snprintf(one, sizeof(one), "%s", p);
        p = NULL;
    }
    char resp[BUFSZ];
    process_request(one, resp, sizeof(resp));
```

```
            // If EXIT, send response then close
            if (strncmp(one, "EXIT", 4) == 0 || strncmp(one, "exit", 4) == 0) {
                send(cfd, resp, strlen(resp), 0);
                goto close_client;
            }
            send(cfd, resp, strlen(resp), 0);
        }
    }

close_client:
    printf("Client disconnected.\n");
    close(cfd);
  }

  // close(srv); // Unreachable with for(;;)
  return 0;
}
```

CLIENT PROGRAM (Client.c)

```
// client.c — Simple bank client using TCP sockets (POSIX)
// Build: gcc -O2 -Wall -Wextra -o client client.c
```

```
// Run:   ./client
// Type commands, e.g. BALANCE, DEPOSIT 200, WITHDRAW 50, EXIT

#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define SERVER_IP "127.0.0.1"
#define SERVER_PORT 5000
#define BUFSZ 1024

static void die(const char *msg) {
    perror(msg);
    exit(EXIT_FAILURE);
```

```
}

int main(void) {
    int fd = socket(AF_INET, SOCK_STREAM, 0);
    if (fd < 0) die("socket");

    struct sockaddr_in srv;
    memset(&srv, 0, sizeof(srv));
    srv.sin_family = AF_INET;
    srv.sin_port = htons(SERVER_PORT);
    if (inet_pton(AF_INET, SERVER_IP, &srv.sin_addr) != 1) die("inet_pton");

    if (connect(fd, (struct sockaddr *)&srv, sizeof(srv)) < 0)
        die("connect");

    printf("Connected to bank server %s:%d\n", SERVER_IP, SERVER_PORT);
    printf("Commands: BALANCE | DEPOSIT <amount> | WITHDRAW <amount> |
EXIT\n");

    char sendbuf[BUFSZ];
    char recvbuf[BUFSZ];
```

```
for (;;) {
    printf("> ");
    if (!fgets(sendbuf, sizeof(sendbuf), stdin)) break;

    // Ensure newline-terminated message to server
    size_t len = strlen(sendbuf);
    if (len == 0 || sendbuf[len-1] != '\n') {
        if (len < sizeof(sendbuf) - 1) {
            sendbuf[len] = '\n';
            sendbuf[len+1] = '\0';
        }
    }

    if (send(fd, sendbuf, strlen(sendbuf), 0) < 0) die("send");

    ssize_t n = recv(fd, recvbuf, sizeof(recvbuf) - 1, 0);
    if (n <= 0) {
        if (n < 0) perror("recv");
        printf("Server closed the connection.\n");
```

```
            break;
        }
        recvbuf[n] = '\0';
        printf("%s", recvbuf);

        if (strncasecmp(sendbuf, "EXIT", 4) == 0) break;
    }

    close(fd);
    return 0;
}
```

COPMPILE AND RUN THE PROGRAMS

gcc server.c -o server

gcc client.c -o client

Run the server **(server.exe)** in the first terminal and the client **(client.exe)** in the second terminal for the server and client to communicate.

The server will start listening for incoming connections on the specified port (e.g., 8080). The client will connect to the server. You can now run the commands for the transactions for example DEPOSIT with the amount to deposit, WITHDRAW with the amount to withdraw, BALANCE command to chech balance. EXIT command to exit the application.

THE SCREEN SHOTS BELOW SHOWS SOME OF THE EXAMPLES OF THE CLIENT-SERVER COMMUNICATION IN THE FINANCIAL SYSTEM

SERVER-SIDE PROCESSING

CLIENT-SIDE PROCESSING