Assignment 2 – ThreadBank
Student: Oliver von Schantz
Student number: 101856454
Date started: 19.11.2025

1. Initial understanding of the assignment
I am implementing a multithreaded bank server (“ThreadBank”). The server has N service desks, each in its own thread. Clients connect to the server using inter-process communication (IPC) over a UNIX domain socket and send simple text commands to check balances, withdraw, deposit, and transfer money. Account data should be stored on disk so it can survive restarts. Threads must use synchronization (read–write locks) so that multiple clients cannot corrupt account data. The server must also handle SIGINT/SIGTERM and shut down cleanly.

2. Project structure
I created the initial project structure for ThreadBank. In the directory 101856454-as2 I added:
- server.c – for the bank server
- client.c – for the bank client
- Makefile – a build script that compiles both programs with -Wall -pedantic -std=c11 -g and links the server with -pthread

At this stage I just wanted to test the files. I made the server only print a short “starting” message, and the client printed “ready” without actually talking to the server. The goal was just to have something that compiles and runs.

3. First compilation and debugging
I ran “make” in the 101856454-as2 directory and initially got a long list of compiler errors. I was stuck for a while trying to understand what went wrong. I later realized that one of the mistakes was that I had forgotten to include “#” in #include <stdio.h> at the top of client.c. This caused the compiler to treat the line as invalid C code and then all the standard library symbols such as size_t, printf, memset, etc. were reported as unknown.

By correcting the preprocessor lines (including the proper headers: <stdio.h>, <stdlib.h>, and <string.h>), the errors disappeared. After fixing these issues, “make” completed successfully and I could run ./server and ./client without crashes.

4. First client–server communication over a UNIX socket
I implemented the first real communication between my ThreadBank client and server using a UNIX domain socket at /tmp/threadbank.sock. The server creates this socket, binds and listens, accepts one client connection, sends a READY line to the client, and then responds with simple “ok: ... “ messages for any command. The client connects to the socket, waits for the ready line, and then prints “ready” to stdout. After that it forwards each line from stdin to the server and forwards one reply line back to stdout.