CSE344 -- MIDTERM

28/04/2025

Cemal BOLAT

STUDENT NO: 210104004010

- **Introduction**
- **System Architecture**
- **Testing with Multiple Scenerios & Valgrind**
- **Project Development Steps with source codes**
- **Challanges that I faced & My Solutions**
- **As mentioned in Teams**
    - **My aim for Midterm is 80+/100 haven’t faced any problem.**

Introduction:

This project focuses on the design and implementation of a robust client-server based simple banking simulator, targeting full compliance with the advanced requirements for full credit. The system enables fundamental banking operations, such as account creation, deposits, and withdrawals, by employing inter-process communication (IPC) and process synchronization techniques.

The main server, representing the core of the bank, is responsible for managing account information, updating the transaction log, and maintaining database consistency. It operates exclusively on the bank database to ensure data integrity. For every client connection, the server dynamically spawns a dedicated teller process. These teller processes validate and process client requests, interacting with the main server through shared memory and semaphores instead of traditional pipes, thus ensuring high-performance and synchronized operations.

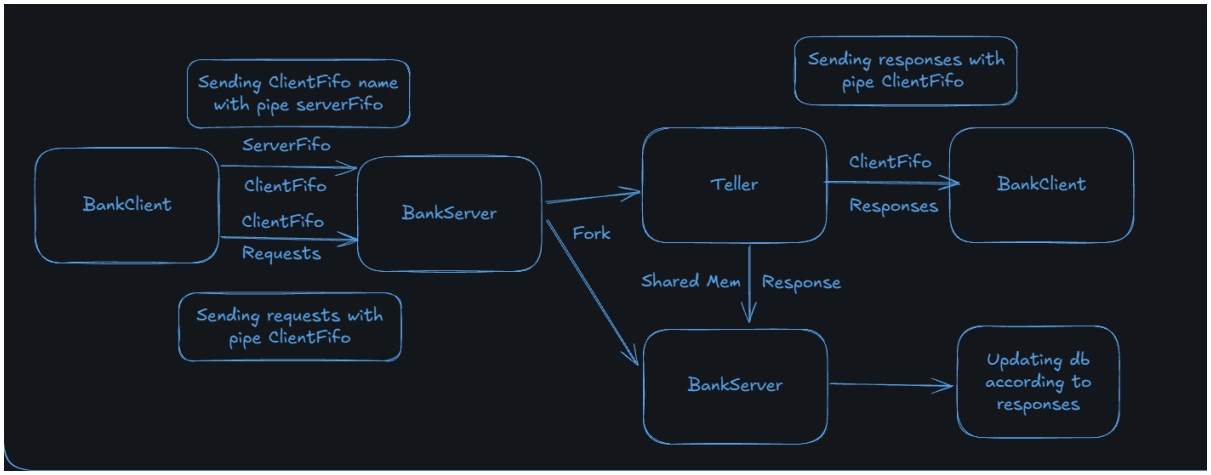
Communication between clients and the main server is established through a server FIFO, while each teller communicates back with its respective client through a client-generated named FIFO. Special attention is given to handling signals and ensuring that both clients and tellers can gracefully handle unexpected terminations without jeopardizing server stability.

To achieve full credit, the teller processes are created not through a simple fork() system call but by implementing a custom Teller function that launches a process executing a specific function with provided arguments. Additionally, teller-server interactions use shared memory regions combined with semaphore synchronization for data exchange and transaction coordination.

Extensive testing, including simultaneous operation of up to 20 clients, has been conducted. Careful memory management and leak checks were performed to ensure system reliability. A comprehensive makefile, complete test plans, and multiple client scenarios demonstrate that the system meets all specified requirements efficiently and robustly.

•

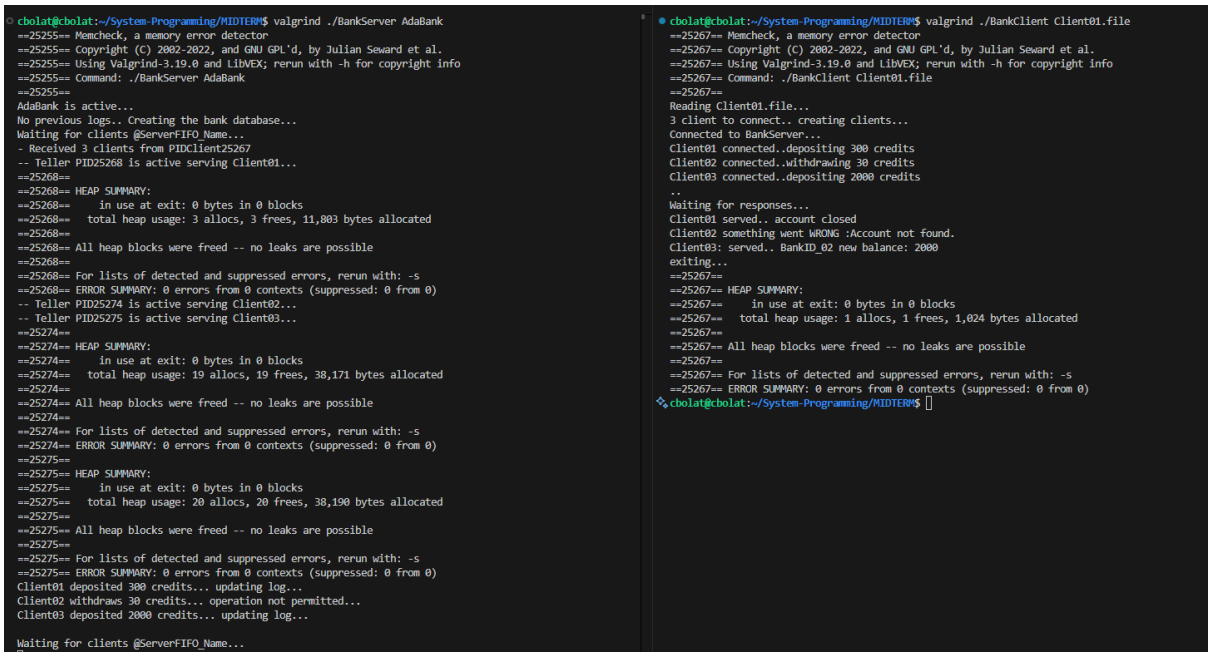
2.System Architecture



Testing with Multiple Scenerios & Valgrind

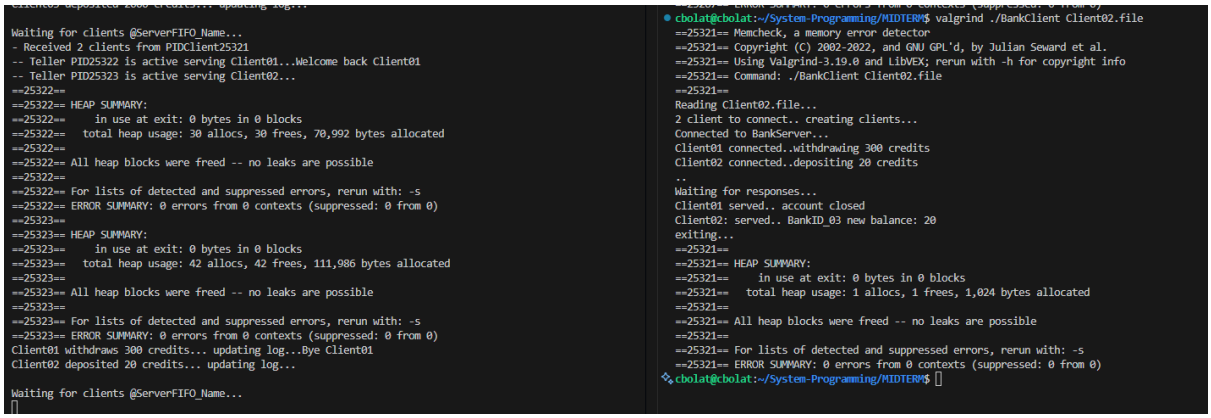
- **(at the end I will enter the non valgrind version for BankServer because its quite hard to detach logs properly)**

TESTING WITHOUT PROBLEM with PDF's test cases 1by1.



Valgrind output is shows there is no leaks on no purposes

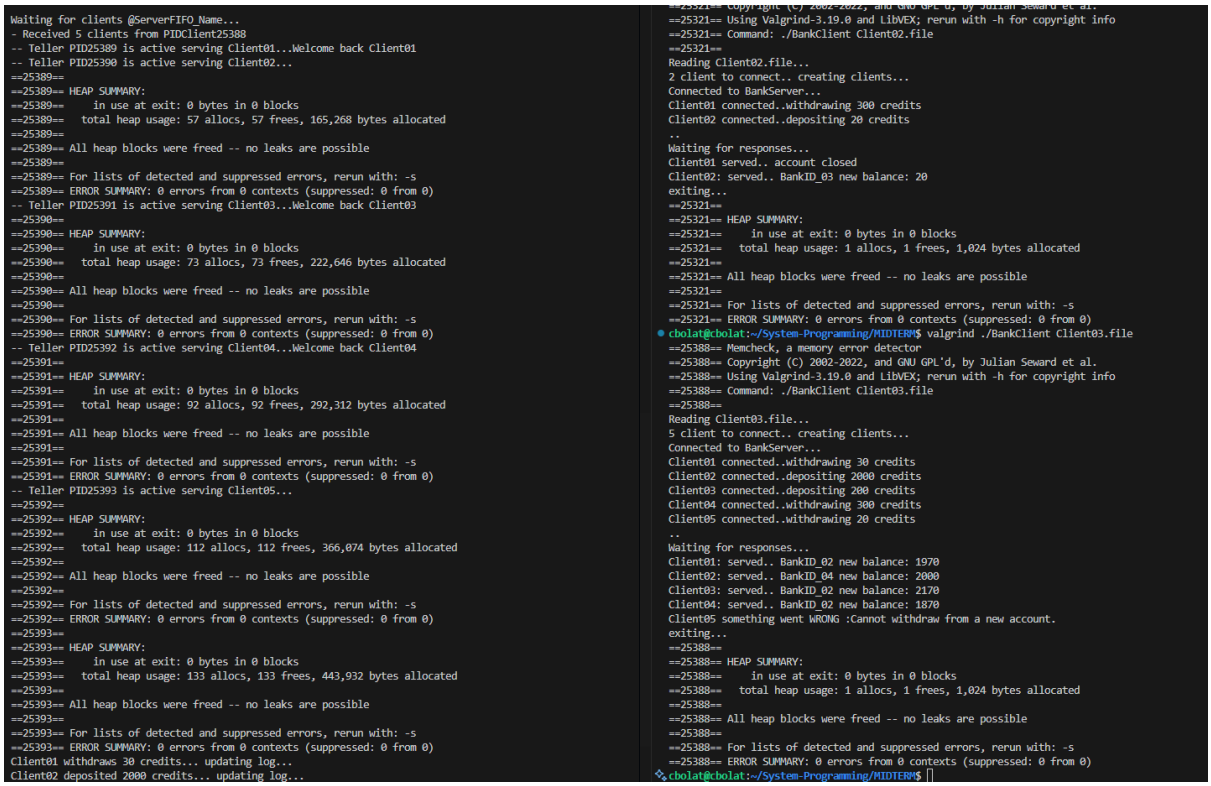
Giving Client02.file as entry



```
Waiting for clients @ServerFIFO_Name...
- Received 2 clients from PIDClient25321
-- Teller PID25322 is active serving Client01...Welcome back Client01
-- Teller PID25323 is active serving Client02...
==25322==
==25322== HEAP SUMMARY:
==25322==     in use at exit: 0 bytes in 0 blocks
==25322==   total heap usage: 30 allocs, 30 frees, 70,992 bytes allocated
==25322==
==25322== All heap blocks were freed -- no leaks are possible
==25322==
==25322== For lists of detected and suppressed errors, rerun with: -s
==25322== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==25323==
==25323== HEAP SUMMARY:
==25323==     in use at exit: 0 bytes in 0 blocks
==25323==   total heap usage: 42 allocs, 42 frees, 111,986 bytes allocated
==25323==
==25323== All heap blocks were freed -- no leaks are possible
==25323==
==25323== For lists of detected and suppressed errors, rerun with: -s
==25323== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Client01 withdraws 300 credits... updating log...Bye Client01
Client02 deposited 20 credits... updating log...

Waiting for clients @ServerFIFO_Name...
```

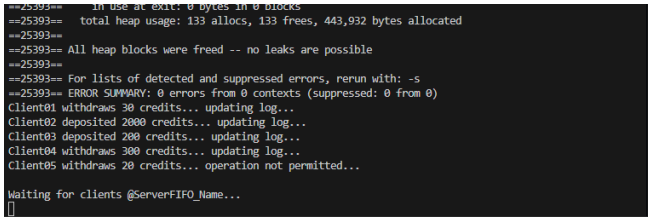
```
cbolat@cbolat:~/System-Programming/MIDTERM$ valgrind ./BankClient Client02.file
==25321== Memcheck, a memory error detector
==25321== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==25321== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==25321== Command: ./BankClient Client02.file
==25321==
Reading Client02.file...
2 client to connect.. creating clients...
Connected to BankServer...
Client01 connected..withdrawing 300 credits
Client02 connected..depositing 20 credits
..
Waiting for responses...
Client01 served.. account closed
Client02: served.. BankID_03 new balance: 20
exiting...
==25321==
==25321== HEAP SUMMARY:
==25321==     in use at exit: 0 bytes in 0 blocks
==25321==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==25321==
==25321== All heap blocks were freed -- no leaks are possible
==25321==
==25321== For lists of detected and suppressed errors, rerun with: -s
==25321== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cbolat@cbolat:~/System-Programming/MIDTERM$
```

Giving Client03.file as entry



```
Waiting for clients @ServerFIFO_Name...
- Received 5 clients from PIDClient25388
-- Teller PID25389 is active serving Client01...Welcome back Client01
-- Teller PID25390 is active serving Client02...
==25389==
==25389== HEAP SUMMARY:
==25389==     in use at exit: 0 bytes in 0 blocks
==25389==   total heap usage: 57 allocs, 57 frees, 165,268 bytes allocated
==25389==
==25389== All heap blocks were freed -- no leaks are possible
==25389==
==25389== For lists of detected and suppressed errors, rerun with: -s
==25389== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID25391 is active serving Client03...Welcome back Client03
==25390==
==25390== HEAP SUMMARY:
==25390==     in use at exit: 0 bytes in 0 blocks
==25390==   total heap usage: 73 allocs, 73 frees, 222,646 bytes allocated
==25390==
==25390== All heap blocks were freed -- no leaks are possible
==25390==
==25390== For lists of detected and suppressed errors, rerun with: -s
==25390== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID25392 is active serving Client04...Welcome back Client04
==25391==
==25391== HEAP SUMMARY:
==25391==     in use at exit: 0 bytes in 0 blocks
==25391==   total heap usage: 92 allocs, 92 frees, 292,312 bytes allocated
==25391==
==25391== All heap blocks were freed -- no leaks are possible
==25391==
==25391== For lists of detected and suppressed errors, rerun with: -s
==25391== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID25393 is active serving Client05...
==25392==
==25392== HEAP SUMMARY:
==25392==     in use at exit: 0 bytes in 0 blocks
==25392==   total heap usage: 112 allocs, 112 frees, 366,074 bytes allocated
==25392==
==25392== All heap blocks were freed -- no leaks are possible
==25392==
==25392== For lists of detected and suppressed errors, rerun with: -s
==25392== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==25393==
==25393== HEAP SUMMARY:
==25393==     in use at exit: 0 bytes in 0 blocks
==25393==   total heap usage: 133 allocs, 133 frees, 443,932 bytes allocated
==25393==
==25393== All heap blocks were freed -- no leaks are possible
==25393==
==25393== For lists of detected and suppressed errors, rerun with: -s
==25393== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Client01 withdraws 30 credits... updating log...
Client02 deposited 2000 credits... updating log...
```

```
==25321== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==25321== Command: ./BankClient Client02.file
==25321==
Reading Client02.file...
2 client to connect.. creating clients...
Connected to BankServer...
Client01 connected..withdrawing 300 credits
Client02 connected..depositing 20 credits
..
Waiting for responses...
Client01 served.. account closed
Client02: served.. BankID_03 new balance: 20
exiting...
==25321==
==25321== HEAP SUMMARY:
==25321==     in use at exit: 0 bytes in 0 blocks
==25321==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==25321==
==25321== All heap blocks were freed -- no leaks are possible
==25321==
==25321== For lists of detected and suppressed errors, rerun with: -s
==25321== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cbolat@cbolat:~/System-Programming/MIDTERM$ valgrind ./BankClient Client03.file
==25388== Memcheck, a memory error detector
==25388== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==25388== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==25388== Command: ./BankClient Client03.file
==25388==
Reading Client03.file...
5 client to connect.. creating clients...
Connected to BankServer...
Client01 connected..withdrawing 30 credits
Client02 connected..depositing 2000 credits
Client03 connected..depositing 200 credits
Client04 connected..withdrawing 300 credits
Client05 connected..withdrawing 20 credits
..
Waiting for responses...
Client01: served.. BankID_02 new balance: 1970
Client02: served.. BankID_04 new balance: 2000
Client03: served.. BankID_02 new balance: 2170
Client04: served.. BankID_02 new balance: 1870
Client05 something went WRONG :Cannot withdraw from a new account.
exiting...
==25388==
==25388== HEAP SUMMARY:
==25388==     in use at exit: 0 bytes in 0 blocks
==25388==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==25388==
==25388== All heap blocks were freed -- no leaks are possible
==25388==
==25388== For lists of detected and suppressed errors, rerun with: -s
==25388== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cbolat@cbolat:~/System-Programming/MIDTERM$
```



```
==25393==   total heap usage: 133 allocs, 133 frees, 443,932 bytes allocated
==25393==
==25393== All heap blocks were freed -- no leaks are possible
==25393==
==25393== For lists of detected and suppressed errors, rerun with: -s
==25393== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Client01 withdraws 30 credits... updating log...
Client02 deposited 2000 credits... updating log...
Client03 deposited 200 credits... updating log...
Client04 withdraws 300 credits... updating log...
Client05 withdraws 20 credits... operation not permitted...

Waiting for clients @ServerFIFO_Name...
```

NO Leaks at all

(CTRL+C) signal send to BankServer after all these 3 Jobs

```
Waiting for clients @ServerFIFO_Name...
^CSignal received closing active Tellers...
Removing ServerFIFO... Updating log file...
AdaBank says "Bye"...
==25255==
==25255== HEAP SUMMARY:
==25255==     in use at exit: 0 bytes in 0 blocks
==25255==   total heap usage: 153 allocs, 153 frees, 521,771 bytes allocated
==25255==
==25255== All heap blocks were freed -- no leaks are possible
==25255==
==25255== For lists of detected and suppressed errors, rerun with: -s
==25255== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cbolat@cbolat:~/System-Programming/MIDTERM$
```

Log File after execution.

MIDTERM > AdaBank.bankLog

```
1  # Adabank Log file updated @Sun Apr 27 22:14:33 2025
2
3  # BankID_01 D 300 W 300 0
4  BankID_02 D 2000 W 30 D 200 W 300 1870
5  BankID_03 D 20 20
6  BankID_04 D 2000 2000
7
8  ## end of log.
9
```

Output without valgrind for seeing graceful output.

```
cbolat@cbolat:~/System-Programming/MIDTERM$ ./BankServer AdaBank
AdaBank is active...
Bank database loaded with 4 accounts...
Waiting for clients @ServerFIFO_Name...
- Received 3 clients from PIDClient25788
-- Teller PID25789 is active serving Client01...
-- Teller PID25790 is active serving Client02...
-- Teller PID25791 is active serving Client03...
Client01 deposited 300 credits... updating log...
Client02 withdraws 30 credits... operation not permitted...
Client03 deposited 2000 credits... updating log...

Waiting for clients @ServerFIFO_Name...
- Received 2 clients from PIDClient25820
-- Teller PID25821 is active serving Client01...
-- Teller PID25822 is active serving Client02...
Client01 withdraws 300 credits... operation not permitted...
Client02 deposited 20 credits... updating log...

Waiting for clients @ServerFIFO_Name...
- Received 5 clients from PIDClient25846
-- Teller PID25847 is active serving Client01...Welcome back Client01
-- Teller PID25848 is active serving Client02...
-- Teller PID25849 is active serving Client03...Welcome back Client03
-- Teller PID25850 is active serving Client04...Welcome back Client04
-- Teller PID25851 is active serving Client05...
Client01 withdraws 30 credits... updating log...
Client02 deposited 2000 credits... updating log...
Client03 deposited 200 credits... updating log...
Client04 withdraws 300 credits... updating log...
Client05 withdraws 20 credits... operation not permitted...

Waiting for clients @ServerFIFO_Name...
```

```
cbolat@cbolat:~/System-Programming/MIDTERM$ ./BankClient Client01.file
Reading Client01.file...
3 client to connect.. creating clients...
Connected to BankServer...
Client01 connected..depositing 300 credits
Client02 connected..withdrawing 30 credits
Client03 connected..depositing 2000 credits
..
Waiting for responses...
Client01 served.. account closed
Client02 something went WRONG :Account not found.
Client03: served.. BankID_06 new balance: 300
exiting...
cbolat@cbolat:~/System-Programming/MIDTERM$ ./BankClient Client02.file
Reading Client02.file...
2 client to connect.. creating clients...
Connected to BankServer...
Client01 connected..withdrawing 300 credits
Client02 connected..depositing 20 credits
..
Waiting for responses...
Client01 something went WRONG :Account is closed.
Client02 served.. account closed
exiting...
cbolat@cbolat:~/System-Programming/MIDTERM$ ./BankClient Client03.file
Reading Client03.file...
5 client to connect.. creating clients...
Connected to BankServer...
Client01 connected..withdrawing 30 credits
Client02 connected..depositing 2000 credits
Client03 connected..depositing 200 credits
Client04 connected..withdrawing 300 credits
Client05 connected..withdrawing 20 credits
..
Waiting for responses...
Client01: served.. BankID_08 new balance: 2000
Client02: served.. BankID_08 new balance: 2000
Client03: served.. BankID_02 new balance: 2040
Client04: served.. BankID_02 new balance: 1740
Client05 something went WRONG :Cannot withdraw from a new account.
exiting...
cbolat@cbolat:~/System-Programming/MIDTERM$
```

Client tries to connect but no bankserver running

```
cbolat@cbolat:~/System-Programming/MIDTERM$
```

```
cbolat@cbolat:~/System-Programming/MIDTERM$ valgrind ./BankClient Client01.file
==26077== Memcheck, a memory error detector
==26077== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==26077== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==26077== Command: ./BankClient Client01.file
==26077==
Reading Client01.file...
3 client to connect.. creating clients...
Cannot connect to ServerFIFO_Name...
exiting...
==26077==
==26077== HEAP SUMMARY:
==26077==     in use at exit: 0 bytes in 0 blocks
==26077==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==26077==
==26077== All heap blocks were freed -- no leaks are possible
==26077==
==26077== For lists of detected and suppressed errors, rerun with: -s
==26077== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cbolat@cbolat:~/System-Programming/MIDTERM$
```

Testing 20 Client properly

```
cbolat@cbolat:~/System-Programming/MIDTERM$ valgrind ./BankServer AdaBank
==26132== Memcheck, a memory error detector
==26132== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==26132== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==26132== Command: ./BankServer AdaBank
==26132==
AdaBank is active...
Bank database loaded with 8 accounts...
Waiting for clients @ServerFIFO_Name...
- Received 20 clients from PIDClient26165
-- Teller PID26175 is active serving Client01...
==26175==
==26175== HEAP SUMMARY:
==26175==     in use at exit: 0 bytes in 0 blocks
==26175==   total heap usage: 3 allocs, 3 frees, 11,803 bytes allocated
==26175==
==26175== All heap blocks were freed -- no leaks are possible
==26175==
==26175== For lists of detected and suppressed errors, rerun with: -s
==26175== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID26176 is active serving Client02...
-- Teller PID26177 is active serving Client03...
==26176==
==26176== HEAP SUMMARY:
==26176==     in use at exit: 0 bytes in 0 blocks
==26176==   total heap usage: 51 allocs, 51 frees, 169,243 bytes allocated
==26176==
==26176== All heap blocks were freed -- no leaks are possible
==26176==
==26176== For lists of detected and suppressed errors, rerun with: -s
==26176== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID26178 is active serving Client04...
==26177==
==26177== HEAP SUMMARY:
==26177==     in use at exit: 0 bytes in 0 blocks
==26177==   total heap usage: 52 allocs, 52 frees, 169,262 bytes allocated
==26177==
==26177== All heap blocks were freed -- no leaks are possible
==26177==
==26177== For lists of detected and suppressed errors, rerun with: -s
==26177== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID26179 is active serving Client05...Welcome back Client05
==26178==
==26178== HEAP SUMMARY:
==26178==     in use at exit: 0 bytes in 0 blocks
==26178==   total heap usage: 53 allocs, 53 frees, 169,281 bytes allocated
==26178==
==26178== All heap blocks were freed -- no leaks are possible
==26178==
==26178== For lists of detected and suppressed errors, rerun with: -s
==26178== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
bash: valgrind./BankClient: Böyle bir dosya ya da dizin yok
cbolat@cbolat:~/System-Programming/MIDTERM$ valgrind ./BankClient Client04.file
==26165== Memcheck, a memory error detector
==26165== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==26165== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==26165== Command: ./BankClient Client04.file
==26165==
Reading Client04.file...
20 client to connect.. creating clients...
Connected to BankServer...
Client01 connected..depositing 5000 credits
Client02 connected..withdrawing 1250 credits
Client03 connected..withdrawing 350 credits
Client04 connected..withdrawing 475 credits
Client05 connected..depositing 3200 credits
Client06 connected..withdrawing 650 credits
Client07 connected..depositing 180 credits
Client08 connected..withdrawing 1200 credits
Client09 connected..depositing 10000 credits
Client10 connected..withdrawing 2000 credits
Client11 connected..withdrawing 85 credits
Client12 connected..depositing 720 credits
Client13 connected..depositing 2800 credits
Client14 connected..depositing 450 credits
Client15 connected..withdrawing 580 credits
Client16 connected..withdrawing 125 credits
Client17 connected..withdrawing 320 credits
Client18 connected..withdrawing 3500 credits
Client19 connected..depositing 1850 credits
Client20 connected..withdrawing 750 credits
..
Waiting for responses...
Client01 served.. account closed
Client02 something went WRONG :Account is closed.
Client03 something went WRONG :Account is closed.
Client04 something went WRONG :Account is closed.
Client05: served.. BankID_02 new balance: 4940
Client06: served.. BankID_02 new balance: 4290
Client07 something went WRONG :Account is closed.
Client08: served.. BankID_02 new balance: 3090
Client09: served.. BankID_10 new balance: 10000
Client10 something went WRONG :Insufficient funds.
Client11 something went WRONG :Account is closed.
Client12: served.. BankID_02 new balance: 3810
Client13 something went WRONG :Account is closed.
Client14: served.. BankID_03 new balance: 470
Client15 something went WRONG :Account is closed.
Client16: served.. BankID_02 new balance: 3685
Client17 something went WRONG :Account is closed.
Client18 something went WRONG :Insufficient funds.
Client19: served.. BankID_02 new balance: 5535
```

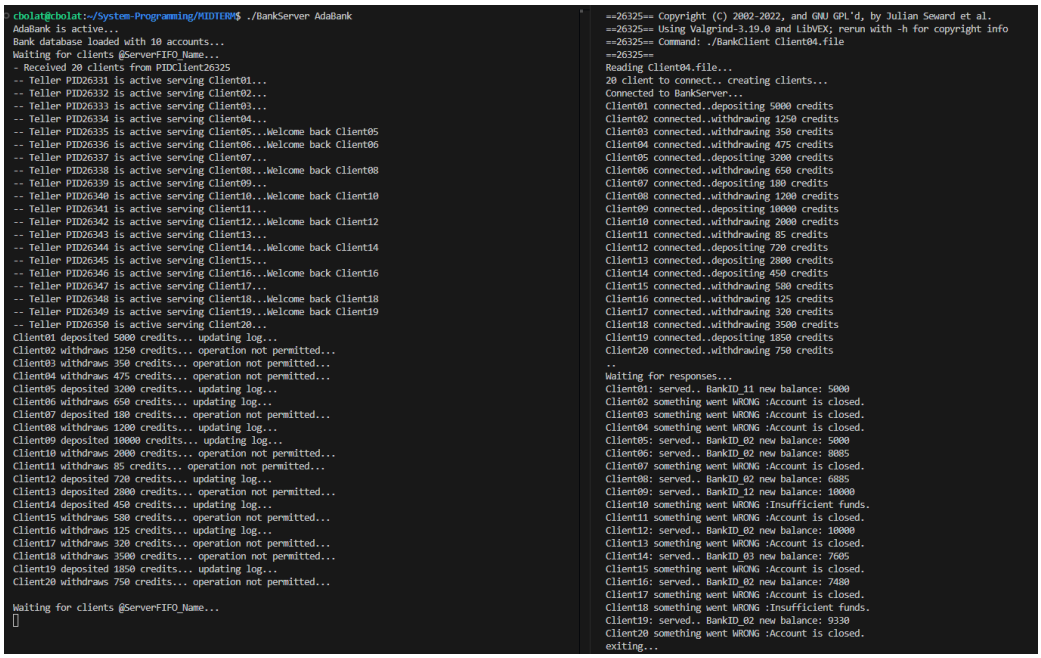
```
-- Teller PID26179 is active serving Client05...Welcome back Client05
==26178==
==26178== HEAP SUMMARY:
==26178==     in use at exit: 0 bytes in 0 blocks
==26178==   total heap usage: 53 allocs, 53 frees, 169,281 bytes allocated
==26178==
==26178== All heap blocks were freed -- no leaks are possible
==26178==
==26178== For lists of detected and suppressed errors, rerun with: -s
==26178== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID26180 is active serving Client06...Welcome back Client06
==26179==
==26179== HEAP SUMMARY:
==26179==     in use at exit: 0 bytes in 0 blocks
==26179==   total heap usage: 54 allocs, 54 frees, 169,300 bytes allocated
==26179==
==26179== All heap blocks were freed -- no leaks are possible
==26179==
==26179== For lists of detected and suppressed errors, rerun with: -s
==26179== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID26181 is active serving Client07...
==26180==
==26180== HEAP SUMMARY:
==26180==     in use at exit: 0 bytes in 0 blocks
==26180==   total heap usage: 94 allocs, 94 frees, 324,982 bytes allocated
==26180==
==26180== All heap blocks were freed -- no leaks are possible
==26180==
==26180== For lists of detected and suppressed errors, rerun with: -s
==26180== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID26182 is active serving Client08...Welcome back Client08
==26181==
==26181== HEAP SUMMARY:
==26181==     in use at exit: 0 bytes in 0 blocks
==26181==   total heap usage: 135 allocs, 135 frees, 484,760 bytes allocated
==26181==
==26181== All heap blocks were freed -- no leaks are possible
==26181==
==26181== For lists of detected and suppressed errors, rerun with: -s
==26181== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==26182==
==26182== HEAP SUMMARY:
==26182==     in use at exit: 0 bytes in 0 blocks
==26182==   total heap usage: 136 allocs, 136 frees, 484,779 bytes allocated
==26182==
==26182== All heap blocks were freed -- no leaks are possible
==26182==
==26182== For lists of detected and suppressed errors, rerun with: -s
==26182== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID26183 is active serving Client09...
-- Teller PID26184 is active serving Client10...Welcome back Client10
```

```
Client03 connected..withdrawing 350 credits
Client04 connected..withdrawing 475 credits
Client05 connected..depositing 3200 credits
Client06 connected..withdrawing 650 credits
Client07 connected..depositing 180 credits
Client08 connected..withdrawing 1200 credits
Client09 connected..depositing 10000 credits
Client10 connected..withdrawing 2000 credits
Client11 connected..withdrawing 85 credits
Client12 connected..depositing 720 credits
Client13 connected..depositing 2800 credits
Client14 connected..depositing 450 credits
Client15 connected..withdrawing 580 credits
Client16 connected..withdrawing 125 credits
Client17 connected..withdrawing 320 credits
Client18 connected..withdrawing 3500 credits
Client19 connected..depositing 1850 credits
Client20 connected..withdrawing 750 credits
..
Waiting for responses...
Client01 served.. account closed
Client02 something went WRONG :Account is closed.
Client03 something went WRONG :Account is closed.
Client04 something went WRONG :Account is closed.
Client05: served.. BankID_02 new balance: 4940
Client06: served.. BankID_02 new balance: 4290
Client07 something went WRONG :Account is closed.
Client08: served.. BankID_02 new balance: 3090
Client09: served.. BankID_10 new balance: 10000
Client10 something went WRONG :Insufficient funds.
Client11 something went WRONG :Account is closed.
Client12: served.. BankID_02 new balance: 3810
Client13 something went WRONG :Account is closed.
Client14: served.. BankID_03 new balance: 470
Client15 something went WRONG :Account is closed.
Client16: served.. BankID_02 new balance: 3685
Client17 something went WRONG :Account is closed.
Client18 something went WRONG :Insufficient funds.
Client19: served.. BankID_02 new balance: 5535
Client20 something went WRONG :Account is closed.
exiting...
==26165==
==26165== HEAP SUMMARY:
==26165==     in use at exit: 0 bytes in 0 blocks
==26165==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==26165==
==26165== All heap blocks were freed -- no leaks are possible
==26165==
==26165== For lists of detected and suppressed errors, rerun with: -s
==26165== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cbolat@cbolat:~/System-Programming/MIDTERM$
```

Valgrind output is shows there is no leaks & data races on no purposes

```
==26193==     in use at exit: 0 bytes in 0 blocks
==26193==   total heap usage: 370 allocs, 370 frees, 1,377,991 bytes allocated
==26193==
==26193== All heap blocks were freed -- no leaks are possible
==26193==
==26193== For lists of detected and suppressed errors, rerun with: -s
==26193== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==26194==
==26194== HEAP SUMMARY:
==26194==     in use at exit: 0 bytes in 0 blocks
==26194==   total heap usage: 419 allocs, 419 frees, 1,570,537 bytes allocated
==26194==
==26194== All heap blocks were freed -- no leaks are possible
==26194==
==26194== For lists of detected and suppressed errors, rerun with: -s
==26194== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Client01 deposited 5000 credits... updating log...
Client02 withdraws 1250 credits... operation not permitted...
Client03 withdraws 350 credits... operation not permitted...
Client04 withdraws 475 credits... operation not permitted...
Client05 deposited 3200 credits... updating log...
Client06 withdraws 650 credits... updating log...
Client07 deposited 180 credits... operation not permitted...
Client08 withdraws 1200 credits... updating log...
Client09 deposited 10000 credits... updating log...
Client10 withdraws 2000 credits... operation not permitted...
Client11 withdraws 85 credits... operation not permitted...
Client12 deposited 720 credits... updating log...
Client13 deposited 2800 credits... operation not permitted...
Client14 deposited 450 credits... updating log...
Client15 withdraws 580 credits... operation not permitted...
Client16 withdraws 125 credits... updating log...
Client17 withdraws 320 credits... operation not permitted...
Client18 withdraws 3500 credits... operation not permitted...
Client19 deposited 1850 credits... updating log...
Client20 withdraws 750 credits... operation not permitted...

Waiting for clients @ServerFIFO_Name...
^CSignal received closing active Tellers...
Removing ServerFIFO... Updating log file...
AdaBank says "Bye"...
==26132==
==26132== HEAP SUMMARY:
==26132==     in use at exit: 0 bytes in 0 blocks
==26132==   total heap usage: 467 allocs, 467 frees, 1,763,064 bytes allocated
==26132==
==26132== All heap blocks were freed -- no leaks are possible
==26132==
==26132== For lists of detected and suppressed errors, rerun with: -s
==26132== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cbolat@cbolat:~/System-Programming/MIDTERM$
```

(Without valgrind for graceful output)



```
cholat@cholat:~/System-Programming/MIDTERM$ ./BankServer AdaBank
AdaBank is active...
Bank database loaded with 10 accounts...
Waiting for clients @ServerFIFO_Name...
 - Received 20 clients from PIDClient26325
 -- Teller PID26331 is active serving Client01...
 -- Teller PID26332 is active serving Client02...
 -- Teller PID26333 is active serving Client03...
 -- Teller PID26334 is active serving Client04...
 -- Teller PID26335 is active serving Client05...Welcome back Client05
 -- Teller PID26336 is active serving Client06...Welcome back Client06
 -- Teller PID26337 is active serving Client07...
 -- Teller PID26338 is active serving Client08...Welcome back Client08
 -- Teller PID26339 is active serving Client09...
 -- Teller PID26340 is active serving Client10...Welcome back Client10
 -- Teller PID26341 is active serving Client11...
 -- Teller PID26342 is active serving Client12...Welcome back Client12
 -- Teller PID26343 is active serving Client13...
 -- Teller PID26344 is active serving Client14...Welcome back Client14
 -- Teller PID26345 is active serving Client15...
 -- Teller PID26346 is active serving Client16...Welcome back Client16
 -- Teller PID26347 is active serving Client17...
 -- Teller PID26348 is active serving Client18...Welcome back Client18
 -- Teller PID26349 is active serving Client19...Welcome back Client19
 -- Teller PID26350 is active serving Client20...
Client01 deposited 5000 credits... updating log...
Client02 withdraws 1250 credits... operation not permitted...
Client03 withdraws 350 credits... operation not permitted...
Client04 withdraws 475 credits... operation not permitted...
Client05 deposited 3200 credits... updating log...
Client06 withdraws 650 credits... updating log...
Client07 deposited 180 credits... operation not permitted...
Client08 withdraws 1200 credits... updating log...
Client09 deposited 10000 credits... updating log...
Client10 withdraws 2000 credits... operation not permitted...
Client11 withdraws 85 credits... operation not permitted...
Client12 deposited 720 credits... updating log...
Client13 deposited 2800 credits... operation not permitted...
Client14 deposited 450 credits... updating log...
Client15 withdraws 580 credits... operation not permitted...
Client16 withdraws 125 credits... updating log...
Client17 withdraws 320 credits... operation not permitted...
Client18 withdraws 3500 credits... operation not permitted...
Client19 deposited 1850 credits... updating log...
Client20 withdraws 750 credits... operation not permitted...

Waiting for clients @ServerFIFO_Name...
```

```
==26325== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==26325== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==26325== Command: ./BankClient Client04.file
==26325==
Reading Client04.file...
20 client to connect.. creating clients...
Connected to BankServer...
Client01 connected..depositing 5000 credits
Client02 connected..withdrawing 1250 credits
Client03 connected..withdrawing 350 credits
Client04 connected..withdrawing 475 credits
Client05 connected..depositing 3200 credits
Client06 connected..withdrawing 650 credits
Client07 connected..depositing 180 credits
Client08 connected..withdrawing 1200 credits
Client09 connected..depositing 10000 credits
Client10 connected..withdrawing 2000 credits
Client11 connected..withdrawing 85 credits
Client12 connected..depositing 720 credits
Client13 connected..depositing 2800 credits
Client14 connected..depositing 450 credits
Client15 connected..withdrawing 580 credits
Client16 connected..withdrawing 125 credits
Client17 connected..withdrawing 320 credits
Client18 connected..withdrawing 3500 credits
Client19 connected..depositing 1850 credits
Client20 connected..withdrawing 750 credits
..
Waiting for responses...
Client01: served.. BankID_11 new balance: 5000
Client02 something went WRONG :Account is closed.
Client03 something went WRONG :Account is closed.
Client04 something went WRONG :Account is closed.
Client05: served.. BankID_02 new balance: 5000
Client06: served.. BankID_02 new balance: 8085
Client07 something went WRONG :Account is closed.
Client08: served.. BankID_02 new balance: 6885
Client09: served.. BankID_12 new balance: 10000
Client10 something went WRONG :Insufficient funds.
Client11 something went WRONG :Account is closed.
Client12: served.. BankID_02 new balance: 10000
Client13 something went WRONG :Account is closed.
Client14: served.. BankID_03 new balance: 7605
Client15 something went WRONG :Account is closed.
Client16: served.. BankID_02 new balance: 7480
Client17 something went WRONG :Account is closed.
Client18 something went WRONG :Insufficient funds.
Client19: served.. BankID_02 new balance: 9330
Client20 something went WRONG :Account is closed.
exiting...
```

LOG FILE

MIDTERM > ≡ AdaBank.bankLog

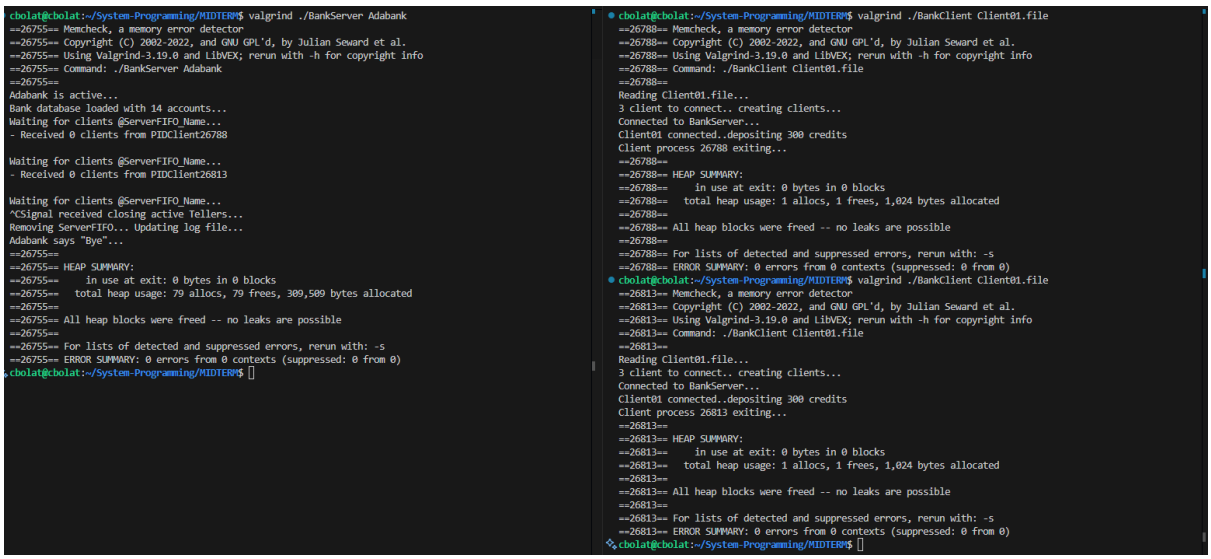
```
# Adabank Log file updated @Sun Apr 27 22:31:32 2025

# BankID_01 D 300 W 300 0
BankID_02 D 2000 W 30 D 200 W 300 W 30 D 200 W 300 D 3200 W 650 W 1200 D 720 W 125 D 1850 D 3200 W 650 W 1200 D 720 W 125 D 1850 9330
BankID_03 D 20 D 450 D 450 920
BankID_04 D 2000 2000
BankID_05 D 300 300
BankID_06 D 2000 2000
BankID_07 D 20 20
BankID_08 D 2000 2000
BankID_09 D 5000 5000
BankID_10 D 10000 10000
BankID_11 D 5000 5000
BankID_12 D 10000 10000

## end of log.
```

Client04.file

```
N deposit 5000
BankID_01 withdraw 1250
BankID_01 withdraw 350
BankID_01 withdraw 475
BankID_02 deposit 3200
BankID_02 withdraw 650
BankID_01 deposit 180
BankID_02 withdraw 1200
N deposit 10000
BankID_03 withdraw 2000
BankID_01 withdraw 85
BankID_02 deposit 720
BankID_01 deposit 2800
BankID_03 deposit 450
BankID_01 withdraw 580
BankID_02 withdraw 125
BankID_01 withdraw 320
BankID_03 withdraw 3500
BankID_02 deposit 1850
BankID_01 withdraw 750
```

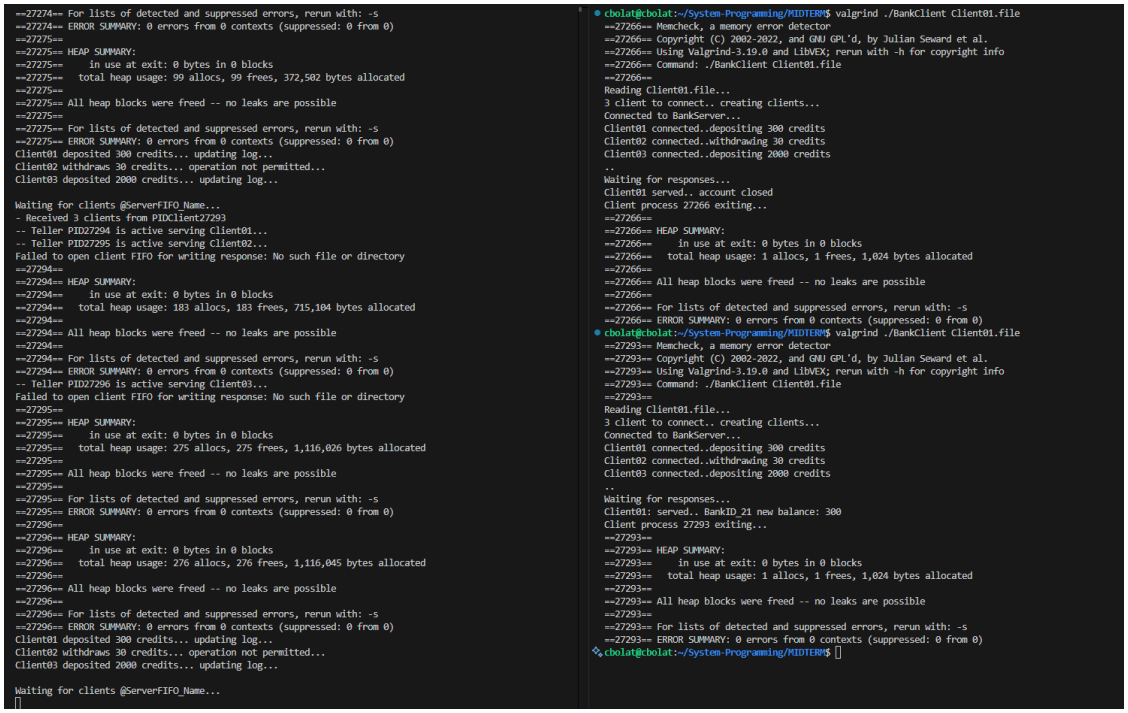
ALL scenerios are commentarized so searching // test case X is ease to find where I changed.

Test Case 1: Client Close while writing ServerFIFO

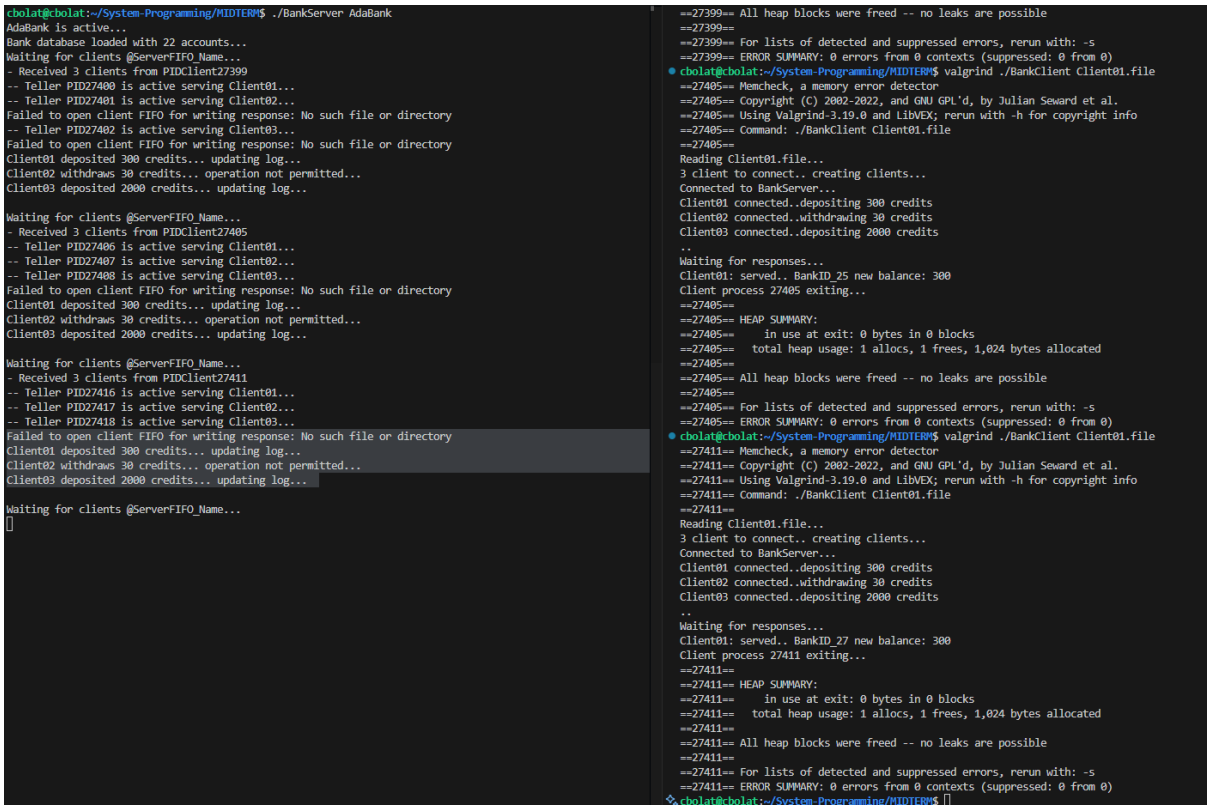


No Deadlocks & Memory Leaks

Test Case 2: Client gets signal after reading 1 response

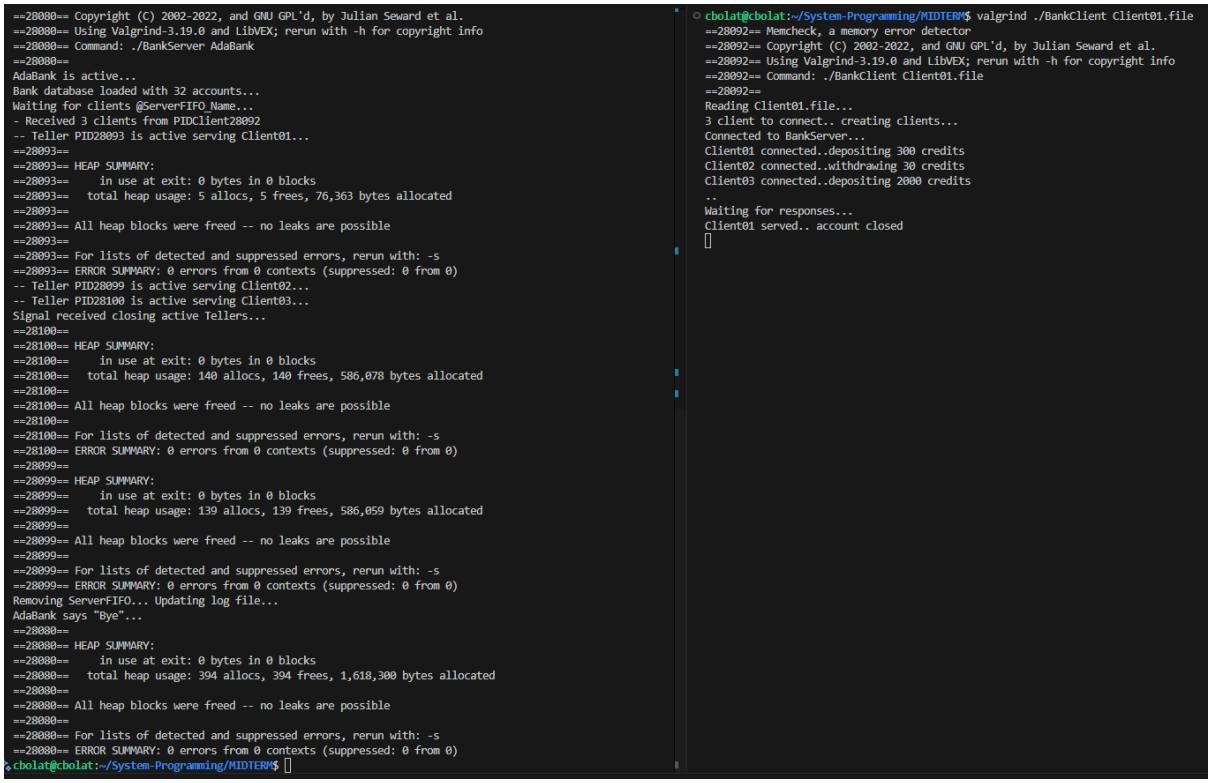


Without Valgrind for graceful output.



It didn’t deadlock. Updates log file properly because request is received succesfully but response delivery is failed.

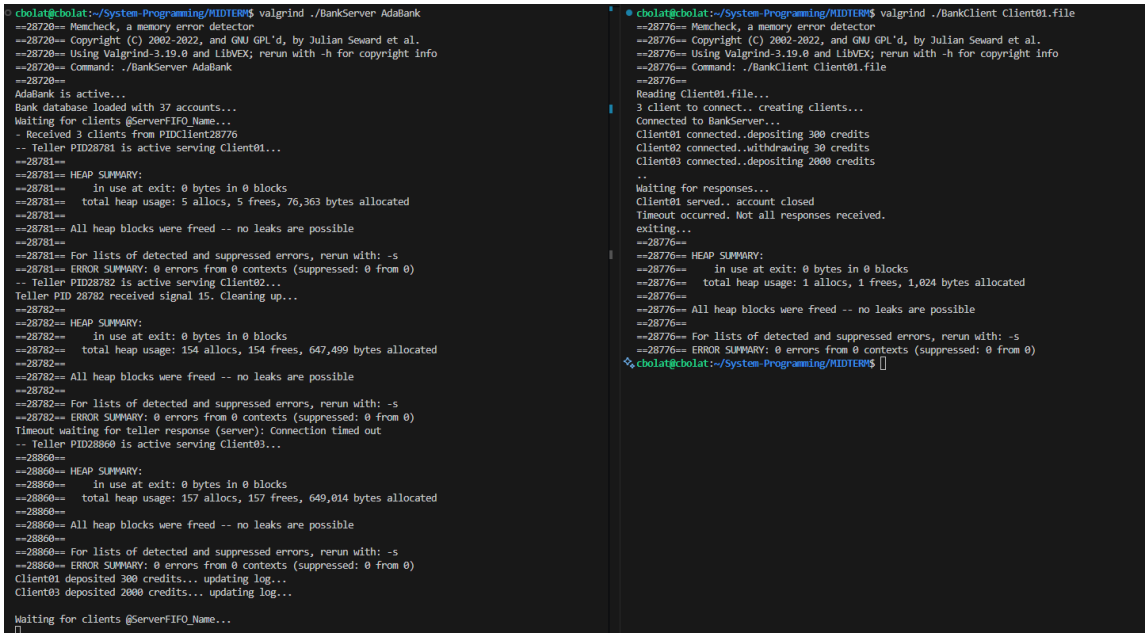
Test Case 3: Server gets SIGTERM while tellers sending the responses.



```
==28080== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==28080== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==28080== Command: ./BankServer AdaBank
==28080==
AdaBank is active...
Bank database loaded with 32 accounts...
Waiting for clients @ServerFIFO_Name...
- Received 3 clients from PIDClient28092
-- Teller PID28093 is active serving Client01...
==28093==
==28093== HEAP SUMMARY:
==28093==     in use at exit: 0 bytes in 0 blocks
==28093==   total heap usage: 5 allocs, 5 frees, 76,363 bytes allocated
==28093==
==28093== All heap blocks were freed -- no leaks are possible
==28093==
==28093== For lists of detected and suppressed errors, rerun with: -s
==28093== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID28099 is active serving Client02...
-- Teller PID28100 is active serving Client03...
Signal received closing active Tellers...
==28100==
==28100== HEAP SUMMARY:
==28100==     in use at exit: 0 bytes in 0 blocks
==28100==   total heap usage: 140 allocs, 140 frees, 586,078 bytes allocated
==28100==
==28100== All heap blocks were freed -- no leaks are possible
==28100==
==28100== For lists of detected and suppressed errors, rerun with: -s
==28100== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==28099==
==28099== HEAP SUMMARY:
==28099==     in use at exit: 0 bytes in 0 blocks
==28099==   total heap usage: 139 allocs, 139 frees, 586,059 bytes allocated
==28099==
==28099== All heap blocks were freed -- no leaks are possible
==28099==
==28099== For lists of detected and suppressed errors, rerun with: -s
==28099== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Removing ServerFIFO... Updating log file...
AdaBank says "Bye"...
==28080==
==28080== HEAP SUMMARY:
==28080==     in use at exit: 0 bytes in 0 blocks
==28080==   total heap usage: 394 allocs, 394 frees, 1,618,300 bytes allocated
==28080==
==28080== All heap blocks were freed -- no leaks are possible
==28080==
==28080== For lists of detected and suppressed errors, rerun with: -s
==28080== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cbolat@cbolat:~/System-Programming/MIDTERM$
```

```
cbolat@cbolat:~/System-Programming/MIDTERM$ valgrind ./BankClient Client01.file
==28092== Memcheck, a memory error detector
==28092== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==28092== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==28092== Command: ./BankClient Client01.file
==28092==
Reading Client01.file...
3 client to connect.. creating clients...
Connected to BankServer...
Client01 connected..depositing 300 credits
Client02 connected..withdrawing 30 credits
Client03 connected..depositing 2000 credits
..
Waiting for responses...
Client01 served.. account closed
```

Test Case 4: Tellers gets SIGTERM while sending the response.



```
cbolat@cbolat:~/System-Programming/MIDTERM$ valgrind ./BankServer AdaBank
==28720== Memcheck, a memory error detector
==28720== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==28720== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==28720== Command: ./BankServer AdaBank
==28720==
AdaBank is active...
Bank database loaded with 37 accounts...
Waiting for clients @ServerFIFO_Name...
- Received 3 clients from PIDClient28776
-- Teller PID28781 is active serving Client01...
==28781==
==28781== HEAP SUMMARY:
==28781==     in use at exit: 0 bytes in 0 blocks
==28781==   total heap usage: 5 allocs, 5 frees, 76,363 bytes allocated
==28781==
==28781== All heap blocks were freed -- no leaks are possible
==28781==
==28781== For lists of detected and suppressed errors, rerun with: -s
==28781== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-- Teller PID28782 is active serving Client02...
Teller PID 28782 received signal 15. Cleaning up...
==28782==
==28782== HEAP SUMMARY:
==28782==     in use at exit: 0 bytes in 0 blocks
==28782==   total heap usage: 154 allocs, 154 frees, 647,499 bytes allocated
==28782==
==28782== All heap blocks were freed -- no leaks are possible
==28782==
==28782== For lists of detected and suppressed errors, rerun with: -s
==28782== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Timeout waiting for teller response (server): Connection timed out
-- Teller PID28860 is active serving Client03...
==28860==
==28860== HEAP SUMMARY:
==28860==     in use at exit: 0 bytes in 0 blocks
==28860==   total heap usage: 157 allocs, 157 frees, 649,014 bytes allocated
==28860==
==28860== All heap blocks were freed -- no leaks are possible
==28860==
==28860== For lists of detected and suppressed errors, rerun with: -s
==28860== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Client01 deposited 300 credits... updating log...
Client03 deposited 2000 credits... updating log...

Waiting for clients @ServerFIFO_Name...
```

```
cbolat@cbolat:~/System-Programming/MIDTERM$ valgrind ./BankClient Client01.file
==28776== Memcheck, a memory error detector
==28776== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==28776== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==28776== Command: ./BankClient Client01.file
==28776==
Reading Client01.file...
3 client to connect.. creating clients...
Connected to BankServer...
Client01 connected..depositing 300 credits
Client02 connected..withdrawing 30 credits
Client03 connected..depositing 2000 credits
..
Waiting for responses...
Client01 served.. account closed
Timeout occurred. Not all responses received.
exiting...
==28776==
==28776== HEAP SUMMARY:
==28776==     in use at exit: 0 bytes in 0 blocks
==28776==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==28776==
==28776== All heap blocks were freed -- no leaks are possible
==28776==
==28776== For lists of detected and suppressed errors, rerun with: -s
==28776== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cbolat@cbolat:~/System-Programming/MIDTERM$
```

Project Development Steps:

4.1. Initial Setup

First, a basic project structure was established, separating client, server, and shared components into different modules. Common header files were prepared to define shared types such as ClientRequest, ServerResponse, and BankAccountLog. This allowed consistent communication between processes and simplified data management.

4.2. FIFO Communication Design

Named FIFOs (First-In-First-Out special files) were used to facilitate initial communication between clients and the server. A global server FIFO (ServerFIFO) was created, where clients would send their own FIFO names. Upon receiving a client's FIFO name, the server opened the corresponding FIFO to read transaction requests.

4.3. Shared Memory Architecture

Shared memory segments were utilized for detailed communication between the server and tellers. For each client request, the server created a unique shared memory segment, transferring the request information and account state. Semaphores were employed to coordinate the exchange of information, ensuring that the teller process would not proceed until the server had fully prepared the shared memory.

4.4. Teller Process Creation

For each client request, a teller process was forked using the Teller function. Depending on the transaction type (deposit or withdrawal), the server triggered the appropriate teller function (deposit() or withdraw()), passing the shared memory reference. Teller processes were responsible for validating the request, updating the shared memory, and sending a response to the client's FIFO.

4.5. Synchronization and Timeout Handling

Semaphores were critical for synchronizing the server and teller processes. sem_request_ready and sem_response_ready were used to notify when a request or response was ready. A timeout mechanism was also implemented using sem_timedwait to prevent the server from indefinitely waiting for a teller that might become unresponsive.

4.6. Database and Log Management

A dynamic in-memory database was designed to maintain all active bank accounts. Functions such as init_database, find_account, ensure_capacity, and write_log were implemented to manage account operations. After each successful transaction, the server updated the in-memory database and immediately flushed changes to a persistent log file to ensure data durability.

4.7. Signal Handling and Cleanup

Signal handlers were carefully designed to handle termination signals (SIGINT, SIGTERM) both for the server and the teller processes. Resources like FIFOs, shared memory segments, and file descriptors were properly released during termination to avoid memory leaks, dangling FIFOs, or orphaned shared memory regions.

4.1. Initial Setup

```c
#define SHM_NAME_TEMPLATE "/shm_bankreq_%d"
#define SHM_SIZE sizeof(SharedMemoryData)

typedef struct {
    ClientRequest request;
    ServerResponse response;

    BankAccountLog account;

    sem_t sem_request_ready;    // server → teller
    sem_t sem_response_ready;   // teller → server
} SharedMemoryData;
```

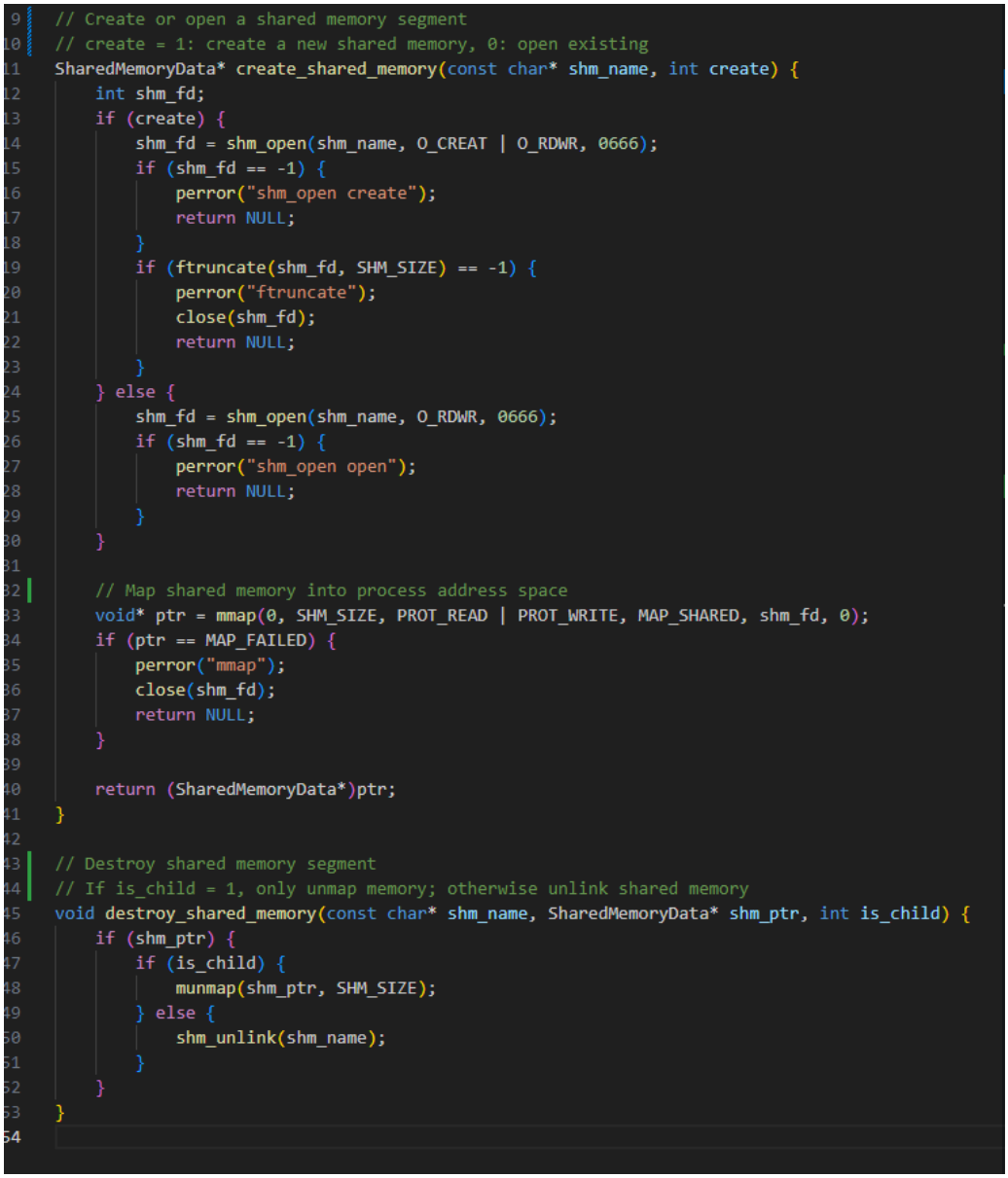
```c
#define MAX_LINE_LEN 256

// Transaction types
typedef enum {
    TRANSACTION_DEPOSIT,
    TRANSACTION_WITHDRAW
} TransactionType;

// Request structure: Client -> Teller
typedef struct {
    pid_t client_pid;                   // PID of the client process
    TransactionType type;               // Deposit or Withdraw
    int amount;                         // Transaction amount
    char bank_id[MAX_BANK_ID_LEN];       // Bank account ID (e.g., "BankID_01")
    char client_fifo[MAX_FIFO_NAME_LEN];  // FIFO name for server response
    int client_id;
    int new_account;                    // Request to open a new account (1: yes, 0: no)
} ClientRequest;

// Response structure: Teller -> Client
typedef struct {
    int success;                        // 1: success, 0: failure
    int new_balance;                    // Updated account balance
    char message[128];                  // Status message
    char bank_id[MAX_BANK_ID_LEN];       // Assigned bank ID if a new account is created
} ServerResponse;

// Structure for log entries (for advanced use)
typedef struct {
    char bank_id[MAX_BANK_ID_LEN];
    int current_balance;
    int is_closed;                      // 1 if the account is closed
    int is_invalid_account;             // 1 if the account is invalid
    char operations[MAX_TRANSACTIONS][8]; // Operations like "D 300", "W 100"
    int op_count;
    pid_t teller_pid;                   // PID of the Teller handling the request
} BankAccountLog;

#endif // COMMON_H
```

4.2. FIFO Communication Design

```
}

// Open a FIFO for writing
int open_fifo_write(const char* fifo_name) {
    int fd = open(fifo_name, O_WRONLY);
    if (fd == -1) {
        return -1;
    }
    return fd;
}

// Open a FIFO for reading
int open_fifo_read(const char* fifo_name) {
    int fd = open(fifo_name, O_RDONLY);
    if (fd == -1) {
        perror("open_fifo_read");
    }
    return fd;
}

// Generate a client FIFO name based on the process PID
void generate_client_fifo_name(char* buffer, pid_t pid) {
    snprintf(buffer, 64, "./ClientFIFO_%d", pid);
}
```

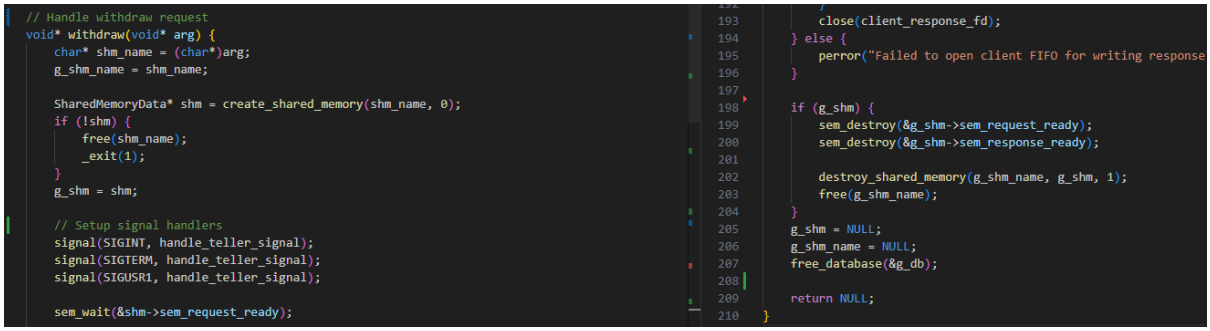
```
// Create a FIFO; remove it first if it already exists
int create_fifo(const char* fifo_name) {
    if (access(fifo_name, F_OK) == 0) {
        if (unlink(fifo_name) == -1) {
            perror("unlink");
            return -1;
        }
    }
    if (mkfifo(fifo_name, 0666) == -1) {
        if (errno != EEXIST) {
            perror("mkfifo");
            return -1;
        }
    }
    return 0;
}

// Remove a FIFO
int remove_fifo(const char* fifo_name) {
    if (unlink(fifo_name) == -1) {
        perror("unlink");
        return -1;
    }
    return 0;
}
```

4.3. Shared Memory Architecture



```c
// Create or open a shared memory segment
// create = 1: create a new shared memory, 0: open existing
SharedMemoryData* create_shared_memory(const char* shm_name, int create) {
    int shm_fd;
    if (create) {
        shm_fd = shm_open(shm_name, O_CREAT | O_RDWR, 0666);
        if (shm_fd == -1) {
            perror("shm_open create");
            return NULL;
        }
        if (ftruncate(shm_fd, SHM_SIZE) == -1) {
            perror("ftruncate");
            close(shm_fd);
            return NULL;
        }
    } else {
        shm_fd = shm_open(shm_name, O_RDWR, 0666);
        if (shm_fd == -1) {
            perror("shm_open open");
            return NULL;
        }
    }

    // Map shared memory into process address space
    void* ptr = mmap(0, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (ptr == MAP_FAILED) {
        perror("mmap");
        close(shm_fd);
        return NULL;
    }

    return (SharedMemoryData*)ptr;
}

// Destroy shared memory segment
// If is_child = 1, only unmap memory; otherwise unlink shared memory
void destroy_shared_memory(const char* shm_name, SharedMemoryData* shm_ptr, int is_child) {
    if (shm_ptr) {
        if (is_child) {
            munmap(shm_ptr, SHM_SIZE);
        } else {
            shm_unlink(shm_name);
        }
    }
}
```

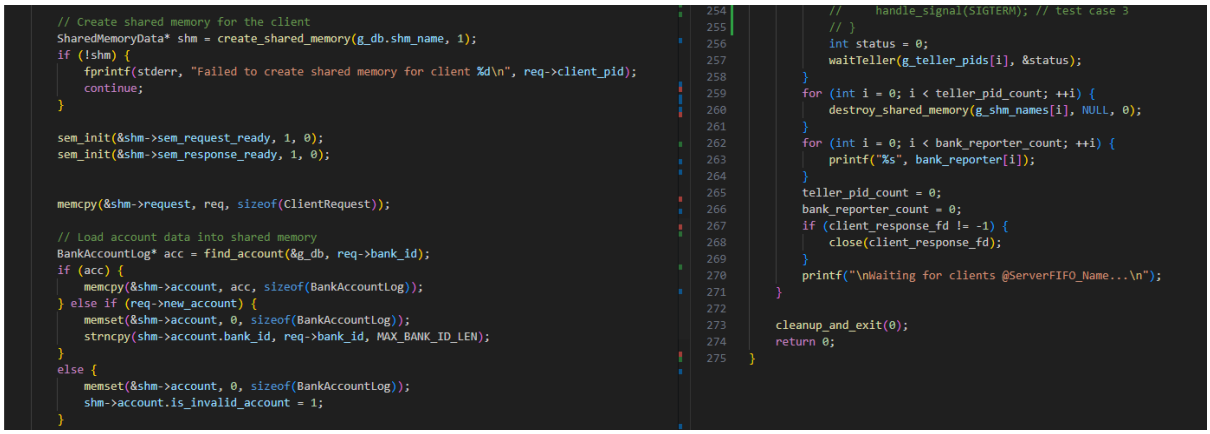
Start & end of teller process



```c
// Handle withdraw request
void* withdraw(void* arg) {
    char* shm_name = (char*)arg;
    g_shm_name = shm_name;

    SharedMemoryData* shm = create_shared_memory(shm_name, 0);
    if (!shm) {
        free(shm_name);
        _exit(1);
    }
    g_shm = shm;

    // Setup signal handlers
    signal(SIGINT, handle_teller_signal);
    signal(SIGTERM, handle_teller_signal);
    signal(SIGUSR1, handle_teller_signal);

    sem_wait(&shm->sem_request_ready);
```

```c
        close(client_response_fd);
    } else {
        perror("Failed to open client FIFO for writing response
    }

    if (g_shm) {
        sem_destroy(&g_shm->sem_request_ready);
        sem_destroy(&g_shm->sem_response_ready);

        destroy_shared_memory(g_shm_name, g_shm, 1);
        free(g_shm_name);
    }
    g_shm = NULL;
    g_shm_name = NULL;
    free_database(&g_db);

    return NULL;
}
```

Start & end of shared memory usage



```c
    // Create shared memory for the client
    SharedMemoryData* shm = create_shared_memory(g_db.shm_name, 1);
    if (!shm) {
        fprintf(stderr, "Failed to create shared memory for client %d\n", req->client_pid);
        continue;
    }

    sem_init(&shm->sem_request_ready, 1, 0);
    sem_init(&shm->sem_response_ready, 1, 0);


    memcpy(&shm->request, req, sizeof(ClientRequest));

    // Load account data into shared memory
    BankAccountLog* acc = find_account(&g_db, req->bank_id);
    if (acc) {
        memcpy(&shm->account, acc, sizeof(BankAccountLog));
    } else if (req->new_account) {
        memset(&shm->account, 0, sizeof(BankAccountLog));
        strncpy(shm->account.bank_id, req->bank_id, MAX_BANK_ID_LEN);
    }
    else {
        memset(&shm->account, 0, sizeof(BankAccountLog));
        shm->account.is_invalid_account = 1;
    }
```

```c
            //     handle_signal(SIGTERM); // test case 3
            // }
            int status = 0;
            waitTeller(g_teller_pids[i], &status);
        }
        for (int i = 0; i < teller_pid_count; ++i) {
            destroy_shared_memory(g_shm_names[i], NULL, 0);
        }
        for (int i = 0; i < bank_reporter_count; ++i) {
            printf("%s", bank_reporter[i]);
        }
        teller_pid_count = 0;
        bank_reporter_count = 0;
        if (client_response_fd != -1) {
            close(client_response_fd);
        }
        printf("\nWaiting for clients @ServerFIFO_Name...\n");
    }

    cleanup_and_exit(0);
    return 0;
}
```

4.4. Teller Process Creation

```c
pid_t pid;
if (req->type == TRANSACTION_DEPOSIT) {
    pid = Teller(deposit, strdup(g_db.shm_name));
} else if (req->type == TRANSACTION_WITHDRAW) {
    pid = Teller(withdraw, strdup(g_db.shm_name));
} else {
    fprintf(stderr, "Invalid transaction type.\n");
    sem_destroy(&shm->sem_request_ready);
    sem_destroy(&shm->sem_response_ready);
    destroy_shared_memory(g_db.shm_name, shm, 0);
    continue;
}
if (pid == -1) {
    fprintf(stderr, "Failed to create teller process.\n");
    sem_destroy(&shm->sem_request_ready);
    sem_destroy(&shm->sem_response_ready);
    destroy_shared_memory(g_db.shm_name, shm, 0);
    continue;
}

strcpy(g_shm_names[teller_pid_count], g_db.shm_name);
g_teller_pids[teller_pid_count++] = pid;
```

```c
pid_t Teller(void* (*func)(void*), void* arg) {
    pid_t pid = fork();
    if (pid < 0) {
        perror("Teller: fork failed");
        return -1;
    }
    if (pid == 0) {
        // Child process: call the function and exit
        func(arg);
        exit(EXIT_SUCCESS);
    } else {
        free(arg); // Parent process: free the argument
    }

    // Parent returns the child's PID
    return pid;
}

// Wait for a teller process to terminate
int waitTeller(pid_t pid, int* status) {
    return waitpid(pid, status, 0);
}
```

4.5. Synchronization and Timeout Handling

The communication flow is structured as follows:

- **Client → Main Server (via ServerFIFO)**:
  The client initiates communication by sending the name of its ClientFIFO to the main server through the ServerFIFO. This informs the main server that a client wants to establish a session.
- **Client → ClientFIFO**:
  After notifying the server, the client writes its transaction requests (deposit, withdraw) into its own ClientFIFO. The requests are not sent through ServerFIFO but through the ClientFIFO created specifically for this session.
- **Main Server → ClientFIFO**:
  The main server, upon receiving the ClientFIFO name, opens the corresponding ClientFIFO for reading. It continuously reads transaction requests from the ClientFIFO written by the client.
- **Main Server → Teller (via Shared Memory)**:
  Each transaction request read by the server is placed into a shared memory segment associated with the client. The server then signals the appropriate teller process by posting the sem_request_ready semaphore to indicate that a request is ready for processing.
- **Teller → Main Server (via Shared Memory)**:
  The teller process waits on the sem_request_ready semaphore. Once it is posted, the teller reads the request from shared memory, processes the transaction logic (e.g., balance update, account validation), and writes the result back into the shared memory. It then signals the server by posting the sem_response_ready semaphore.

- **Teller → Client (via ClientFIFO)**:
  After processing the request, the teller sends the transaction result directly back to the client by writing into the ClientFIFO. The main server does not send any transaction result directly to the client; it is entirely handled by the teller.

4.6. Timeout and Signal Handling

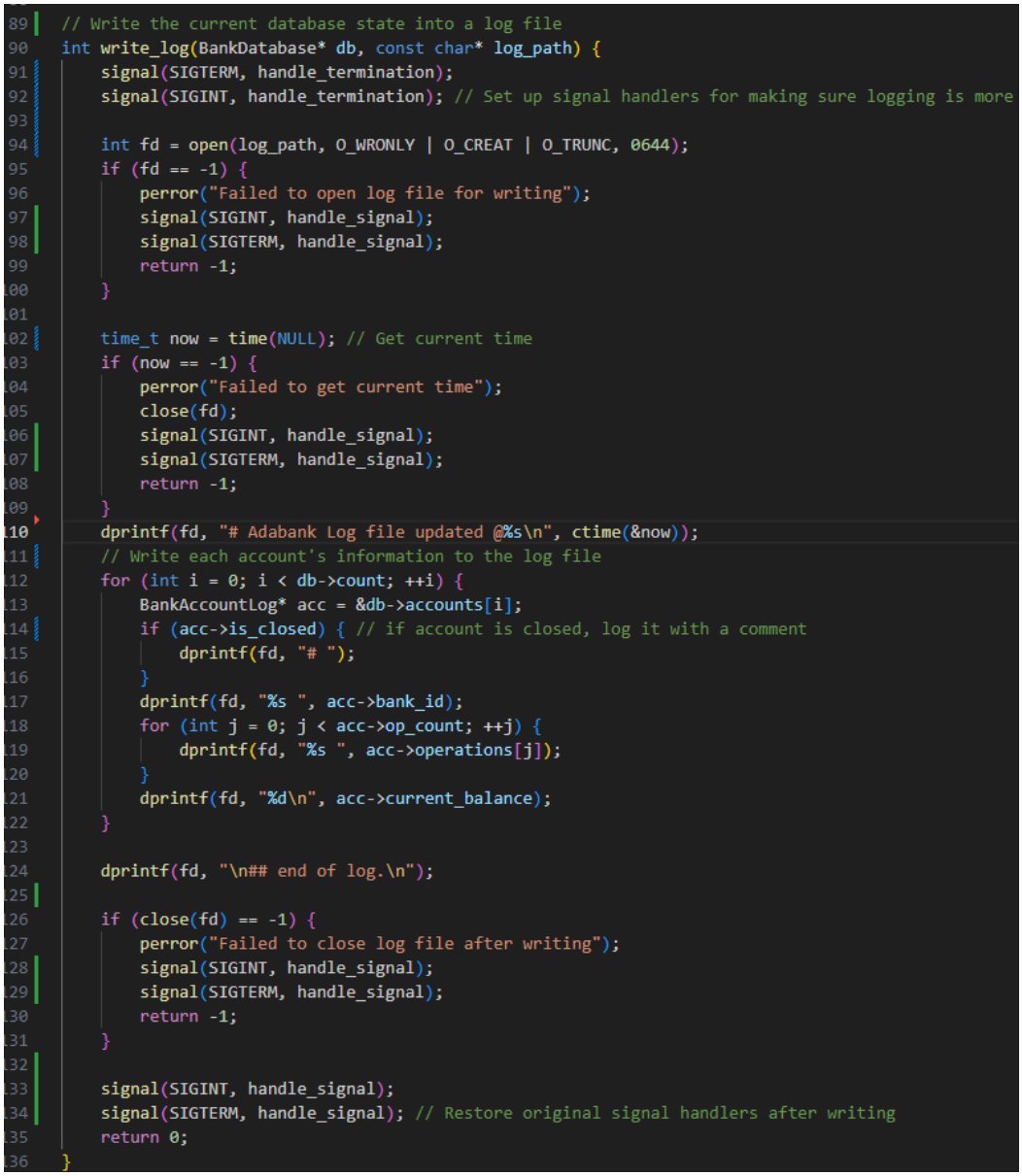
- **Client-Side Timeout**:
  Clients implement timeout handling to detect unresponsive scenarios and terminate safely if they do not receive a response within a specified duration.
- **Server-Side Timeout/Interrupt Handling**:
  The server monitors client activity. If a client or a teller disconnects unexpectedly, the server ensures that shared resources (e.g., FIFOs, shared memory) are properly cleaned up.
- **Teller-Side Handling**:
  Tellers also handle client disconnections and server signals. They release allocated shared memory and semaphores when interrupted to prevent resource leaks or zombie processes.

4.7. Synchronization Primitives Used

- **Shared Memory**:
  Used for efficient, low-latency communication of transaction data between the server and teller processes.
- **Semaphores (sem_t)**:
  - sem_request_ready: Signaled by the server to notify a teller that a transaction is available.
  - sem_response_ready: Signaled by the teller to notify the server that processing has completed.
- **Signals**:
  Custom signal handlers for SIGINT and SIGTERM allow clean process termination and consistent resource deallocation. SIGUSR1 is also used for server’s termination to teller then teller to client.

4.8. Database and Log Management

Write log function for insteresting solution of deadlock while using semaphores with shared_memory.



```c
// Write the current database state into a log file
int write_log(BankDatabase* db, const char* log_path) {
    signal(SIGTERM, handle_termination);
    signal(SIGINT, handle_termination); // Set up signal handlers for making sure logging is more

    int fd = open(log_path, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1) {
        perror("Failed to open log file for writing");
        signal(SIGINT, handle_signal);
        signal(SIGTERM, handle_signal);
        return -1;
    }

    time_t now = time(NULL); // Get current time
    if (now == -1) {
        perror("Failed to get current time");
        close(fd);
        signal(SIGINT, handle_signal);
        signal(SIGTERM, handle_signal);
        return -1;
    }
    dprintf(fd, "# Adabank Log file updated @%s\n", ctime(&now));
    // Write each account's information to the log file
    for (int i = 0; i < db->count; ++i) {
        BankAccountLog* acc = &db->accounts[i];
        if (acc->is_closed) { // if account is closed, log it with a comment
            dprintf(fd, "# ");
        }
        dprintf(fd, "%s ", acc->bank_id);
        for (int j = 0; j < acc->op_count; ++j) {
            dprintf(fd, "%s ", acc->operations[j]);
        }
        dprintf(fd, "%d\n", acc->current_balance);
    }

    dprintf(fd, "\n## end of log.\n");

    if (close(fd) == -1) {
        perror("Failed to close log file after writing");
        signal(SIGINT, handle_signal);
        signal(SIGTERM, handle_signal);
        return -1;
    }

    signal(SIGINT, handle_signal);
    signal(SIGTERM, handle_signal); // Restore original signal handlers after writing
    return 0;
}
```

4.9. Signal Handling and Cleanup

Temp signal_handler on write_log for making it more atomic.

```c
// Signal handler to set termination flag
void handle_termination(int signo) {
    if (signo == SIGTERM || signo == SIGINT) {
        // Set global termination flag to 1 after logging depending
        is_terminated = 1;
    }
}
```

After each execution I check that flag for termination process on purpose.

```c
write_log(&g_db, LOG_PATH);
if (is_terminated == 1){ // if signa
    handle_signal(SIGTERM);
}
```

Tellers signal handler

If it received SIGUSR1 from server it means server is going to shutdown so it send sigusr1 to inform about that termination to client.

```c
// Handle signals for teller process (cleanup resources)
void handle_teller_signal(int signo) {
    printf("Teller PID %d received signal %d. Cleaning up...\n", getpid(), signo);

    if (signo == SIGUSR1) {
        kill(g_shm->request.client_pid, SIGUSR1);
    }

    g_shm->response.success = 0;
    snprintf(g_shm->response.message, sizeof(g_shm->response.message), "Something went wrong. Teller is terminated.");

    if (strlen(client_fifo_path) > 0) {
        if (unlink(client_fifo_path) == -1) {
            perror("Failed to unlink client FIFO");
        } else {
            printf("Client FIFO %s unlinked.\n", client_fifo_path);
        }
    }

    if (g_shm && g_shm_name) {
        sem_destroy(&g_shm->sem_request_ready);
        sem_destroy(&g_shm->sem_response_ready);

        destroy_shared_memory(g_shm_name, g_shm, 1);
        free(g_shm_name);
        g_shm = NULL;
        g_shm_name = NULL;
    }

    free_database(&g_db);
    _exit(0); // Safe exit from signal handler
}
```

cleanup_and_exit() for termination of server

```c
// Clean up resources and exit the program
void cleanup_and_exit(int status) {
    printf("Signal received closing active Tellers...\n");

    int i = 0;
    while (i < teller_pid_count) {
        kill(g_teller_pids[i], SIGUSR1);
        waitTeller(g_teller_pids[i], &status);
        ++i;
    }

    printf("Removing ServerFIFO... Updating log file...\n");
    destroy_shared_memory(g_db.shm_name, NULL, 0);

    write_log(&g_db, LOG_PATH);
    close(g_db.server_fd);
    free_database(&g_db);
    remove_fifo(SERVER_FIFO_NAME);
    printf("%s says \"Bye\"...\n", g_db.bank_name);
    exit(status);
}
```

Signal handler for server

```c
// Signal handler to trigger cleanup
void handle_signal(int signo) {
    keep_running = 0;
    cleanup_and_exit(0);
}
```

5. Challenges that I Faced & My Solutions

5.1 Synchronization Issues Leading to Deadlocks

Challenge:
One critical issue encountered was ensuring that the teller only sends a response back to the client after the main server successfully completes the database update and log file writing. Initially, this synchronization was attempted using additional semaphores, but it introduced complexity and potential new deadlocks.

Solution:
Instead of relying solely on semaphores for signaling the completion of database operations, the database update and log writing sections were restructured to be as atomic as possible. Database update and Teller's responses are not synchronized but after that atomic attempt I am pretty sure that I cannot face any other problem except for system call error.

5.2 Race Conditions in Shared Memory Access

Challenge:
Shared memory segments accessed concurrently by tellers and the server introduced race conditions, where data inconsistencies or crashes could occur due to unsynchronized reads and writes.

Solution:
Semaphore-based synchronization was enforced rigorously. `sem_request_ready` and `sem_response_ready` semaphores ensured that at no point could both server and teller modify or read the shared memory simultaneously. Additionally, shared memory accesses were isolated and minimized to only the necessary transaction data.

5.3 Client Disconnection During Transaction Processing

Challenge:
Unexpected client disconnections during an active transaction led to broken FIFO descriptors, causing tellers or the server to block or crash.

Solution:
Signal handling (SIGPIPE, SIGTERM, SIGINT) was carefully implemented for both server and teller processes. Upon detecting a closed FIFO or a signal, the teller immediately cleaned up its resources and safely terminated without affecting the server. Similarly, the server monitored and properly closed dangling FIFOs when disconnections occurred.