SYSC 4001 B Final Project Report

David Casciano, 101069255

Sebastian Ross, 101073435

Architecture

The system comprises 4 processes, the ATM's, Database Server, Database Editor, and Interest Calculator. All of which communicate through IPC services. List below are the processes and their specifications.

ATM(s)

Role: The role of an ATM is to gather information from the user and forward it onto the server. It will then wait for a response and display information provided by the server to the client. It will run in an infinite loop until there is an "X" in the account number or "X" is selected in the menus.

Number of Process: The number of ATM's are bounded only by the number of PID's given that the ATM's message type is its PID.

Messages:

- Message Type: The PID of the process that the ATM is running on
- Message Text:

Request Type	Message Text	Expected Response
PIN / Login	{ACCOUNT_NUM},{PIN_NUM}	"OK" "PIN_WRONG" "ANF"
Balance	"BALANCE"	{ACCOUNT_BALANCE}
Withdraw	"WITHDRAW,"{AMOUNT}	"NSF" "FUNDS_OK"
Deposit	"DEPOSIT,"{AMOUNT}	"DEPOSIT_OK"
Loan	"LOAN,"{AMOUNT}	"LOAN_OK"

IPC Services used: Message Passing

DB Server

Role: Read messages from an inbound message queue, and handle the request accordingly. Once the request is handled, generate a response and send a message back to the ATM or Editor on the outbound message queue.

Number of Process: 1

Request Handling:

- Message Type: The PID of the process that the ATM is running on
- Message Text:

Message Text	Condition	Expected Response
{ACCOUNT_NUM},{PIN_NUM}	If account and pin match	“OK”
{ACCOUNT_NUM},{PIN_NUM}	If pin doesn’t match	“PIN_WRONG”
{ACCOUNT_NUM},{PIN_NUM}	If account is not found	“ANF”
“BALANCE”	N/A	{ACCOUNT_BALANCE}
“WITHDRAW,”{AMOUNT}	Not enough funds	“NSF”
“WITHDRAW,”{AMOUNT}	There are enough funds	“FUNDS_OK”
“DEPOSIT,”{AMOUNT}	N/A	“DEPOSIT_OK”
“LOAN,”{AMOUNT}	N/A	“LOAN_OK”
“UPDATE_DB,”{ACCOUNT}	N/A	N/A

Shared Memory: Interest Rate Struct, Database Text File

IPC Services used: Message Passing, Semaphores, Shared Memory

DB

A text file containing account entries in a specific format

Account No.	Encoded PIN	Funds available
5 characters	3 characters	Floating point number

DB Editor

Role: Gather information from the user, which will then be concatenated into a string which is the format of an account in the database. It will then send the message to the Server. Another option is adjusting the interest rate, which will be stored in shared memory.

Number of Process: 1

Request Handling:

- Message Type: The PID of the process that the ATM is running on
- Message Text:

Request Type	Message Text	Expected Response
Acc. Creation	“UPDATE_DB,”{ACCOUNT},{PIN},{BALANCE}	N/A

Shared Memory: Interest Rate Struct

IPC Services used: Message Passing, Semaphores, Shared Memory

Interest Calculator

Role: Once every minute, it traverses the whole DB, and if the customer has a positive balance, adds set interest to their funds. If the customer has a negative balance, it subtracts interest from their funds.

Number of Process: 1

Shared Memory: Interest Rate Struct, Database Text File

IPC Services used: Semaphores, Shared Memory

Deadlock and Livelock Solutions

Deadlock

Since the majority of the system operates through message passing, there isn’t much opportunity for a deadlock case to arise. In addition, the database server cannot request a lock on a specific line in a text file, therefore limiting the shared resources to one. To generate a deadlock case, we added an extra component which needed access to multiple shared resources at once, this being the loan application. As deadlocks are a very specific case of starvation the following conditions where made:

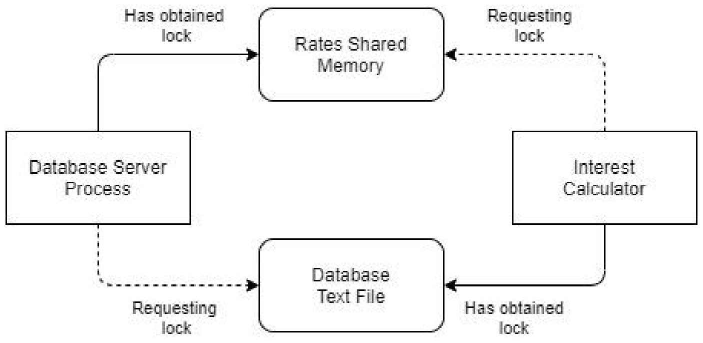

Figure 1: Case where a deadlock is made

To create the deadlock both the Server and Interest components have to access the interest rates and database text file at the same time. To create this condition, the user must request a loan at the same time as the interest component is updating the database. This is because the server requests lock on the rates first then requests lock on the database text file. The interest component requests access to the database first, then the rates. If both of these were to happen at the same time it would create a deadlock. To increase the probability of the deadlock happening, there was a 30 second delay added between requesting resources in the interest component. By doing this we were able to log the system states and visualize the event leading up to the deadlock.

A solution to this deadlock case would be using a circular wait, where the system has an order of requests to lock shared memory. As seen above, both accessed the two resources in the opposite order at the exact same time. Using a circular wait, one of the processes would wait for both resources to be released, simply by waiting to gain access to the first resource.

Livelock

For the livelock case, we tried to implement an unfair algorithm where odd PID processes got an extra 30 seconds with lock on the database. This did not create a livelock but it did create a starvation case. This can be seen in the livelock logs text file that was submitted with the project.

Livelocks can come from handling deadlocks incorrectly. For an example that is applicable to our system, if both the server and the interest calculator were to detect a deadlock and both give up the lock on each resource, it would cause a deadlock. This would be because they would constantly be switching resources and waiting on the resource that it had just released. A solution to the livelock would be the same as a deadlock. Using a simple circular wait, the process will never be in a situation where they hold one resource and are waiting for another.

New Functionality

Three new features were added to the system for this project, these being:

- Deposit option from the ATM
- Loan option from the ATM
- Variable interest rates

In the following paragraphs, the features and their IPC implementation will be described.

Deposit Option From The ATM

To mimic the functionality of the real ATM, we added a deposit function which will prompt you for an amount to deposit. The ATM will then send the amount to the database server through message queues, this would be through the in-bound queue where the server is receiving with flag zero. Once received, the server will add the balance to the local store and the database. Once the funds are added a confirmation message is returned to the ATM.

IPC Services used:
- Semaphores
- Message Passing

Loan Option From The ATM

To use the complete function of the interest calculator component, we added a loan application portal in the ATM. Since the withdrawal feature does not let you take out more than your account has, you will never have a negative balance. The feature begins once you select it in the ATM menu. It will first prompt you for an amount, once entered it will send the amount via message queue to the server. Once the server has it it will try to attain a lock on the shared memory struct for the rates. It will calculate the loan with an initial fee of the interest rate at the moment. The server will then subtract the amount from the balance and gain lock on the database text file to update the balance. Lastly, the server will respond with a confirmation message.

IPC Services used:
- Semaphores
- Message Passing
- Shared Memory

Variable interest Rates

Using the Database Editor component, the user is able to adjust the interest rates. A structure was created which would hold both the positive and negative interest rates. From the editor, you could choose to edit these rates. Once navigated into the menu, it will prompt you for the new positive interest rate and then the negative interest rate. While updating the struct it will attain lock, so that others cannot read or edit.

IPC Services used:
- Semaphores
- Shared Memory