ECOLE NATIONALE
SUPÉRIEURE
D'INFORMATIQUE

Lab Work on Distributed Architectures - ALOG
By: Mahdia Toubal

Blockchain Architecture Simulation: System Guide

Github

This guide provides an overview of the blockchain architecture simulation I've developed for the distributed systems project. The simulation showcases key blockchain features including transaction processing, consensus mechanisms, and security features.

Project Overview

This implementation demonstrates a peer-to-peer blockchain system with:

- Block creation and transaction management
- Hash-based block chaining
- Distributed consensus through a proof-of-work mechanism
- Network simulation with multiple nodes
- Transaction validation and malicious node detection

System Architecture

The system consists of the following components:

Core Data Structures

1. Transactions
   - Sender and receiver addresses
   - Transaction amount
   - Timestamp
2. Blocks
   - Block index
   - Timestamp
   - Array of transactions
   - Previous block hash
   - Current block hash
   - Pointer to next block

3. Blockchain

    - Linked list of blocks
    - Current proof-of-work value
    - Mutex for thread safety

4. Network Nodes

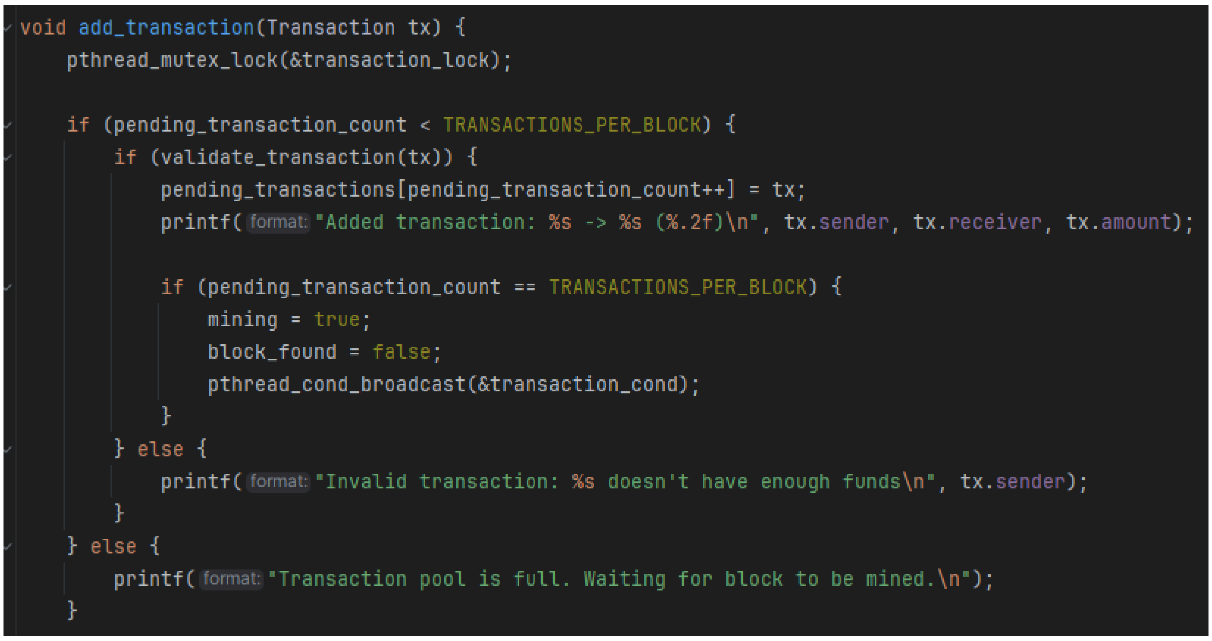
    - Unique node identifier
    - Local blockchain copy
    - Mining thread
    - Status flags (running, malicious)
    - Mining rewards tracking

Key System Features

1. Transaction Processing

Transactions are validated before being added to the pending transaction pool:

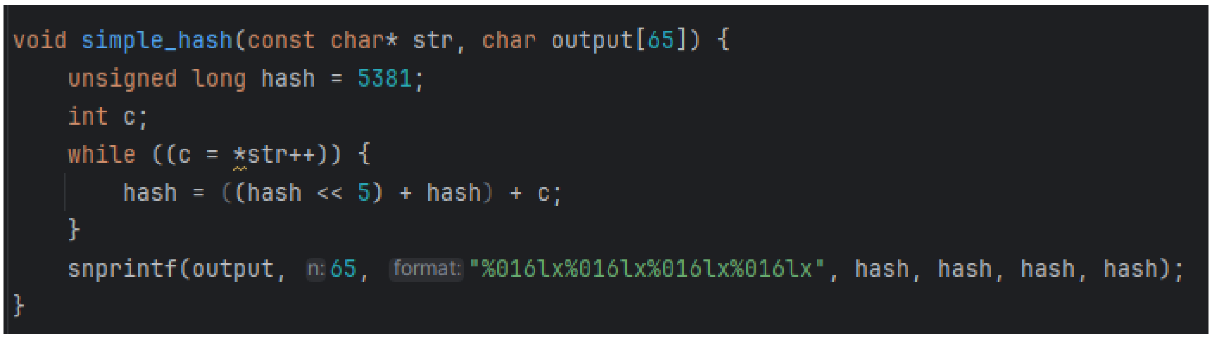
- Sender must exist in the system
- Sender must have sufficient funds
- Pending transactions are grouped into blocks (3 per block)



```
void add_transaction(Transaction tx) {
    pthread_mutex_lock(&transaction_lock);

    if (pending_transaction_count < TRANSACTIONS_PER_BLOCK) {
        if (validate_transaction(tx)) {
            pending_transactions[pending_transaction_count++] = tx;
            printf( format: "Added transaction: %s -> %s (%.2f)\n", tx.sender, tx.receiver, tx.amount);

            if (pending_transaction_count == TRANSACTIONS_PER_BLOCK) {
                mining = true;
                block_found = false;
                pthread_cond_broadcast(&transaction_cond);
            }
        } else {
            printf( format: "Invalid transaction: %s doesn't have enough funds\n", tx.sender);
        }
    } else {
        printf( format: "Transaction pool is full. Waiting for block to be mined.\n");
    }
```

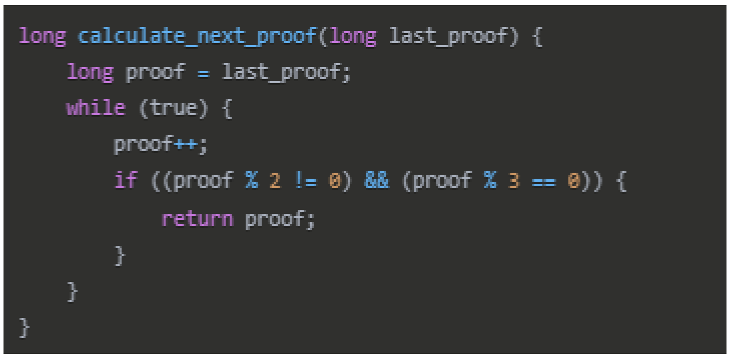
2. Block Chaining with Hash

Each block contains a hash of its own content and the previous block's hash, creating a tamper-evident chain:



```c
void simple_hash(const char* str, char output[65]) {
    unsigned long hash = 5381;
    int c;
    while ((c = *str++)) {
        hash = ((hash << 5) + hash) + c;
    }
    snprintf(output, n: 65, format: "%016lx%016lx%016lx%016lx", hash, hash, hash, hash);
}
```

3. Consensus Mechanism



```c
long calculate_next_proof(long last_proof) {
    long proof = last_proof;
    while (true) {
        proof++;
        if ((proof % 2 != 0) && (proof % 3 == 0)) {
            return proof;
        }
    }
}
```

The consensus mechanism implements a simplified proof-of-work algorithm:

1. Starts with the previous proof value
2. Increments the value until finding one that satisfies two conditions:
    - Not divisible by 2 (odd number)
    - Divisible by 3
3. Returns the valid proof value

While simplified compared to real blockchain implementations, this mechanism demonstrates key consensus principles:

- Requires computational work to find valid proofs
- Proof validity is easily verifiable
- Adjusts difficulty based on network state

4. Mining Process

The mining process occurs in several steps:

1. Wait for enough transactions to form a block
2. Copy pending transactions to local storage
3. Attempt to find a valid proof of work
4. Create a new block with:
    - Current transactions
    - Previous block hash
    - Valid proof of work

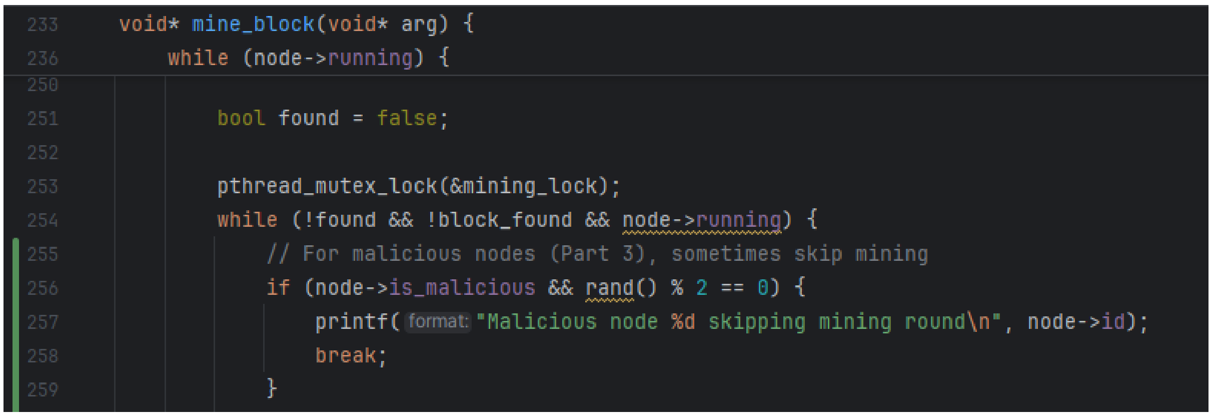
5. For malicious nodes, possibly tamper with transaction data
6. Broadcast the new block to all nodes
7. Reset the transaction pool

This process simulates the competitive nature of blockchain mining, where nodes race to find valid proofs and add blocks to the chain.

5. Security Features

The system includes protections against malicious behavior:

- Transaction validation prevents double-spending
- Malicious nodes is simulated to test system resilience:



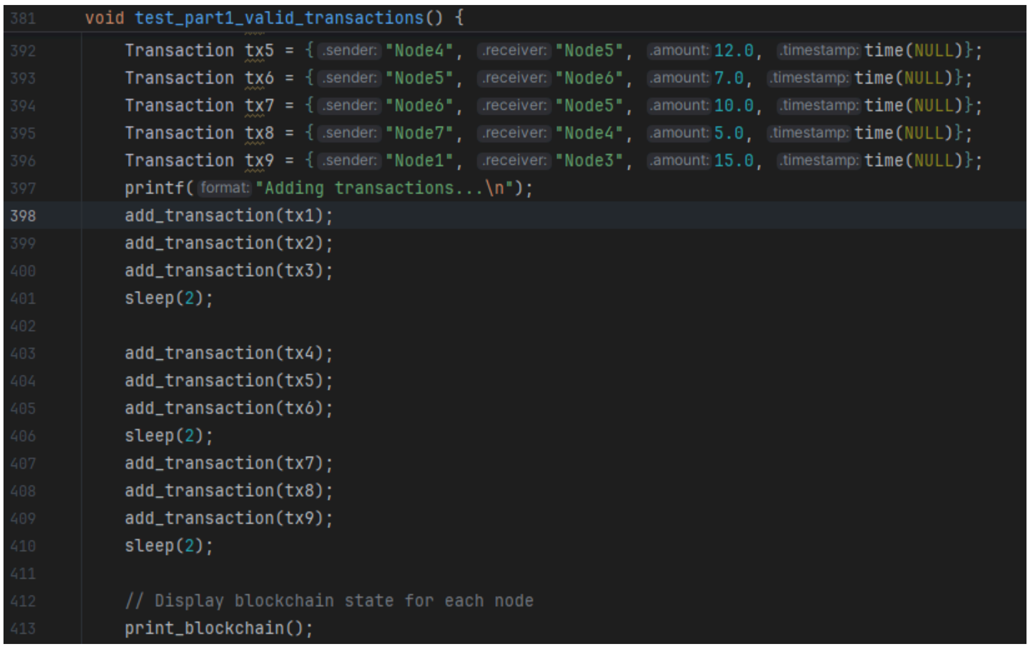
```c
void* mine_block(void* arg) {
    while (node->running) {

        bool found = false;

        pthread_mutex_lock(&mining_lock);
        while (!found && !block_found && node->running) {
            // For malicious nodes (Part 3), sometimes skip mining
            if (node->is_malicious && rand() % 2 == 0) {
                printf( format: "Malicious node %d skipping mining round\n", node->id);
                break;
            }
```

System Testing

The implementation includes test scenarios to verify system functionality:

1. Valid Transaction Testing

Demonstrates normal blockchain operation with valid transactions.

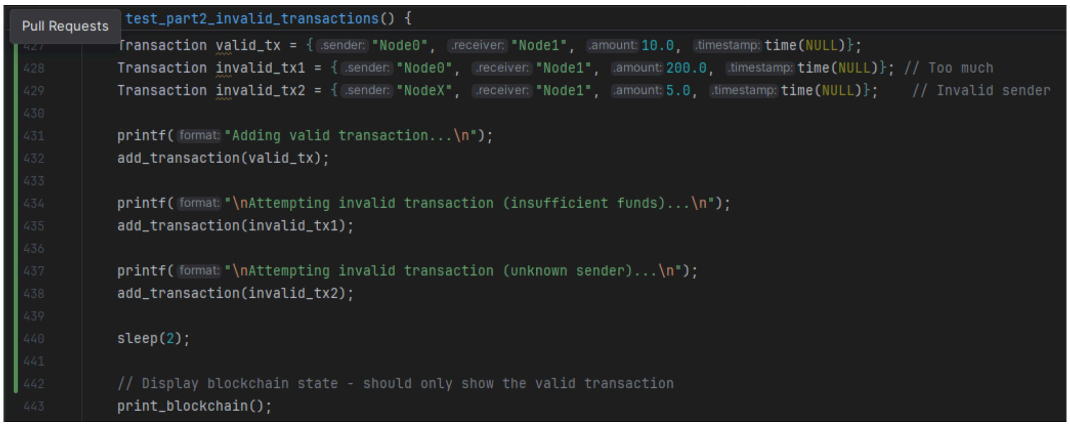


```c
void test_part1_valid_transactions() {
    Transaction tx5 = { .sender: "Node4", .receiver: "Node5", .amount: 12.0, .timestamp: time(NULL)};
    Transaction tx6 = { .sender: "Node5", .receiver: "Node6", .amount: 7.0, .timestamp: time(NULL)};
    Transaction tx7 = { .sender: "Node6", .receiver: "Node5", .amount: 10.0, .timestamp: time(NULL)};
    Transaction tx8 = { .sender: "Node7", .receiver: "Node4", .amount: 5.0, .timestamp: time(NULL)};
    Transaction tx9 = { .sender: "Node1", .receiver: "Node3", .amount: 15.0, .timestamp: time(NULL)};
    printf( format: "Adding transactions...\n");
    add_transaction(tx1);
    add_transaction(tx2);
    add_transaction(tx3);
    sleep(2);

    add_transaction(tx4);
    add_transaction(tx5);
    add_transaction(tx6);
    sleep(2);
    add_transaction(tx7);
    add_transaction(tx8);
    add_transaction(tx9);
    sleep(2);

    // Display blockchain state for each node
    print_blockchain();
```

2. Invalid Transaction Testing

Shows how the system rejects transactions with:

- Insufficient funds
- Non-existent sender accounts



```c
test_part2_invalid_transactions() {
    Transaction valid_tx = { .sender: "Node0", .receiver: "Node1", .amount: 10.0, .timestamp: time(NULL)};
    Transaction invalid_tx1 = { .sender: "Node0", .receiver: "Node1", .amount: 200.0, .timestamp: time(NULL)}; // Too much
    Transaction invalid_tx2 = { .sender: "NodeX", .receiver: "Node1", .amount: 5.0, .timestamp: time(NULL)};    // Invalid sender

    printf( format: "Adding valid transaction...\n");
    add_transaction(valid_tx);

    printf( format: "\nAttempting invalid transaction (insufficient funds)...\n");
    add_transaction(invalid_tx1);

    printf( format: "\nAttempting invalid transaction (unknown sender)...\n");
    add_transaction(invalid_tx2);

    sleep(2);

    // Display blockchain state - should only show the valid transaction
    print_blockchain();
```

3. Malicious Node Testing

Tests system resilience with different percentages of malicious nodes:

- <50% malicious nodes (2 out of 8)
- 50% malicious nodes (5 out of 8)

System Evaluation

The simulation successfully demonstrates key blockchain features:

1. Transaction Integrity:
   - Valid transactions are processed correctly
   - Invalid transactions are rejected
2. Block Chaining:
   - Blocks are properly linked through hash references
   - Block history is maintained consistently
3. Distributed Consensus:
   - Nodes reach agreement on the blockchain state
   - Mining rewards are distributed fairly
4. Security:
   - System maintains integrity with minority malicious nodes
   - Demonstrates vulnerability when majority of nodes are malicious

Implementation Details

The project is implemented in C using:

- POSIX threads for concurrency
- Mutex locks for thread safety
- Condition variables for synchronization
- Simple hash functions for block chaining (since it’s for educational purpose)

Conclusion

This blockchain simulation demonstrates the fundamental architecture of distributed ledger systems. It helped me understand the architecture in a more practical and fun way!