A Simple Banking System Simulation

A Simple Banking System Simulation.

In this scenario, we will simulate a banking application where multiple customers can create accounts, deposit and withdraw money, and check their balances concurrently. This will demonstrate the core OS functionalities such as process creation, memory management, multithreading, synchronization, CPU scheduling, and IPC.

Problem Statement: Simple Banking System Simulation

Objective :

Create a simulated banking system where multiple customers can perform transactions concurrently, showcasing OS functionalities.

Modules Implementation

1. System Call Interface (10 Marks)

Objective:

Provide basic system calls for account management and transaction processing.

Implementation Steps:

Define API Function:

1) **create_account(customer_id, initial_balance):** Creates a new account.
2) **deposit(account_id, amount):** Deposits money into an account.
3) **withdraw(account_id, amount):** Withdraws money from an account.
4) **check_balance(account_id):** Checks the balance of an account.

Key Concepts:

System calls will be the interface through which user requests are processed by the OS.

2. Process Creation(10 Marks)

Objective:

Implement the ability to create and manage processes for each customer transaction.

Implementation Steps:

Use a process table to track customer transactions as processes.

Each transaction can be a separate process that updates shared account data.

Implement functions to create and terminate transaction processes.

Features:

Each transaction process should have its own context (e.g., transaction ID, status).

3. Multithreading and Synchronization (10 Marks)

Objective:

Simulate concurrent transactions using threads and ensure data consistency.

Implementation Steps:

Use threads within a process to handle multiple transactions for the same customer.

Implement mutexes or semaphores to synchronize access to shared account data, preventing race conditions.

Key Concepts:

Synchronization will ensure that only one thread can access account data at a time, preventing inconsistencies.

4. CPU Scheduling (10 Marks)

Objective:

Simulate CPU scheduling to manage transaction execution.

Implementation Steps:

Implement a Round Robin scheduling algorithm to allocate CPU time to each transaction process.

Track metrics like average waiting time for transactions and CPU utilization.

Visualization:

Display a Gantt chart to visualize how CPU time is allocated to different transactions.

5. Memory Management and Paging (10 Marks)

Objective:

Manage memory for storing account information and transaction data.

Implementation Steps:

Divide memory into pages for storing account data and transaction logs.

Implement a simple page replacement algorithm (e.g., LRU) to manage memory efficiently.

Visualization:

Display a memory map showing how pages are allocated for accounts and transactions.

6. Inter-Process Communication (IPC) (10 Marks)

Objective:

Enable communication between transaction processes.

Implementation Steps:

Use message queues to pass messages between processes, such as notifying a process when a transaction is completed.

Support both synchronous and asynchronous communication between transaction processes.

Key Concepts:

IPC will help maintain communication between processes while ensuring data isolation.

Example Workflow

Customer Interaction:

A customer requests to create an account through the system call interface.

The system creates a new process for this request.

Concurrent Transactions:

The customer initiates multiple transactions (e.g., deposits and withdrawals).

Each transaction is handled by a separate thread within a process.

Synchronization:

As threads attempt to update the account balance, mutexes ensure that only one thread can modify the balance at a time.

CPU Scheduling:

The CPU scheduler allocates time slices to each transaction process, ensuring fair execution.

Memory Management:

Transaction data is stored in pages, with a page replacement algorithm managing memory usage.

Logging Transactions:

Each completed transaction is logged to a file using the file management system.

Communication:

Processes communicate through message queues to notify each other of transaction completions.

Outcome

By implementing this banking system simulation, you will demonstrate the interaction of various OS components in a multitasking environment. This will provide insights into how processes are created and managed, how memory and resources are allocated, and how synchronization ensures data integrity in concurrent operations.

Additional Considerations

Error Handling:(Marks 20)

Implement interface and error handling for insufficient funds, invalid account IDs, etc.

Viva:(Marks 20)

Viva + understanding