Mohamed Salama
Veton Abazovic
Asst3 pdf

Multi-thread bank

We first deiced to create command syntax that allows the user to create, accounts, starts sessions to serve specific accounts, and to exit the client process altogether. We had used create, serve, deposit, withdraw, query, end and quit. We also added an help function to explain what each command did. Server starts a service for a specific account, Deposit and withdraw adds and subtracts amounts from balance. Query command returns the account balance, End command ends the current service session. Quit command disconnects the client from the server and ends the client process. Now for our thread synchronization requirements, we essentially locked the thread so that data cannot be changed by another user, and unlock it once the transaction was done. This was very helpful because it allowed us to have the threads not interfere with each other. This is really important to that we don't have to worry about collisions or other threads mixing with others. Basically every time a customer session was open it had access to that account and only that account causing everything to be separate and not colloid with each other. We used three semaphores in the project , two for the client and one for the server to interrupt current connections threads. We used a function called sem_post to catch SIGINTS, we also used semaphores so that it would be really easy to have things to wait on a timer. In the server we had two threads one that sends information to the server, and one that receives information to the server, and for the server we used two threads along with 2 mutex threads, the mutex threads where used for each client and a separate client service thread will not interfere with each other.