In concurrent programming, concurrent accesses to shared resources can lead to unexpected or erroneous behavior, so parts of the program where the shared resource is accessed are protected. This protected section is the critical section or critical region. It cannot be executed by more than one process. To make it easier to implement or apply critical sections, there are 3 things to check, Mutual exclusion, Progress, and Bound waiting. In windows, we made the bank system and created a log file detailing the date and time when the user deposits and withdraws. We corrected the errors that occurred when depositing and withdrawing money at the same time in the process of depositing and withdrawing money. So, we have implemented how mutual exclusion is possible. This system seems to have no problem with mutual exclusion because only one user can deposit and withdraw money, but when we run two compilations, we found an error that the output was only affected by the last input. Therefore we applied Win32 API for the mutex to use the bank system only one at a time. When one user is using the system, if someone connects, it makes it inaccessible with the statement that someone else is using it. The reason we applied the mutex to the executable file is basically the idea that the system is that on the bank website, when the user logs in with id number 331, the other user with id number 331 that logged in first automatically logs out. However, after we had tested for many times, we found that the deadlock occurred. Also, one of the 4 conditions that must occur No preemption which is one user logs in the other user may not log in. Therefore, we made the program if the first user logs in and it for a limited time, it automatically logs out, so the other user can log in. In conclusion, to prevent mutual exclusion, a mutex was used so that the user could not access at the same time and made the program synchronized perfectly.