PROJECT 2

DESIGN – BANK SIMULATION

List of Semaphores

S. No.	Name of Semaphore	Array size (if array)	Initial Value	Purpose
1.	Max_customers	1	5	To limit the number of customer threads being run at a time.
2.	Queue1notempty	1	0	To signal that the queue for bank teller is not empty and bank teller can now remove customer from queue.
3.	Queue2notempty	1	0	To signal that the queue for loan officer is not empty and loan officer can now remove customer from queue.
4.	BanktellerRequest	2	0	To request for a withdrawal or deposit from bank teller after a specific bank teller is ready to serve that customer.
5.	depositReciept	2	0	To signal that deposit is processed by a specific teller.
6.	depositComplete	2	0	To signal that deposit transaction is completed by a specific teller.
7.	withdrawReciept	2	0	To signal that withdrawal is processed by a specific teller.
8.	withdrawalComplete	2	0	To signal that withdrawal transaction is completed by a specific teller.
9.	LoanOfficerRequest	1	0	To request for a loan from loan officer after loan officer is ready to serve that customer.
10.	LoanOfficerReciept	1	0	To signal that loan is approved is processed by loan officer.
11.	LoanTransactionComplete	1	0	To signal that loan transaction is completed by loan officer.
12.	tellerReady	5	0	To signal that a teller is ready to serve a specific customer. For each customer we have a semaphore.
13.	loanOfficerReady	5	0	To signal that a loan officer is ready to serve a specific customer. For each customer we have a semaphore.
14.	mutex1	1	1	To ensure mutual exclusion for bank teller queue, so that addition or removal from the queue is done one at a time.
15.	mutex2	1	1	To ensure mutual exclusion for loan officer queue, so that addition or removal from the queue is done one at a time.

Pseudo Code

```
/* program BankSimulation */

Semaphore max_customers =5;
Semaphore mutex1 =0, mutex2 =0, queue1NotEmpty =0, queue2notempty =0;
Semaphore loanOfficerReady[5] =0, tellerReady[5] = 0;
Semaphore banktellerRequest[2] = 0, depositReciept[2] =0, depositComplete[2] =0;
Semaphore withdrawReciept[2] =0 , withdrawalComplete[2] =0;
Semaphore loanOfficerRequest = 0, loanOfficerReciept = 0, loanTransactionComplete =0;

void main()
{
        NUMCUSTOMERS = 5;
        createThread(LoanOfficer);
        startThread(LoanOfficer);
        for(i=0;i<2;i++)
        {
                createThread(BankTeller[i]);
                startThread(BankTeller[i]);
        }
        for(i=0;i<NUMCUSTOMERS;i++)
        {
                createThread(Customer[i]);
                startThread(Customer[i]);
        }
        for(i=0;i<NUMCUSTOMERS;i++)
        {
                joinThread(Customer[i]);
        }
}
void Customer()
{
        for(i=0;i<3;i++)                    // Makes each customer thread run for three times
        {
                Wait(max_customers);  // Limits the max number of customer threads running at a time
                Assigntask();               //Assigns task deposit, withdraw or loan.
                If(task == deposit)
                {
                        wait(mutex1);   //critical section for adding to teller queue
                        queueBankTeller.add(customer);
                        signal(queue1NotEmpty); //signals teller that queue is not empty
                        signal(mutex1);
```

```
                wait(tellerReady[customer]);  //waits till teller is ready for this customer thread
                signal(banktellerRequest[teller]); //signals teller to start processing deposit
                wait(depositreciept[teller]); //waits till teller is done with processing
                signal(depositComplete[teller]); //signals teller to move to next customer
        }
        If(task == withdraw) //works similar to deposit
        {
                wait(mutex1);
                queueBankTeller.add(customer);
                signal(queue1NotEmpty);
                signal(mutex1);
                wait(tellerReady[customer]);
                signal(banktellerRequest[teller]);
                wait(withdrawreciept[teller]);
                signal(withdrawalComplete[teller]);
        }

        If(task == loan)
        {
                wait(mutex2); //critical section for adding to loan officer queue
                queueLoanOfficer.add(customer);
                signal(queue2NotEmpty); //signals loan officer that queue is not empty
                signal(mutex2);
                wait(loanOfficerReady[customer]); //waits till officer is ready for this thread
                signal(loanOfficerRequest); //signals teller to start processing loan
                wait(loanOfficerReciept); //waits till loan officer is done with processing
                signal(loanTransactionComplete); //signals officer to move to next customer
        }
        signal(max_customers);
    }
}

void Bankteller()
{
    while(true)
    {
        wait(queue1notempty); //wait till bank teller queue is not empty
        wait(mutex1); // critical section for removing from teller queue
        queueBankTeller.remove(customer);
        signal(mutex1);
        tellerBeginServingCustomer();
        signal(tellerReady[customer]); //teller signals customer that it is ready to serve
        if(task(customer) == deposit)
        {
```

```
                wait(banktellerRequest[teller]); //waits for customers request for processing
                processDeposit();
                signal(depositReciept[teller]); //signals that processing is done
                wait(depositComplete[teller]); //wait till customer frees the teller
        }
        if(task(customer) == withdraw) //works similar to deposit
        {
                wait(banktellerRequest[teller]);
                processWithdrawal();
                signal(withdrawReciept[teller]);
                wait(withdrawalComplete[teller]);
        }


    }
}


void LoanOfficer()
{
    while(true)
    {
        wait(queue2notempty); //wait till loan officer queue is not empty
        wait(mutex2); // critical section for removing from loan officer queue
        queueLoanOfficer.remove(customer);
        signal(mutex2);
        loanOfficerBeginServingCustomer();
        signal(loanOfficerReady[customer]); //officer signals customer that it is ready to serve
        wait(loanOfficerRequest); //waits for customers request for processing
        approveLoan();
        signal(LoanOfficerReciept); //signals that processing is done
        wait(loanTransactionComplete); //wait till customer frees the loan officer
    }

}

/******************************** Pseudo Code Ends**********************************/
```

Submitted by –
PULKIT KHEMKA