```
/**************************************************************************/
/*          CSC 2108H COURSE PROJECT - CASH MACHINE SYSTEM               */
/*                        SOURCE CODE                                     */
/*                  DONE BY JEFFERY TANG                                  */
/*                          WEI ZHOU                                      */
/**************************************************************************/




#define TILL_NUM        2
#define MAX_REQ         3
#define INIT_BALANCE    100
#define UPPER_BOUND     100
#define DAILY_LIMIT           80
#define USER_NUM        2
#define ILLEGAL_TRY           3
#define DAY_BREAK       5

#define TRUE      1
#define FALSE     0

/*************** Till Def Begin**************/

#define TILL_IDLE 0
#define TILL_OCCUPIED   1

typedef TILL {
      byte id;    /* the id identifies each till */
      bit  status; /* whether there is an authorized user on the till or not */
      byte count; /* a counter to keep the number of unsuccessful trials of
                     login */

      byte till_req; /* the type of request sends to control */
      byte till_res; /* the type of response received from control */

      byte msgdata; /* the para of the request sending to control */
      byte user_id; /* the current user that the till is serving */

      chan reply = [MAX_REQ] of {   mtype, byte };
      chan request = [MAX_REQ] of { byte, mtype, byte, byte };
};

TILL m_tills[TILL_NUM];
/*************** Till Def End**************/

/*************** User Def Begin**************/

typedef USER {
      byte  uid;
      byte  upwd;
      byte  balance;
      byte  daily_limit;

      byte    cur_user; /* how many users are logged in with the same userid and
                        card */
```

```
      byte    tmp;      /* a temporary variable to keep the withdraw amount when
                        db is offline */
};

USER  m_users[USER_NUM];
/*************** User Def End**************/

bit   m_dbavailable; /* whether the db is online */

/*************** Message Type Def Begin**************/
mtype = { LOGIN, LOGOUT, BALANCE, WITHDRAW, STATE, /* Request */
                        LOGIN_OK, LOGIN_FAIL,      /* Reply */
                        LOGOUT_OK, LOGOUT_FAIL,
                        BALANCE_OK, BALANCE_FAIL,
                        WITHDRAW_OK, WITHDRAW_FAIL,
                        STATE_OK, STATE_FAIL,
                        ILLEGAL_CARD
};

/*************** Message Type Def End**************/

proctype day_timer()
{
    byte counter;
    do
    :: /* a new transaction day */
                counter = 0;
                do
                :: (counter < USER_NUM) ->
                  /* reset the daily_limit on each card */
                        m_users[counter].daily_limit = DAILY_LIMIT;
                  /* if the balance is 0, reset it to INIT_BALANCE */
                  if
                  :: m_users[counter].balance == 0 ->
                        m_users[counter].balance = INIT_BALANCE;
                  :: else -> skip;
                  fi;
                        counter = counter + 1;
                :: else -> break;
                od;
     :: /* db is online */
            m_dbavailable = TRUE;
            counter = 0;
            /* deduct the withdraw amount when db is offline from the balance on
               each card */
            atomic{
                do
                :: (counter < USER_NUM) ->
                        m_users[counter].daily_limit = DAILY_LIMIT;
                  if
                  :: (m_users[counter].balance <= m_users[counter].tmp) ->
                        m_users[counter].balance = 0;
                  :: else ->
                        m_users[counter].balance = m_users[counter].balance -
                                                   m_users[counter].tmp;
                        m_users[counter].tmp = 0;
                  fi;
```

```
                        counter = counter + 1;
                :: else -> break;
            od;
            }
    :: /* db is offline */
            m_dbavailable = FALSE;
    :: else -> skip;
    od;
}

proctype till(TILL till_info)
{
      byte action ;

      byte pin;
      byte pre_user_id;

      do
      ::
            if
            /* the till is not serving a user */
            :: (till_info.status == TILL_IDLE) ->
                 /* LOGIN */
                  if
                :: till_info.user_id = 0;
                :: till_info.user_id = 1;
                fi;

                if
                :: pin = 0;
                :: pin = 1;
                fi;

                till_info.till_req = LOGIN;
                till_info.request ! till_info.id, LOGIN,
                                          till_info.user_id, pin;
                printf(">>>>> till[%d]::send_ login \n", till_info.id);
                till_info.reply ? action, till_info.msgdata;
                till_info.till_res = action;
                printf("<<<<< till[%d]::read_ login \n", till_info.id);

                if
                :: (action == LOGIN_OK) ->
                  till_info.status = TILL_OCCUPIED;
                  m_users[till_info.user_id].cur_user =
                        m_users[till_info.user_id].cur_user + 1;
                  pre_user_id = USER_NUM + 1;
                  assert (m_users[till_info.user_id].cur_user <= TILL_NUM);
                  till_info.count = 0;
                :: else ->
                  /* keep track of the number of login failure */
                  if
                  :: (till_info.user_id == pre_user_id) ->
                        till_info.count = till_info.count + 1;
                  :: else
                        till_info.count = 1;
                        pre_user_id = till_info.user_id;
```

```
              fi;

              /* identify an illegal card */
              if
              :: (till_info.count >= ILLEGAL_TRY) ->
                    printf("!!!!! till[%d]::ILLEGE_CARD \n", till_info.id);
                    till_info.till_res = ILLEGAL_CARD;
                    till_info.count = 0;
                    pre_user_id = USER_NUM + 1;
              :: else -> skip;
              fi;
            fi;
        ::  else ->
              if
              ::
                    /* LOGOUT */
                  till_info.till_req = LOGOUT;

                    assert (m_users[till_info.user_id].cur_user >= 1 );
                    m_users[till_info.user_id].cur_user =
                           m_users[till_info.user_id].cur_user - 1;
                    till_info.status = TILL_IDLE;
                    printf(">>>>> till[%d]::logout done \n", till_info.id);
                    till_info.till_res = LOGOUT_OK;
              ::
                    /* WITHDRAW */
                  till_info.msgdata = 30;
                  till_info.till_req = WITHDRAW;

                  till_info.request ! till_info.id, WITHDRAW,
                                till_info.user_id, till_info.msgdata;
                  printf(">>>>> till[%d]::send_ withdraw \n", till_info.id);
                  till_info.reply ? action, till_info.msgdata;
                  printf("<<<<< till[%d]::read_ withdraw \n", till_info.id);
                  till_info.till_res = action;

            ::  /* STATEMENT */
                  till_info.till_req = STATE;
                  till_info.request ! till_info.id, STATE,
                          till_info.user_id, 0;
                  printf(">>>>> till[%d]::send_ statement \n",
                          till_info.id);
                  till_info.reply ? action, till_info.msgdata;
                  printf("<<<<< till[%d]::read_ statement \n",
                          till_info.id);
                  till_info.till_res = action;

          :: /* BALANCE */
                  till_info.request ! till_info.id, BALANCE,
                          till_info.user_id, 0;
                  printf(">>>>> till[%d]::send_ balance \n", till_info.id);
                  till_info.reply ? action, till_info.msgdata;
                  printf("<<<<< till[%d]::read_ balance \n", till_info.id);
                  till_info.till_res = action;
          fi;
      fi;
```

```
      od;
}

proctype control(TILL till_info)
{
      byte id;
      byte msgtype, msgdata1, msgdata2;

      byte till_req;
      byte till_userid;
      byte till_amount;

      do
      ::
          till_info.request ? id, msgtype, msgdata1, msgdata2;
            printf("<<<<< control[%d]:: read_request\n", till_info.id);
            if
            :: (id != till_info.id) ->
                        printf("Wrong message! \n");
            :: else ->
                  till_req = msgtype;
                  if
                  /* LOGIN */
                  :: (till_req == LOGIN) ->
                        /* encrypt algorithm and authorization*/
                        if
                        :: (m_users[msgdata1].upwd == msgdata2 + 'a') ->
                              till_info.reply ! LOGIN_OK, 0;
                              printf(">>>>>> control[%d]:: reply - login_ok\n",
                                          till_info.id);
                        :: else ->
                              till_info.reply ! LOGIN_FAIL, 0;
                              printf(">>>>>> control[%d]:: reply -
                                          login_fail\n", till_info.id);
                        fi;
                  /* WITHDRAW */
                  :: (till_req == WITHDRAW) ->
                      till_userid = msgdata1;
                      till_amount = msgdata2;
                      if
                      :: (till_userid >= USER_NUM) ->
                          till_info.reply ! WITHDRAW_FAIL, 0;
                          printf(">>>>>> control[%d]:: reply -
                                          withdraw_fail\n", till_info.id);
                      :: else ->
                        if
                        :: (m_dbavailable == TRUE) ->
                              atomic {
                              /* in case the db is online, the withdraw amount
                                    should less than the balance
                                    and the daily withdraw limit */
                                if
                                :: (( m_users[till_userid].daily_limit <
                                            till_amount )
                                        || ( m_users[till_userid].balance <
                                            till_amount ))->
                                      till_info.reply ! WITHDRAW_FAIL, 0;
```

```
                    printf(">>>>> control[%d]:: reply –
                              withdraw_fail\n", till_info.id);
                :: else ->
                     m_users[till_userid].daily_limit =
                              m_users[till_userid].daily_limit
                              - till_amount;
                     m_users[till_userid].balance =
                              m_users[till_userid].balance –
                                    till_amount;
                     till_info.reply ! WITHDRAW_OK, 0;
                     printf(">>>>> control[%d]:: reply –
                              withdraw_ok\n", till_info.id);
                     /* **    assertion   *** */
                     assert (m_users[till_userid].daily_limit
                                                    >=0);
                     assert (m_users[till_userid].balance
                                                    >=0);

              fi;
            }
      :: else ->
            atomic {
            /* in case the db is offline, the withdraw amount
                  should less than the daily withdraw
                  limit and the model accumulates the withdraw
                  amount in tmp*/
              if
              :: ( m_users[till_userid].daily_limit <
                                        till_amount ) ->
                    till_info.reply ! WITHDRAW_FAIL, 0;
                  printf(">>>>> control[%d]:: reply –
                              withdraw_fail\n", till_info.id);
            :: else ->
                  m_users[till_userid].daily_limit =
                              m_users[till_userid].daily_limit
                               - till_amount;
                  if
                  :: (m_users[till_userid].tmp + till_amount <
                                        UPPER_BOUND) ->
                         m_users[till_userid].tmp =
                              m_users[till_userid].tmp +
                                    till_amount;
                  :: else ->
                        m_users[till_userid].tmp =
                                              UPPER_BOUND;
                  fi;
                    till_info.reply ! WITHDRAW_OK, 0;
                  printf(">>>>> control[%d]:: reply –
                              withdraw_ok\n", till_info.id);
                    /* **     assertion   *** */
                    assert (m_users[till_userid].daily_limit
                                                    >=0);
                    assert (m_users[till_userid].balance >=0);

              fi;
            }
      fi;
```

```
                        fi;
                         /* ask for a statement */
                         :: (till_req == STATE) ->
                             till_userid = msgdata1;
                             if
                             :: (till_userid >= USER_NUM) ->
                                 till_info.reply ! STATE_FAIL, 0;
                             :: else ->
                                     m_qStatement ! till_userid;
                                     till_info.reply ! STATE_OK, 0;
                                  fi;
                         /* account balance */
                         :: (till_req == BALANCE) ->
                             till_userid = msgdata1;
                             if
                             :: (till_userid >= USER_NUM) ->
                                 till_info.reply ! BALANCE_FAIL, 0;
                             :: (m_dbavailable == FALSE ) ->
                                 till_info.reply ! BALANCE_FAIL, 0;
                             :: else ->
                         atomic {
                                msgdata2 = BALANCE_OK;
                                msgdata1 = m_users[till_userid].balance;
                         };
                         till_info.reply ! msgdata2, msgdata1;
                         fi;
                   ::else -> skip;
                   fi;
             fi;
      od;
}

init {
      byte n = 0;

      do
      :: n < USER_NUM ->
            m_users[n].uid = n;
            m_users[n].upwd = 'a' + n;
            m_users[n].balance = INIT_BALANCE;
            m_users[n].daily_limit = DAILY_LIMIT;
            m_users[n].tmp = 0;

            n = n + 1;
      :: else -> break;
      od;

      n = 0;

      do
      :: n < TILL_NUM ->
          m_tills[n].id = n;
          m_tills[n].status = TILL_IDLE;
          m_tills[n].till_req = LOGIN;
            atomic{
                  run till(m_tills[n]);
                  run control(m_tills[n]);
```

```
            }
            n = n + 1;
      :: else -> break;
      od;

      run day_timer();

}
```