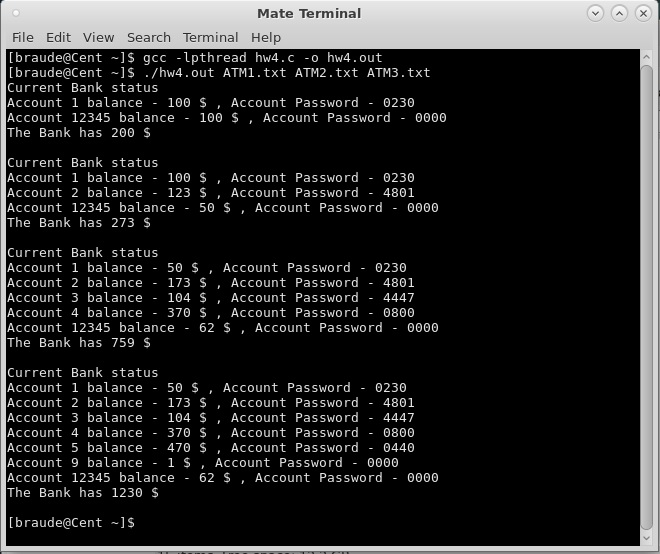
**תרגיל בית 4 – מערכות הפעלה**

מגישים:

אריק סקיגין ת.ז

אור אליהו ת.ז

תצלום מסך



קבצי קלט

ATM1.txt

O 12345 0000 100

W 12345 0000 50

D 12345 0000 12

O 12345 1234 45

ATM2.txt

O 1 0230 100

O 2 4801 123

O 3 4447 104

O 4 0800 370

O 5 0440 470

O 9 0000 970

ATM3.txt

W 777 2313 1000

B 1 0230

Q 9 0000

T 1 0230 2 50

T 9 0000 2 52

O 9 0000 1

Q 8 0222

W 777 7777 50

קובץ פלט

log.txt

2: New account id is 1 with password 0230 and initial balance 100

Error 3: Your transaction failed - account id 777 does not exists

1: New account id is 12345 with password 0000 and initial balance 100

3: Account 1 balance is 100

2: New account id is 2 with password 4801 and initial balance 123

1: Account 12345 new balance is 50 after amount 50 $ was withdrew

Error 3: Your transaction failed - account id 9 does not exists

2: New account id is 3 with password 4447 and initial balance 104

3: Transfer 50 from account 1 to account 2 new account balance is 50 new target account balance is 173

1: Account 12345 new balance is 62 after amount 12 $ was deposited

2: New account id is 4 with password 0800 and initial balance 370

Error 3: Your transaction failed - account id 9 does not exists

Error 1: Your transaction failed - account with the same id exists

2: New account id is 5 with password 0440 and initial balance 470

3: New account id is 9 with password 0000 and initial balance 1

Error 2: Your transaction failed - account with the same id exists

Error 3: Your transaction failed - account id 8 does not exists

Error 3: Your transaction failed - account id 777 does not exists

קוד התוכנית

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <string.h>

#include <semaphore.h>

#include <unistd.h>

#include <pthread.h>

//HW4 - Operating Systems

//Arik skigin 312360449

//Or Eliyahu 307884890

//LOCK ACCOUNT FUNCTION///

#define ACN\_LOCK(a) &(a->data).lock

#define MIN\_ACN\_LOCK(a,b) (((a->data.id)<(b->data.id))? ACN\_LOCK(a) : ACN\_LOCK(b))

#define MAX\_ACN\_LOCK(a,b) (((a->data.id)>(b->data.id))? ACN\_LOCK(a) : ACN\_LOCK(b))

//////////LOG/PRINT STRING//////////

#define LOG\_ACCOUNT\_EXISTS "Error %d: Your transaction failed - account with the same id exists\n"

#define LOG\_ACCOUNT\_NOTEXISTS "Error %d: Your transaction failed - account id %d does not exists\n"

#define LOG\_ACCOUNT\_WRONG\_PASSWORD "Error %d: Your transaction failed - password for account id %d is incorrect\n"

#define LOG\_ACCOUNT\_OPENED\_ACCOUNT "%d: New account id is %d with password %s and initial balance %d\n"

#define LOG\_ACCOUNT\_DESPOSITED\_ACCOUNT "%d: Account %d new balance is %d after amount %d $ was deposited\n"

#define LOG\_ACCOUNT\_DRAWN\_ACCOUNT "%d: Account %d new balance is %d after amount %d $ was withdrew\n"

#define LOG\_ACCOUNT\_BALANCED\_ACCOUNT "%d: Account %d balance is %d\n"

#define LOG\_ACCOUNT\_CLOSED\_ACCOUNT "%d: Account %d is now closed. Balance was %d\n"

#define LOG\_ACCOUNT\_TRANSFERRED\_ACCOUNT "%d: Transfer %d from account %d to account %d new account balance is %d new target account balance is %d\n"

#define LOG\_ACCOUNT\_LOW\_BALANCE "Error %d: Your transaction failed - account id %d balance is lower than %d\n"

#define BANK\_PRINT\_TITLE "Current Bank status\n"

#define BANK\_PRINT\_ACCOUNT "Account %d balance - %d $ , Account Password - %s\n"

#define BANK\_AMOUNT\_BALANCES "The Bank has %d $\n\n"

//General Settings///

#define BANK\_LOGFILE "log.txt"

#define ACCOUNT\_PASSWORD\_LENGTH 4

#define FILE\_WRITE\_BUFFER 256

enum BOOL {TRUE = 1, FALSE = 0};

enum CRITICAL\_ERRORS {ERR\_NIL=-1, ERR\_ALLOCATION=-2, ERR\_NINPUT=-3, ERR\_FILEOPEN=-4, ERR\_FILECLOSE=-5, ERR\_CREATETHREAD=-6, ERR\_LINEOVERFLOW=-7, ERR\_ATM\_INVALIDCMD=-8, ERR\_FILEWRITE=-9 };

enum ATM\_ACTION { ACCOUNT\_OPEN = 'O', ACCOUNT\_DEPOSIT = 'D', ACCOUNT\_WITHDRAW = 'W', ACCOUNT\_BALANCE = 'B', ACCOUNT\_CLOSE = 'Q', ACCOUNT\_TRANSFER = 'T'};

enum ATM\_ERRORS {ATM\_SUCCESS=0, ATM\_INVALIDCMD, ATM\_ACCOUNT\_NOTEXISTS, ATM\_ACCOUNT\_EXISTS, ATM\_ACCOUNT\_WRONG\_PASSWORD, ATN\_ACCOUNT\_NOBALANCE};

//WRITER-READER LOCK STRUCTURE

typedef struct WR\_LOCK{

int readc;

sem\_t read\_lock, write\_lock;

}WR\_LOCK;

//Account details(data) with lock

typedef struct Account\_Details{

int id;

char password[ACCOUNT\_PASSWORD\_LENGTH+1];

int balance;

WR\_LOCK lock;

}Account\_Details;

//Account node in bank accounts

typedef struct Account\_Node{

Account\_Details data;

struct Account\_Node \*next,\*perv;

}Account\_Node;

//Bank accounts with lock

typedef struct Bank {

Account\_Node \*acs;

pthread\_t tBank;

WR\_LOCK lock;

}Bank;

//Local ATM

typedef struct Local\_ATM{

int id;

pthread\_t tATM;

char \*file;

int hr\_file;

}Local\_ATM;

//ATMs

typedef struct ATMs{

Local\_ATM \*atm;

int size;

}ATMs;

//LOG FILE

typedef struct LOG\_FILE{

int hw\_file;

sem\_t write\_lock;

}LOG\_FILE;

//Threads of ATMs and Bank

void\* ATM(void\* latm);

void\* BANK();

//Initialization

void BANK\_init();

void ATMs\_init(Local\_ATM \*latms, char \*\*files, int size);

enum CRITICAL\_ERRORS LOG\_FILE\_init();

//Log

enum CRITICAL\_ERRORS LOG\_FILE\_close();

enum CRITICAL\_ERRORS LOG\_FILE\_write(char \*buffer);

//WR initialization&locks

void WR\_Lock\_init(WR\_LOCK\* lock);

void WR\_startRead(WR\_LOCK\* lock);

void WR\_endRead(WR\_LOCK\* lock);

void WR\_startWrite(WR\_LOCK\* lock);

void WR\_endWrite(WR\_LOCK\* lock);

//Troubleshooting

void CRITICAL\_ERROR\_EXIT(int THRD\_HAND\_ID);

enum BOOL CRITICAL\_ERROR(int THRD\_HAND\_ID, enum CRITICAL\_ERRORS err, char\* fname);

void BANK\_deleteAccounts();

//ATM/BANK auxiliary functions

void ATM\_writeToLOG(enum ATM\_ACTION action, enum ATM\_ERRORS err, int t1, int t2, int t3, int t4, int t5, int t6, char\*t7);

enum ATM\_ACTION actionProcessing(char\* cmd, int \*actID, char pass[ACCOUNT\_PASSWORD\_LENGTH+1], int \*t1, int \*t2);

enum ATM\_ERRORS ATM\_Execution(int atmID, char\* cmd);

Account\_Node\* BANK\_getAccount(int actID);

Account\_Node\* ACCOUNT\_create(int actID, char \*password, int init\_amount);

enum BOOL BANK\_existsACN(Account\_Node \*\*acn, int atmID, int actID, void(\*endLock)(WR\_LOCK\*), enum BOOL existsError);

enum ATM\_ERRORS BANK\_existsAndPass(Account\_Node \*\*acn, int atmID, int actID, char\*password, void(\*endLock)(WR\_LOCK\*));

//ACTIONS FUNCTIONS

enum ATM\_ERRORS BANK\_addAccount(int atmID, int actID, char \*password, int init\_amount);

enum ATM\_ERRORS BANK\_depositAccount(int atmID, int actID, char\*password, int amount);

enum ATM\_ERRORS BANK\_withdrawAccount(int atmID, int actID, char\*password, int amount);

enum ATM\_ERRORS BANK\_checkBalanceAccount(int atmID, int actID, char\*password);

enum ATM\_ERRORS BANK\_deleteAccount(int atmID, int actID, char\*password);

enum ATM\_ERRORS BANK\_transferFTAccount(int atmID, int actID, char\*password, int t\_actID, int amount);

//GLOBAL

ATMs atms;

LOG\_FILE \_log;

Bank bnk;

int main(int argc, char\* argv[]){

int i, ans[argc];

Local\_ATM latms[argc-1]; //create ATMs

if(argc == 1) CRITICAL\_ERROR(0, ERR\_NINPUT, "main");

CRITICAL\_ERROR(0, LOG\_FILE\_init(), "main"); //try to open log file

BANK\_init();

ATMs\_init(latms, argv+1 , argc-1);

ans[0]=pthread\_create(&bnk.tBank, NULL, BANK, NULL);

for(i=0; i<atms.size; i++)

ans[i+1]=pthread\_create(&atms.atm[i].tATM, NULL, ATM, &atms.atm[i]);

for(i=0; i<argc; i++)

if(ans[i]) CRITICAL\_ERROR(0, ERR\_CREATETHREAD, "main");;

for(i=0; i<atms.size; i++) pthread\_join(atms.atm[i].tATM, NULL);

BANK\_deleteAccounts();

CRITICAL\_ERROR(0, LOG\_FILE\_close(), "main");

return EXIT\_SUCCESS;

}

//ATM - ATM thread

//get (void \*) - local\_atm strcut(contains details about the activity)

//get (void\*) - default of thread(not important)

void\* ATM(void\* latm){

Local\_ATM \*atm = (Local\_ATM\*)latm; //cast to local\_atm

char buffer[1],line[FILE\_WRITE\_BUFFER+1];//buffers to read

int retin, indexfile = 0;

if((atm->hr\_file = open(atm->file, O\_RDONLY)) == -1) CRITICAL\_ERROR(atm->id, ERR\_FILEOPEN, "ATM THREAD");

do{

if(indexfile == sizeof(line)) CRITICAL\_ERROR(atm->id, ERR\_LINEOVERFLOW, "ATM THREAD"); //read line overflow

retin = read(atm->hr\_file, &buffer, sizeof(buffer));//read char

if ((indexfile) && (((retin) && (buffer[0] == '\n')) || (!retin))){

line[indexfile]='\0'; //end of string

indexfile=0;

//if its not only empty new line and run atm action

//if have problem of invalid command critical error take care

if(strcmp(line,"\n")) if(ATM\_Execution(atm->id, line)) CRITICAL\_ERROR(atm->id,ERR\_ATM\_INVALIDCMD,"ATM THREAD");

}

else

line[indexfile++] = buffer[0];

usleep(100\*1000); //1 microsecond = 0.001 milliseconds

}while(retin);

if(close(atm->hr\_file) == -1) CRITICAL\_ERROR(atm->id,ERR\_FILECLOSE,"ATM THREAD");

}

//BANK - BANK thread

//get (void\*) - default of thread(not important)

void\* BANK(){

Account\_Node \*acn;

int sum;

while(TRUE){

sleep(3);

//need to consistent print => need read lock of bank!!!

WR\_startRead(&bnk.lock);

acn=bnk.acs;

sum=0;

printf(BANK\_PRINT\_TITLE);

while(acn){

WR\_startRead(ACN\_LOCK(acn));

printf(BANK\_PRINT\_ACCOUNT,acn->data.id,acn->data.balance,acn->data.password);

sum+=acn->data.balance;

WR\_endRead(ACN\_LOCK(acn));

acn=acn->next;

}

printf(BANK\_AMOUNT\_BALANCES,sum);

WR\_endRead(&bnk.lock);

}

}

//BANK\_init - initialization global bank

void BANK\_init(){

bnk.acs=NULL;

bnk.tBank=0;

WR\_Lock\_init(&bnk.lock);

}

//ATMs\_init - initialization global ATM

//get (Local\_ATM \*) - address to local atms array

//get (char\*\*) - address to files string

//get (int) - number of local ATMs

void ATMs\_init(Local\_ATM \*latms, char \*\*files, int size){

int i;

atms.atm=latms;

atms.size=size;

for(i = 0; i<atms.size; i++){

atms.atm[i].id=i+1;

atms.atm[i].file=files[i]; //save only address!! (not need strcpy)

atms.atm[i].hr\_file=-1;

atms.atm[i].tATM=0;

}

}

//ATMs\_init - initialization global ATM

//get (Local\_ATM \*) - address to local atms array

//get (char\*\*) - address to files string

//get (int) - number of local ATMs

void CRITICAL\_ERROR\_EXIT(int THRD\_HAND\_ID){

int i;

if(bnk.tBank) pthread\_cancel(bnk.tBank);

for(i=0; i<atms.size; i++)

if(atms.atm[i].id!=THRD\_HAND\_ID)

if(atms.atm[i].tATM)

pthread\_cancel(atms.atm[i].tATM);

BANK\_deleteAccounts();

close(\_log.hw\_file);

for(i=0; i<atms.size; i++)

close(atms.atm[i].hr\_file);

exit(EXIT\_FAILURE);

}

//CRITICAL\_ERROR - handling critical issues for running the system

//get (int) - who called the error handler(0 is main)

//get (enum CRITICAL\_ERRORS) - the error sent

//get (char\*) - function called

//return (enum BOOL) - false if thats not critical error

enum BOOL CRITICAL\_ERROR(int THRD\_HAND\_ID, enum CRITICAL\_ERRORS err, char\* fname){

if(err == ERR\_NIL) return FALSE;

printf("CRITICAL ERROR! (THREAD NUMBER %d)\n",THRD\_HAND\_ID);

printf("Function \"%s\" ERROR:\n", fname);

switch(err){

case ERR\_NINPUT:

printf("--need to get files of ATMs\n");

break;

case ERR\_FILEOPEN:

printf("--cannot open file\n");

break;

case ERR\_FILECLOSE:

printf("--cannot close file\n");

break;

case ERR\_CREATETHREAD:

printf("--cannot create thread\n");

break;

case ERR\_ALLOCATION:

printf("-- dynamic allocation failed\n");

break;

case ERR\_ATM\_INVALIDCMD:

printf("-- Invalid ATM command file\n");

break;

case ERR\_FILEWRITE:

printf("--cannot write file\n");

break;

default:

return FALSE;

break;

}

CRITICAL\_ERROR\_EXIT(THRD\_HAND\_ID);

return TRUE;

}

//WR\_Lock\_init - initialization WRITER-READER LOCK STRUCTURE

//get (WR\_LOCK\*) - address to WE LOCK

void WR\_Lock\_init(WR\_LOCK\* lock){

lock->readc=0;

sem\_init(&(lock->read\_lock),0,1);

sem\_init(&(lock->write\_lock),0,1);

}

//WR\_startRead - start read locking

//get (WR\_LOCK\*) - address to WR LOCK

void WR\_startRead(WR\_LOCK\* lock){

sem\_wait(&(lock->read\_lock));

lock->readc++;

if(lock->readc == 1)

sem\_wait(&(lock->write\_lock));

sem\_post(&(lock->read\_lock));

}

//WR\_endRead - end read open locking

//get (WR\_LOCK\*) - address to WR LOCK

void WR\_endRead(WR\_LOCK\* lock){

sem\_wait(&(lock->read\_lock));

lock->readc--;

if(lock->readc == 0)

sem\_post(&(lock->write\_lock));

sem\_post(&(lock->read\_lock));

}

//WR\_startWrite - start write locking

//get (WR\_LOCK\*) - address to WR LOCK

void WR\_startWrite(WR\_LOCK\* lock){

sem\_wait(&(lock->write\_lock));

}

//WR\_endWrite - end write open locking

//get (WR\_LOCK\*) - address to WR LOCK

void WR\_endWrite(WR\_LOCK\* lock){

sem\_post(&(lock->write\_lock));

}

//LOG\_FILE\_init - initialization global log file

//get (enum CRITICAL\_ERRORS) - error at initialization

enum CRITICAL\_ERRORS LOG\_FILE\_init(){

if((\_log.hw\_file = open(BANK\_LOGFILE,O\_WRONLY|O\_CREAT|O\_TRUNC, 0644)) == -1) return ERR\_FILEOPEN;

sem\_init(&(\_log.write\_lock),0,1);

return ERR\_NIL;

}

//LOG\_FILE\_close - close log file

//get (enum CRITICAL\_ERRORS) - error at closing

enum CRITICAL\_ERRORS LOG\_FILE\_close(){

if(close(\_log.hw\_file) == -1) return ERR\_FILECLOSE;

\_log.hw\_file = -1;

return ERR\_NIL;

}

//LOG\_FILE\_write - write to log

//get (char\*) - buffer to write

//get (enum CRITICAL\_ERRORS) - error at writing

enum CRITICAL\_ERRORS LOG\_FILE\_write(char \*buffer){

sem\_wait(&(\_log.write\_lock));

//save the lock of log file!!! no one can write now & if have error its ok to save the lock

if(write(\_log.hw\_file, buffer, (ssize\_t)strlen(buffer)) != strlen(buffer)) return ERR\_FILEWRITE;

sem\_post(&(\_log.write\_lock));

return ERR\_NIL;

}

//ACCOUNT\_create - create new account

//get (int) - account id

//get (char\*) - buffer to write

//get (int) - initial balance

//get (Account\_Node\*) - Account node

Account\_Node\* ACCOUNT\_create(int actID, char \*password, int init\_amount){

Account\_Node\* acn = (Account\_Node\*)malloc(sizeof(Account\_Node));

if(acn){

acn->data.id=actID;

strcpy(acn->data.password,password);

acn->data.balance=init\_amount;

WR\_Lock\_init(ACN\_LOCK(acn));

acn->next=acn->perv=NULL;

}

return acn;

}

//BANK\_deleteAccounts - delete all accounts(free)

void BANK\_deleteAccounts(){

Account\_Node \*acn;

acn = bnk.acs;

while(bnk.acs){

acn = bnk.acs->next;

free(bnk.acs); //bye bye

bnk.acs = acn;

}

bnk.acs=NULL;

}

//ATM\_writeToLOG - atm preparing buffer to write to log & call to write

//get (enum ATM\_ACTION) - atm action

//get (enum ATM\_ERRORS) - atm error

//get (int...int) - parameter1-6

//get (char\*) - parameter1 string

void ATM\_writeToLOG(enum ATM\_ACTION action, enum ATM\_ERRORS err, int t1, int t2, int t3, int t4, int t5, int t6, char\*t7){

char buffer[FILE\_WRITE\_BUFFER+1];

switch(err){

case ATM\_ACCOUNT\_NOTEXISTS:

snprintf(buffer,sizeof(buffer),LOG\_ACCOUNT\_NOTEXISTS,t1,t2);

break;

case ATM\_ACCOUNT\_EXISTS:

snprintf(buffer,sizeof(buffer),LOG\_ACCOUNT\_EXISTS,t1);

break;

case ATM\_ACCOUNT\_WRONG\_PASSWORD:

snprintf(buffer,sizeof(buffer),LOG\_ACCOUNT\_WRONG\_PASSWORD,t1,t2);

break;

case ATN\_ACCOUNT\_NOBALANCE:

snprintf(buffer,sizeof(buffer),LOG\_ACCOUNT\_LOW\_BALANCE,t1,t2,t3);

break;

case ATM\_SUCCESS:

switch(action){

case ACCOUNT\_OPEN:

snprintf(buffer,sizeof(buffer),LOG\_ACCOUNT\_OPENED\_ACCOUNT,t1,t2,t7,t3);

break;

case ACCOUNT\_DEPOSIT:

snprintf(buffer,sizeof(buffer),LOG\_ACCOUNT\_DESPOSITED\_ACCOUNT,t1,t2,t3,t4);

break;

case ACCOUNT\_WITHDRAW:

snprintf(buffer,sizeof(buffer), LOG\_ACCOUNT\_DRAWN\_ACCOUNT,t1,t2,t3,t4);

break;

case ACCOUNT\_BALANCE:

snprintf(buffer,sizeof(buffer),LOG\_ACCOUNT\_BALANCED\_ACCOUNT,t1,t2,t3);

break;

case ACCOUNT\_CLOSE:

snprintf(buffer,sizeof(buffer),LOG\_ACCOUNT\_CLOSED\_ACCOUNT,t1,t2,t3);

break;

case ACCOUNT\_TRANSFER:

snprintf(buffer,sizeof(buffer),LOG\_ACCOUNT\_TRANSFERRED\_ACCOUNT,t1,t2,t3,t4,t5,t6);

break;

default:

return;

break;

}

break;

default:

return;

break;

}

CRITICAL\_ERROR(t1,LOG\_FILE\_write(buffer),"ATM\_writeToLOG"); //call to write &

}

//actionProcessing - get line and analyze to parameters

//get (char\*) - line to analyze

//get (int\*) - address to write account id

//get (char\*) - address to write account password

//get (int\*) - address to write another parameter

//get (int\*) - address to write another parameter

//get (enum ATM\_ACTION) - atm action

enum ATM\_ACTION actionProcessing(char\* cmd, int \*actID, char pass[ACCOUNT\_PASSWORD\_LENGTH+1], int \*t1, int \*t2){

char action;

switch(sscanf(cmd,"%c %d %s %d %d",&action, actID, pass, t1, t2)-1){

case 2:

if((action == ACCOUNT\_BALANCE) || (action == ACCOUNT\_CLOSE)) return action;

break;

case 3:

if((action == ACCOUNT\_OPEN) || (action == ACCOUNT\_DEPOSIT) || (action == ACCOUNT\_WITHDRAW)) return action;

break;

case 4:

if(action == ACCOUNT\_TRANSFER) return action;

break;

default:

break;

}

return ATM\_INVALIDCMD;

}

//ATM\_Execution - analyze command and execution

//get (int) - atm id(requesting action atm)

//get (char\*) - command

//get (enum ATM\_ERRORS) - atm action replay

enum ATM\_ERRORS ATM\_Execution(int atmID, char\* cmd){

int actID, t1=0, t2=0;

char password[ACCOUNT\_PASSWORD\_LENGTH+1];

switch(actionProcessing(cmd,&actID,password,&t1,&t2)){

case ACCOUNT\_OPEN:

BANK\_addAccount(atmID,actID,password,t1);

break;

case ACCOUNT\_DEPOSIT:

BANK\_depositAccount(atmID,actID,password,t1);

break;

case ACCOUNT\_WITHDRAW:

BANK\_withdrawAccount(atmID,actID,password,t1);

break;

case ACCOUNT\_BALANCE:

BANK\_checkBalanceAccount(atmID,actID,password);

break;

case ACCOUNT\_CLOSE:

BANK\_deleteAccount(atmID,actID,password);

break;

case ACCOUNT\_TRANSFER:

BANK\_transferFTAccount(atmID,actID,password,t1,t2);

break;

default:

return ATM\_INVALIDCMD;

break;

}

return ATM\_SUCCESS;

}

////////////////////////////////WARNING!!!!//////////////////////////////

/////////////////BANK ACCOUNTS CRITICAL SECTION!!!//////////////////////

//////////////////USE ONLY WHEN DB IS LOCKED!!!!!!!/////////////////////

//BANK\_getAccount - get account node from bank

//get (int) - account id

//get (Account\_Node\*) - account

Account\_Node\* BANK\_getAccount(int actID){

Account\_Node\* acn=NULL;

acn=bnk.acs;

while(acn){

if(acn->data.id==actID)

return acn;

acn=acn->next;

}

return NULL;

}

//BANK\_existsACN - check if account is exists

//get (Account\_Node \*\*) - address to write the address of account

//get (int) - atm id(requesting action atm)

//get (int) - account id

//get (void(\*endLock)(WR\_LOCK\*)) - WR FUNC

//get (enum BOOL) - error at exists/notexists

//get (enum BOOL) - true if account is exist

enum BOOL BANK\_existsACN(Account\_Node \*\*acn, int atmID, int actID, void(\*endLock)(WR\_LOCK\*), enum BOOL existsError){

\*acn=BANK\_getAccount(actID);

if((existsError) && (\*acn)){

ATM\_writeToLOG(FALSE,ATM\_ACCOUNT\_EXISTS,atmID,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE); //write to log that account is exist

endLock(&bnk.lock);

return TRUE;

}

if((!existsError) && (\*acn == NULL)){

ATM\_writeToLOG(FALSE,ATM\_ACCOUNT\_NOTEXISTS,atmID,actID,FALSE,FALSE,FALSE,FALSE,FALSE); //write to log that account is exist

endLock(&bnk.lock);

return FALSE;

}

return (\*acn) ? TRUE : FALSE;

}

//BANK\_existsAndPass - check account exists and password

//get (Account\_Node \*\*) - address to write the address of account

//get (int) - atm id(requesting action atm)

//get (int) - account id

//get (char\*) - account password

//get (void(\*endLock)(WR\_LOCK\*)) - WR FUNC

//get (enum ATM\_ERRORS) - error if not exists account or wrong password

enum ATM\_ERRORS BANK\_existsAndPass(Account\_Node \*\*acn, int atmID, int actID, char\*password, void(\*endLock)(WR\_LOCK\*)){

if(BANK\_existsACN(acn,atmID,actID,endLock,FALSE) == FALSE) return ATM\_ACCOUNT\_NOTEXISTS;

//CHECK PASSWORD IS NOT CRITICAL SECTION FOR ACCOUNT DATA

//Assuming there is no password change

//CHECK PASSWORD IS CRITICAL SECTION FOR BANK ACCOUNTS!!!

if(strcmp((\*acn)->data.password,password)){

ATM\_writeToLOG(FALSE,ATM\_ACCOUNT\_WRONG\_PASSWORD,atmID,actID,FALSE,FALSE,FALSE,FALSE,FALSE); //write to log that the password is wrong

endLock(&bnk.lock);

return ATM\_ACCOUNT\_WRONG\_PASSWORD;

}

return ATM\_SUCCESS;

}

////////////////////////////////END WARNING!!!!/////////////////////////////////////

////////////////////////////////ATM ACTIONS////////////////////////////////////////

//BANK\_addAccount - add new account to bank

//get (int) - atm id(requesting action atm)

//get (int) - account id

//get (char\*) - account password

//get (int) - initial balance

//get (enum ATM\_ERRORS) - atm action replay

enum ATM\_ERRORS BANK\_addAccount(int atmID, int actID, char \*password, int init\_amount){

Account\_Node \*acn\_cur, \*acn;

WR\_startWrite(&bnk.lock); //BANK DB WRITER LOCK

if(BANK\_existsACN(&acn,atmID,actID,WR\_endWrite,TRUE)) return ATM\_ACCOUNT\_EXISTS;

//save the lock of bank!!! no one can change

if((acn=ACCOUNT\_create(actID,password,init\_amount)) == NULL) CRITICAL\_ERROR(atmID,ERR\_ALLOCATION,"BANK\_addAccount");

if(bnk.acs == NULL){

bnk.acs=acn;

}else{

if(bnk.acs->data.id>acn->data.id){

acn->next=bnk.acs;

bnk.acs->perv=acn;

bnk.acs=acn;

}else{

acn\_cur=bnk.acs;

while((acn\_cur->next) && (acn\_cur->next->data.id<acn->data.id))

acn\_cur=acn\_cur->next;

acn->next=acn\_cur->next;

if(acn\_cur->next)

acn\_cur->next->perv=acn;

acn\_cur->next=acn;

acn->perv=acn\_cur;

}

}

sleep(1);

ATM\_writeToLOG(ACCOUNT\_OPEN,ATM\_SUCCESS,atmID,actID,init\_amount,FALSE,FALSE,FALSE,password); //write to log

WR\_endWrite(&bnk.lock); //BANK DB RELEASE WRITER LOCK

return ATM\_SUCCESS;

}

//BANK\_depositAccount - deposit to account

//get (int) - atm id(requesting action atm)

//get (int) - account id

//get (char\*) - account password

//get (int) - amount to deposit

//get (enum ATM\_ERRORS) - atm action replay

enum ATM\_ERRORS BANK\_depositAccount(int atmID, int actID, char\*password, int amount){

Account\_Node \*acn;

enum ATM\_ERRORS err;

WR\_startRead(&bnk.lock); //BANK DB READER LOCK

if(err = BANK\_existsAndPass(&acn,atmID,actID,password,WR\_endRead)) return err;

WR\_startWrite(ACN\_LOCK(acn)); //ACCOUNT DATA WRITE LOCK

acn->data.balance+=amount;

sleep(1);

ATM\_writeToLOG(ACCOUNT\_DEPOSIT,ATM\_SUCCESS,atmID,actID,acn->data.balance,amount,FALSE,FALSE,FALSE); //write to log

WR\_endWrite(ACN\_LOCK(acn));//ACCOUNT DATA RELEASE WRITE LOCK

WR\_endRead(&bnk.lock); //BANK DB RELEASE READER LOCK

return ATM\_SUCCESS;

}

//BANK\_withdrawAccount - withdraw from account

//get (int) - atm id(requesting action atm)

//get (int) - account id

//get (char\*) - account password

//get (int) - amount to withdraw

//get (enum ATM\_ERRORS) - atm action replay

enum ATM\_ERRORS BANK\_withdrawAccount(int atmID, int actID, char\*password, int amount){

Account\_Node \*acn;

enum ATM\_ERRORS err;

WR\_startRead(&bnk.lock); //BANK DB READER LOCK

if(err = BANK\_existsAndPass(&acn,atmID,actID,password,WR\_endRead)) return err;

WR\_startWrite(ACN\_LOCK(acn)); //ACCOUNT DATA WRITE LOCK

if(acn->data.balance<amount){

ATM\_writeToLOG(ACCOUNT\_WITHDRAW,ATN\_ACCOUNT\_NOBALANCE,atmID,actID,amount,FALSE,FALSE,FALSE,FALSE); //write to log(error balance)

WR\_endWrite(ACN\_LOCK(acn)); //ACCOUNT DATA RELEASE WRITE LOCK

WR\_endRead(&bnk.lock); //BANK DB RELEASE READER LOCK

return ATN\_ACCOUNT\_NOBALANCE;

}

acn->data.balance-=amount;

sleep(1);

ATM\_writeToLOG(ACCOUNT\_WITHDRAW,ATM\_SUCCESS,atmID,actID,acn->data.balance,amount,FALSE,FALSE,FALSE); //write to log

WR\_endWrite(ACN\_LOCK(acn)); //ACCOUNT DATA RELEASE WRITE LOCK

WR\_endRead(&bnk.lock); //BANK DB RELEASE READER LOCK

return ATM\_SUCCESS;

}

//BANK\_checkBalanceAccount - account check balance

//get (int) - atm id(requesting action atm)

//get (int) - account id

//get (char\*) - account password

//get (enum ATM\_ERRORS) - atm action replay

enum ATM\_ERRORS BANK\_checkBalanceAccount(int atmID, int actID, char\*password){

Account\_Node \*acn;

enum ATM\_ERRORS err;

WR\_startRead(&bnk.lock); //BANK DB READER LOCK

if(err = BANK\_existsAndPass(&acn,atmID,actID,password,WR\_endRead)) return err;

WR\_startRead(ACN\_LOCK(acn)); //ACCOUNT DATA READER LOCK

sleep(1);

ATM\_writeToLOG(ACCOUNT\_BALANCE,ATM\_SUCCESS,atmID,actID,acn->data.balance,FALSE,FALSE,FALSE,FALSE); //write to log

WR\_endRead(ACN\_LOCK(acn)); //ACCOUNT DATA RELEASE READER LOCK

WR\_endRead(&bnk.lock); //BANK DB RELEASE READER LOCK

return ATM\_SUCCESS;

}

//BANK\_deleteAccount - delete account from bank

//get (int) - atm id(requesting action atm)

//get (int) - account id

//get (char\*) - account password

//get (enum ATM\_ERRORS) - atm action replay

enum ATM\_ERRORS BANK\_deleteAccount(int atmID, int actID, char\*password){

Account\_Node \*acn;

int tmpBalance;

enum ATM\_ERRORS err;

WR\_startWrite(&bnk.lock); //BANK DB WRITER LOCK

if(err = BANK\_existsAndPass(&acn,atmID,actID,password,WR\_endWrite)) return err;

WR\_startWrite(ACN\_LOCK(acn));//ACCOUNT DATA WRITE LOCK

if(bnk.acs == acn){

bnk.acs=acn->next;

bnk.acs->perv=NULL;

}

else{

acn->perv->next=acn->next;

if(acn->next)

acn->next->perv=acn->perv;

}

tmpBalance=acn->data.balance;

free(acn);

sleep(1);

ATM\_writeToLOG(ACCOUNT\_CLOSE,ATM\_SUCCESS,atmID,actID,tmpBalance,FALSE,FALSE,FALSE,FALSE); //write to log

//Not releasing the lock of account

WR\_endWrite(&bnk.lock); //BANK DB RELEASE WRITE LOCK

return ATM\_SUCCESS;

}

//BANK\_transferFTAccount - transfer between accounts

//get (int) - atm id(requesting action atm)

//get (int) - account id

//get (char\*) - account password

//get (int) - target account id

//get (int) - amount to transfer

//get (enum ATM\_ERRORS) - atm action replay

enum ATM\_ERRORS BANK\_transferFTAccount(int atmID, int actID, char\*password, int t\_actID, int amount){

Account\_Node \*acn, \*t\_acn;

enum ATM\_ERRORS err;

WR\_startRead(&bnk.lock); //BANK DB READER LOCK

if(err = BANK\_existsAndPass(&acn,atmID,actID,password,WR\_endRead)) return err;

if(BANK\_existsACN(&t\_acn,atmID,t\_actID,WR\_endRead,FALSE) == FALSE) return ATM\_ACCOUNT\_NOTEXISTS;

WR\_startWrite(MIN\_ACN\_LOCK(acn,t\_acn)); //ACCOUNT DATA WRITER LOCK

WR\_startWrite(MAX\_ACN\_LOCK(acn,t\_acn)); //ACCOUNT DATA WRITER LOCK

if(acn->data.balance<amount){

ATM\_writeToLOG(ACCOUNT\_TRANSFER,ATN\_ACCOUNT\_NOBALANCE,atmID,actID,amount,FALSE,FALSE,FALSE,FALSE); //write to log(error balance)

WR\_endWrite(MIN\_ACN\_LOCK(acn,t\_acn));//ACCOUNT DATA RELEASE WRITER LOCK

WR\_endWrite(MAX\_ACN\_LOCK(acn,t\_acn));//ACCOUNT DATA RELEASE WRITER LOCK

WR\_endRead(&bnk.lock);//BANK DB RELEASE READER LOCK

return ATN\_ACCOUNT\_NOBALANCE;

}

acn->data.balance-=amount;

t\_acn->data.balance+=amount;

sleep(1);

ATM\_writeToLOG(ACCOUNT\_TRANSFER,ATM\_SUCCESS,atmID,amount,actID,t\_actID,acn->data.balance,t\_acn->data.balance,FALSE); //write to log

WR\_endWrite(MIN\_ACN\_LOCK(acn,t\_acn)); //ACCOUNT DATA RELEASE WRITER LOCK

WR\_endWrite(MAX\_ACN\_LOCK(acn,t\_acn)); //ACCOUNT DATA RELEASE WRITER LOCK

WR\_endRead(&bnk.lock); //BANK DB RELEASE READER LOCK

return ATM\_SUCCESS;

}