Санкт-Петербургский политехнический университет Петра Великого
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Отчёт по курсовой работе

Дисциплина: Операционные системы

Тема: Банковская система

Выполнил студент гр. 3530901/80203 _____ А.С.Джеус

Преподаватель _____ И.С.Егорова

2020 г.

Санкт-Петербург
2020

Цель работы

Разработать модель банковской системы. Сервер создает отдельные потоки для интерактивной обработки запросов каждого клиента, основной поток занимается прослушиванием запросов. Разделяемый ресурс — счет. Клиент может создать новый аккаунт или зайти в уже имеющийся, также может завести новую карту или выбрать уже имеющуюся, посмотреть сумму на счете, снять или внести деньги.

Ход работы

Для выполнения задания использовался язык Python. Были написаны следующие скрипты (код приведен в Приложении):

1. *Client.py*: создается сокет, по которому происходит подключение к серверу; после установки соединения клиент может посылать соответствующие команды на сервер и получать результат их выполнения.
2. *Server.py*: создает сокет для установления связи с несколькими клиентами; читает базу данных с информацией об уже зарегистрированных клиентах; после подключения очередного клиента создает для работы с ним отдельный поток; перед завершением скрипт дожидается окончания работы всех запущенных потоков и обновляет базу данных на диске.
3. *Database.py*: класс, который получает в качестве входного значения имя файла с информацией о клиентах (db.json), читает эту информацию и на ее основе создается объекты класса Account; в классе есть методы для поиска уже существующих аккаунтов и для добавления новых, а также метод для сохранения измененной базы данных на диск.

4. *Account.py*: класс, который получает в качестве входных значений идентификатор и имя клиента, а также информацию о всех его картах; в классе есть методы для поиска уже существующих карт и для добавления новых.
5. *Card.py*: класс, который получает в качестве входных значений информацию по конкретной карте (id, pin code, money); в классе есть методы для получения остатка на счете, внесения и снятия денежных средств (предусмотрен синхронизированный доступ к данным с помощью threading.Lock()).
6. *BankSystemThread.py*: класс, который наследуется от стандартного класса threading.Thread, для того чтобы работать в новом потоке; в качестве входных данных получает сокет для общения с клиентом и базу данных с информацией о клиентах; выполняет команды клиента.
7. *Tests.py*: запускает сервер, подключается к нему и посылает тестовые команды, после чего проверяет результат их выполнения на правильность.

Скриншоты с примерами работы приведены ниже:

```
andj@MacBook-Air BankSystem % python3 Server.py
Ctrl-C to exit
```

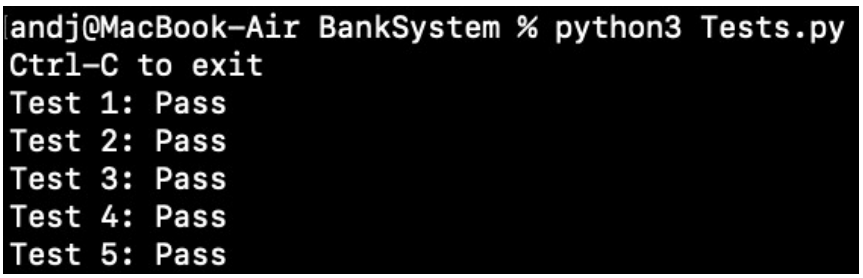
Сервер

```
andj@MacBook-Air BankSystem % python3 Client.py
Ctrl-C to exit
1) sign in <Client ID>
2) sign up <Client Name>
>>> sign up Ivan
Your ID is 1
You are signed in
1) select <card ID> <pin code>
2) new <pin code>
3) amount
4) get <money>
5) put <money>
>>> amount
First select your card
>>> new 1234
Card id is 1
>>> amount
0
>>> put 5
5
>>> get 10
There is no such money
```

Клиент

```
andj@MacBook-Air BankSystem % python3 Client.py
Ctrl-C to exit
1) sign in <Client ID>
2) sign up <Client Name>
>>> sign in 1
Hello, Ivan
You are signed in
1) select <card ID> <pin code>
2) new <pin code>
3) amount
4) get <money>
5) put <money>
>>> select 1 1111
Wrong password
>>> select 1 1234
Done
>>> amount
5
```

Клиент



Тесты

Благодаря многопоточности и синхронизированному доступу к общим ресурсам, возможна работа с картами одного или разных аккаунтов с нескольких терминалов одновременно.

Вывод

В ходе лабораторной работы была разработана модель банковской системы с использованием клиент-серверной архитектуры на языке Python.

Приложение

GitHub репозиторий проекта: https://github.com/MickeyMouseMouse/BankSystem.

Client.py

```
import signal
import socket
import re


def sendMessage(msg):
```

```
    try:
        client.send(msg.encode('utf-8'))
    except socket.error:
        print("Server disconnected")
        exit(0)


def receiveMessage():
    try:
        return str(client.recv(40).decode('utf-8'))
    except socket.error:
        print("Server disconnected")
        exit(0)


# actions before ending (on sigint)
def sigint_handler(sig, frame):
    client.send("exit".encode('utf-8'))
    client.close()
    exit(0)


client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.settimeout(1.5)
try:
    client.connect(("127.0.0.1", 8080))
```

```
except socket.error:
    print("Server is unavailable")
    exit(0)


signal.signal(signal.SIGINT, sigint_handler)
print("Ctrl-C to exit")


print("1) sign in <Client ID>")
print("2) sign up <Client Name>")
while True:
    line = input(">>> ")


    if re.fullmatch("sign in [0-9]+", line):
        sendMessage(line)
        name = receiveMessage()
        if name == "-1":
            print("Error occurred. Try again")
            continue
        else:
            print("Hello,", name)
            break


    if re.fullmatch("sign up [A-z]+", line):
```

```
        sendMessage(line)
        _id_ = receiveMessage()
        print("Your ID is", _id_)
        break

    print("Incorrect input")

print("You are signed in")
print("1) select <card ID> <pin code>")
print("2) new <pin code>")
print("3) amount")
print("4) get <money>")
print("5) put <money>")
while True:
    line = input(">>> ")

    if re.fullmatch("select [0-9]+ [0-9]{4}|new [0-9]{4}|amount|(get|put) [0-9]+",
line):
        sendMessage(line)
        print(receiveMessage())
    else:
        print("Incorrect input")
```

Server.py

```
import signal
import Database
import socket
import BankSystemThread


# actions before ending (on sigint)
def sigint_handler(sig, frame):
	for thread in threads:
		thread.stop_event.set()
		thread.join()


	server.close()
	database._update_()
	exit(0)


server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(("127.0.0.1", 8080))
server.listen(5)


database = Database.Database("db.json")


signal.signal(signal.SIGINT, sigint_handler)
```

```
print("Ctrl-C to exit")


threads = []
while True:
	sock, _ = server.accept()
	threads.append(BankSystemThread.BankSystemThread(sock, database))
	threads[-1].start() # start new thread for new client
```

Database.py

```
import os.path
import json
import Account


class Database:
	def __init__(self, databaseFile):
		self.databaseFile = databaseFile
		self.accounts = []
		if not os.path.exists(databaseFile): return
		with open(databaseFile, 'r') as db:
			for client in json.load(db).items():
				self.accounts.append(Account.Account(client))


	def _update_(self):
```

```
            data = {}
            for account in self.accounts:
                result = account.getData()
                data[result[0]] = result[1]
            with open(self.databaseFile, 'w') as db:
                    json.dump(data, db)


    def getAccount(self, client_ID):
            for account in self.accounts:
                if account.client_ID == client_ID:
                    return account
            return None


    def addAccount(self, client_name):
            newAccount = Account.Account((len(self.accounts) + 1,
{"client_name" : client_name, "cards" : []}))
            self.accounts.append(newAccount)
            return newAccount
```

Account.py

```
import Card


class Account:
```

```
        def __init__(self, data):
                self.client_ID = int(data[0])
                self.client_name = data[1].get("client_name")
                self.cards = []
                for card_data in data[1].get("cards"):
                        self.cards.append(Card.Card(card_data))


        def __str__(self):
                result = "_Client:_\n" + self.client_name +  ", id = " + str(self.client_ID)
+ "\n_Cards:_\n"
                for card in self.cards:
                        result += card.__str__() + "\n"
                return result


        def getData(self):
                cards_data = []
                for card in self.cards:
                        cards_data.append(card.getData())
                return [self.client_ID, {"client_name" : self.client_name, "cards" :
cards_data}]


        def getCard(self, card_ID, pin_code):
                for card in self.cards:
                        if card.card_ID == card_ID:
```

```
                        if card.pin_code == pin_code:
                                return (0, card)
                        else:
                                return (-1, None)
                return (None, None)


        def addCard(self, pin_code):
                newCard = Card.Card({"card_ID" : str(len(self.cards) + 1), "pin_code" :
pin_code, "money" : "0"})
                self.cards.append(newCard)
                return newCard
```

Card.py

```
import threading


class Card:
        def __init__(self, data):
                self.card_ID = int(data.get("card_ID"))
                self.pin_code = int(data.get("pin_code"))
                self.money = int(data.get("money"))


                self.lock = threading.Lock() # to synchronize threads
```

```
    def __str__(self):
        return  str(self.card_ID) + " " + str(self.pin_code) + " " + str(self.money)


    def getData(self):
        return {"card_ID" : str(self.card_ID), "pin_code" : str(self.pin_code),
"money" : str(self.money)}


    def getAmount(self):
        self.lock.acquire(1)


        result = self.money


        self.lock.release()
        return result


    def withdrawMoney(self, amount):
        self.lock.acquire(1)


        result = -1
        if self.money >= amount:
            self.money -= amount
            result = self.money


        self.lock.release()
```

```
            return result


        def addMoney(self, amount):
            self.lock.acquire(1)


            self.money += amount
            result = self.money


            self.lock.release()
            return result
```

BankSystemThread.py

```
import threading
import socket


class BankSystemThread(threading.Thread):
        def __init__(self, sock, db):
            threading.Thread.__init__(self)
            self.sock = sock
            self.sock.settimeout(1.5)
            self.db = db
            self.stop_event = threading.Event()
```

```
def sendMessage(self, msg):
        try:
                self.sock.send(msg.encode('utf-8'))
        except socket.error:
                self.stop_event.set()


def run(self):
        account = None
        card = None
        while True:
                if self.stop_event.is_set(): break


                try:
                        parts = str(self.sock.recv(40).decode('utf-8')).split()
                        if len(parts) == 0: # client is dead
                                self.stop_event.set()
                                continue
                except socket.timeout:
                        continue


                if (parts[0] == "exit"):
                        break
```

```
                if account == None:
                        if parts[0] == "sign":
                                if parts[1] == "in":
                                        account = self.db.getAccount(int(parts[2]))
                                        if account == None:
                                                self.sendMessage("-1")
                                        else:
                                                self.sendMessage(account.client_name)
                                else:
                                        account = self.db.addAccount(parts[2])
                                        self.sendMessage(str(account.client_ID))
                else:
                        if parts[0] == "select":
                                status, card = account.getCard(int(parts[1]),
int(parts[2]))
                                if status == None: self.sendMessage("No card with
this id")
                                if status == -1: self.sendMessage("Wrong password")
                                if status == 0: self.sendMessage("Done")

                        if parts[0] == "new":
                                card = account.addCard(parts[1])
                                self.sendMessage("Card id is " + str(card.card_ID))
```

```
                    if parts[0] == "amount":
                        if card != None:
                            self.sendMessage(str(card.getAmount()))
                        else:
                            self.sendMessage("First select your card")

                    if parts[0] == "get":
                        if card != None:
                            result = card.withdrawMoney(int(parts[1]))
                            if result == -1:
                                self.sendMessage("There is no such
money")
                            else:
                                self.sendMessage(str(result))
                        else:
                            self.sendMessage("First select your card")


                    if parts[0] == "put":
                        if card != None:
                            result = card.addMoney(int(parts[1]))
                            self.sendMessage(str(result))
                        else:
```

```
                                        self.sendMessage("First select your card")



            self.sock.close()
```

Tests.py

```
import socket
import os
import  subprocess
import time
import signal


if os.path.exists("db.json"): os.remove("db.json")


serverProcess = subprocess.Popen("python3 Server.py", shell = True) # start Server
time.sleep(2) # wait a little


client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(("127.0.0.1", 8080))


# Test 1
client.send("sign up Ivan".encode('utf-8'))
if str(client.recv(25).decode('utf-8')) == "1":
```

```
        print("Test 1: Pass")
else:
        print("Test 1: Fail")


# Test 2
client.send("amount".encode('utf-8'))
if str(client.recv(25).decode('utf-8')) == "First select your card":
        print("Test 2: Pass")
else:
        print("Test 2: Fail")


# Test 3
client.send("new 1111".encode('utf-8'))
if str(client.recv(25).decode('utf-8')) == "Card id is 1":
        print("Test 3: Pass")
else:
        print("Test 3: Fail")


# Test 4
client.send("get 10".encode('utf-8'))
if str(client.recv(25).decode('utf-8')) == "There is no such money":
        print("Test 4: Pass")
else:
```

```
        print("Test 4: Fail")


# Test 5
client.send("put 20".encode('utf-8'))
if str(client.recv(25).decode('utf-8')) == "20":
        print("Test 5: Pass")
else:
        print("Test 5: Fail")


client.send("exit".encode('utf-8'))
client.close()


try:
        os.killpg(os.getpgid(serverProcess.pid), signal.SIGINT) # stop Server
except KeyboardInterrupt:
        exit(0)
```