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 банковскую систему с тестами. 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, 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 своей работы скрипт обновляет базу данных на диске.
3. *Database.py*: класс, который получает в качестве входного значения имя файла с информацией о клиентах, 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 модифицированных данных в файл.

4. *Account.py*: класс, который получает в качестве входных значений идентификатор клиента, идентификатор карты клиента, пин код от карты и сумму денег на счете; в классе есть методы для получения остатка, снятия и внесения денежных средств (предусмотрен синхронизированный доступ к данным с помощью threading.Event()).
5. *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.
6. *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
```

```
andj@MacBook-Air BankSystem % python3 Client.py
1) sign in <Client ID> <Card ID> <Pin Code>
2) sign up <Client ID> <Pin Code>
3) exit
>>> sign in Ivan 1 1234
You are signed in
1) amount
2) get <money>
3) put <money>
4) exit
>>> amount
0
>>> get 5
Error
>>> put 8
8
>>> get 3
5
```

```
andj@MacBook-Air BankSystem % python3 Client.py
1) sign in <Client ID> <Card ID> <Pin Code>
2) sign up <Client ID> <Pin Code>
3) exit
>>> sign up Danil 1234
Your Card ID is 2
You are signed in
1) amount
2) get <money>
3) put <money>
4) exit
>>> put 10
10
>>> exit
andj@MacBook-Air BankSystem % python3 Client.py
1) sign in <Client ID> <Card ID> <Pin Code>
2) sign up <Client ID> <Pin Code>
3) exit
>>> sign in Danil 2 1234
You are signed in
1) amount
2) get <money>
3) put <money>
4) exit
>>> amont
Incorrect input
>>> amount
10
```

BankSystem — Python Client.py — 54×17

~/Downloads/BankSystem — Python Client.py

```
andj@MacBook-Air BankSystem % python3 Client.py
1) sign in <Client ID> <Card ID> <Pin Code>
2) sign up <Client ID> <Pin Code>
3) exit
>>> sign in Ivan 1 1234
You are signed in
1) amount
2) get <money>
3) put <money>
4) exit
>>> amount
55
>>> amount
60
>>> amount
20
```

BankSystem — Python Client.py — 56×17

~/Downloads/BankSystem — Python Client.py

```
andj@MacBook-Air BankSystem % python3 Client.py
1) sign in <Client ID> <Card ID> <Pin Code>
2) sign up <Client ID> <Pin Code>
3) exit
>>> sign in Ivan 1 1234
You are signed in
1) amount
2) get <money>
3) put <money>
4) exit
>>> put 50
55
>>> put 5
60
>>> get 40
20
```

```
andj@MacBook-Air BankSystem % python3 Tests.py
Ctrl-C to exit
Test 1: Pass
Test 2: Pass
Test 3: Pass
Test 4: Pass
```

Вывод

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

Приложение

Листинг 1. Client.py

```
import socket
import re


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


print("1) sign in <Client ID> <Card ID> <Pin Code>")
print("2) sign up <Client ID> <Pin Code>")
print("3) exit")
while True:
    while True:
        line = input(">>> ")


        if line == "exit":
            client.send(line.encode('utf-8'))
            client.close()
            exit(0)
```

```
        if re.fullmatch("sign (in [A-z]+ [0-9]+ [0-9]{4}|up [A-z]+ [0-9]{4})", line):
            client.send(line.encode('utf-8'))
            break

        print("Incorrect input")

    card_ID = str(client.recv(10).decode('utf-8'))
    if card_ID == "-1":
        print("Something went wrong. Try again")
    else:
        if card_ID != "0": print("Your Card ID is", card_ID)
        print("You are signed in")
        break

print("1) amount")
print("2) get <money>")
print("3) put <money>")
print("4) exit")
while True:
    line = input(">>> ")

    if line == "exit":
```

```
        client.send(line.encode('utf-8'))
        client.close()
        break

    if re.fullmatch("amount|(get|put) [0-9]+", line):
        client.send(line.encode('utf-8'))
        print(str(client.recv(40).decode('utf-8')))
    else:
        print("Incorrect input")
```

Листинг 2. Server.py

```
import threading
import os.path
import signal
import BankSystemThread
import Database
import Account
import socket


# actions before ending (on sigint)
def signal_handler(sig, frame):
        server.close()
        database.save()
```

```
        exit(0)


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


database = Database.Database("Database.txt")


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


while True:
        sock, _ = server.accept()
        BankSystemThread.BankSystemThread(sock, database).start() # start new
thread for new client
```

Листинг 3. Database.py

```
import os.path
import Account


class Database:
        def __init__(self, databaseFile):
                self.databaseFile = databaseFile
```

```
            self.data = []
            if not os.path.exists(databaseFile): return
            with open(databaseFile, 'r') as db:
                for line in db:
                    parts = line.split()
                    self.data.append(Account.Account(parts[0], parts[1],
parts[2], parts[3]))

    def save(self):
            with open(self.databaseFile, 'w') as db:
                for account in self.data:
                    db.write(account.__str__() + "\n")

    def getAccount(self, client_ID, card_ID, pin_code):
            for account in self.data:
                if account.equals(client_ID, card_ID, pin_code):
                    return account
            return None

    def addAccount(self, client_ID, pin_code):
            newAccount = Account.Account(client_ID, len(self.data) + 1, pin_code,
0)
            self.data.append(newAccount)
            return newAccount
```

Листинг 4. Account.py

```
import threading


class Account:
	def __init__(self, client_ID, card_ID, pin_code, money):
		self.client_ID = str(client_ID)
		self.card_ID = int(card_ID)
		self.pin_code = int(pin_code)
		self.money = int(money)


		self.event = threading.Event() # to synchronize threads
		self.event.set()


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


	def equals(self, client_ID, card_ID, pin_code):
		if (self.client_ID == client_ID and
			self.card_ID == card_ID and
			self.pin_code == pin_code):
			return True
		else:
```

```
                return False


    def getAmount(self):
            self.event.wait()
            self.event.clear()


            result = self.money


            self.event.set()
            return result


    def withdrawMoney(self, amount):
            self.event.wait()
            self.event.clear()


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


            self.event.set()
            return result
```

```
    def addMoney(self, amount):
        self.event.wait()
        self.event.clear()

        self.money += amount
        result = self.money

        self.event.set()
        return result
```

Листинг 5. BankSystemThread.py

```
import threading
import Database
import socket

class BankSystemThread(threading.Thread):
    def __init__(self, sock, db):
        threading.Thread.__init__(self)
        self.sock = sock
        self.db = db

    def run(self):
        account = None
```

```
            while True:
                parts = str(self.sock.recv(40).decode('utf-8')).split()

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

                if account == None:
                    if parts[0] == "sign":
                        if parts[1] == "in":
                            account = self.db.getAccount(parts[2],
int(parts[3]), int(parts[4]))
                            if account == None:
                                self.sock.send("-1".encode('utf-8'))
                            else:
                                self.sock.send("0".encode('utf-8'))
                        else:
                            account = self.db.addAccount(parts[2],
int(parts[3]))

        self.sock.send(str(account.card_ID).encode('utf-8'))
                else:
                    if parts[0] == "amount":
                        self.sock.send(str(account.getAmount()).encode('utf-
8'))
```

```
                if parts[0] == "get":
                        result = account.withdrawMoney(int(parts[1]))
                        if result == -1:
                                self.sock.send("Error".encode('utf-8'))
                        else:
                                self.sock.send(str(result).encode('utf-8'))

                if parts[0] == "put":
                        result = account.addMoney(int(parts[1]))
                        self.sock.send(str(result).encode('utf-8'))

        self.sock.close()
```

Листинг 6. Tests.py

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

if os.path.exists("Database.txt"): os.remove("Database.txt")
```

```
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 1234".encode('utf-8'))
if str(client.recv(10).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(10).decode('utf-8')) == "0":
        print("Test 2: Pass")
else:
        print("Test 2: Fail")


# Test 3
client.send("get 10".encode('utf-8'))
if str(client.recv(10).decode('utf-8')) == "Error":
```

```
        print("Test 3: Pass")
else:
        print("Test 3: Fail")


# Test 4
client.send("put 10".encode('utf-8'))
if str(client.recv(10).decode('utf-8')) == "10":
        print("Test 4: Pass")
else:
        print("Test 4: Fail")


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


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