НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ ТА СПЕЦІАЛІЗОВАНИХ КОМП’ЮТЕРНИХ СИСТЕМ

**Лабораторна робота №2  
з дисципліни «Операційні системи»**

**Варіант 9**

Виконав  
студент 4-го курсу  
групи КВ-41  
Курач Віктор

Київ – 2017

**Постановка задачі**

Автомат приймає гроші (тут тільки одного визначеного номіналу – 1 грн.) і видає здачу монетами вартістю до 1 грн. (1, 2, 5, 10, 25, 50 коп.). Сума здачі розраховується. Початкова кількість монет кожного номіналу задається і становить: 1 коп. – 50 шт., 2 коп. – 25 шт., 5 коп. – 20 шт., 10 коп. – 15 шт., 25 коп. – 10 шт., 50 коп. – 5 шт.

Задано вартість квитків: до Києва – 28 коп., до Москви – 37 коп., до Лондона – 50 коп., до Берліна – 77 коп., до Парижа – 91 коп.

Якщо здачу видати можна, програма формує потрібний набір монет для здачі (також коригує банк монет) і формує сигнал на видачу. Якщо потрібних монет для здачі не вистачає, формується відповідне повідомлення.

Вимоги на видачу грошей надходять через випадкові проміжки часу після чергового сеансу продажу або відмови.

Модель автомата представити у вигляді *двох* взаємодіючих процесів А і Б. Процес А визначає моменти надходження вимоги на продаж і потрібну суму здачі.

Процес Б очікує момент появи необхідності видати здачу і, якщо величина здачі відома, визначає потрібні монети або неможливість видачі.

Для організації доступу до подільних ресурсів використати семафори.

**Приклад роботи програми**

Moment: 1

City: Kiev

Price: 28

Rest: 72 ( 1\*50 2\*10 1\*2 )

Bank: 878

Moment: 2

City: Paris

Price: 91

Rest: 9 ( 1\*5 2\*2 )

Bank: 969

Moment: 5

City: London

Price: 50

Rest: 50 ( 1\*50 )

Bank: 1019

Moment: 7

City: Berlin

Price: 77

Rest: 23 ( 2\*10 1\*2 1\*1 )

Bank: 1096

Moment: 8

City: Moscow

Price: 37

Rest: 63 ( 1\*50 1\*10 1\*2 1\*1 )

Bank: 1133

Moment: 10

City: Berlin

Price: 77

Rest: 23 ( 2\*10 1\*2 1\*1 )

Bank: 1210

**Лістинг**

import threading  
from time import sleep  
  
semaphore = threading.BoundedSemaphore()  
  
*# Common resource*bank = {"1": 50, "2": 25, "5": 20, "10": 15, "25": 10, "50": 5, "100": 0}  
  
price\_to\_Kiev = 28  
price\_to\_Moscow = 37  
price\_to\_London = 50  
price\_to\_Berlin = 77  
price\_to\_Paris = 91  
  
purchases = ["", "Kiev", "Paris", "", "", "London", "", "Berlin", "Moscow", "", "Berlin"]  
  
*# Common resources*rest = 0  
rest\_signal = False  
stop\_signal = False  
  
  
def count\_bank():  
 res = 0  
 for x in bank.keys():  
 res += int(x) \* bank[x]  
 return str(res)  
  
  
def get\_ticket\_price(city):  
 if city == "Kiev":  
 return price\_to\_Kiev  
 if city == "Moscow":  
 return price\_to\_Moscow  
 if city == "Berlin":  
 return price\_to\_Berlin  
 if city == "Paris":  
 return price\_to\_Paris  
 if city == "London":  
 return price\_to\_London  
  
  
def count\_rest(c1, c2, c5, c10, c25, c50):  
 return c1 + c2 \* 2 + c5 \* 5 + c10 \* 10 + c25 \* 25 + c50 \* 50  
  
  
def gather\_rest():  
 c1, c2, c5, c10, c25, c50 = 0, 0, 0, 0, 0, 0  
 while bank["50"] > 0 and count\_rest(0, 0, 0, 0, 0, c50 + 1) <= rest:  
 c50 += 1  
 bank["50"] -= 1  
 while bank["25"] > 0 and count\_rest(0, 0, 0, 0, c25 + 1, c50) <= rest:  
 c25 += 1  
 bank["25"] -= 1  
 while bank["10"] > 0 and count\_rest(0, 0, 0, c10 + 1, c25, c50) <= rest:  
 c10 += 1  
 bank["10"] -= 1  
 while bank["5"] > 0 and count\_rest(0, 0, c5 + 1, c10, c25, c50) <= rest:  
 c5 += 1  
 bank["5"] -= 1  
 while bank["2"] > 0 and count\_rest(0, c2 + 1, c5, c10, c25, c50) <= rest:  
 c2 += 1  
 bank["2"] -= 1  
 while bank["1"] > 0 and count\_rest(c1 + 1, c2, c5, c10, c25, c50) <= rest:  
 c1 += 1  
 bank["1"] -= 1  
 return c1, c2, c5, c10, c25, c50  
  
  
def give\_money\_back(c1, c2, c5, c10, c25, c50):  
 bank["100"] -= 1  
 bank["50"] += c50  
 bank["25"] += c25  
 bank["10"] += c10  
 bank["5"] += c5  
 bank["2"] += c2  
 bank["1"] += c1

def get\_rest\_string(c1, c2, c5, c10, c25, c50):  
 rest\_string = str(count\_rest(c1, c2, c5, c10, c25, c50)) + " ( "  
 if c50 > 0:  
 rest\_string += str(c50) + "\*50 "  
 if c25 > 0:  
 rest\_string += str(c25) + "\*25 "  
 if c10 > 0:  
 rest\_string += str(c10) + "\*10 "  
 if c5 > 0:  
 rest\_string += str(c5) + "\*5 "  
 if c2 > 0:  
 rest\_string += str(c2) + "\*2 "  
 if c1 > 0:  
 rest\_string += str(c1) + "\*1 "  
 return rest\_string + ")"  
  
  
def sell\_ticket():  
 time = 0  
 global rest  
 global rest\_signal  
 global stop\_signal  
 while True:  
 if purchases[time] != "":  
 semaphore.acquire()  
 bank["100"] += 1  
 rest = 100 - get\_ticket\_price(purchases[time])  
 print("Moment: %s\nCity: %s\nPrice: %s"  
 % (time, purchases[time], get\_ticket\_price(purchases[time])))  
 rest\_signal = True  
 semaphore.release()  
 sleep(0.001)  
 time += 1  
 if time == len(purchases):  
 stop\_signal = True  
 break  
  
  
def give\_rest():  
 global rest  
 global rest\_signal  
 while True:  
 if rest\_signal:  
 semaphore.acquire()  
 rest\_signal = False  
 c1, c2, c5, c10, c25, c50 = gather\_rest()  
 if count\_rest(c1, c2, c5, c10, c25, c50) < rest:  
 print("Unable to give rest")  
 give\_money\_back(c1, c2, c5, c10, c25, c50)  
 else:  
 print("Rest: " + get\_rest\_string(c1, c2, c5, c10, c25, c50))  
 print("Bank: %s\n" % count\_bank())  
 semaphore.release()  
 if stop\_signal:  
 break  
  
  
thread1 = threading.Thread(target=sell\_ticket)  
thread2 = threading.Thread(target=give\_rest)  
thread1.start()  
thread2.start()  
thread1.join()  
thread2.join()