|  |  |
| --- | --- |
|  | **ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**  **ФАКУЛТЕТ КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ** |

**проект за оценка по СПр**

**Дисциплина: Системно програмиране**

**Тема: Магазинери „бъркат в касата“**

***Изготвил****:*

***Димитър Тодоров Колев***

Фак. № 381222063

Група: 91

IV курс, Киб. Сиг.

e-mail: [dimitakolev@tu-mail.bg](mailto:dimitakolev@tu-mail.bg)

***Ръководител****:*

Д. Андреев

**София, 2025**

Съдържание

[I. Анализ на изготвеното приложение 3](#_Toc193641860)

[1. Задание 3](#_Toc193641861)

[2. Анализ на заданието 3](#_Toc193641862)

[II. Функционално описание на приложението 3](#_Toc193641863)

[1. Инициализация на системата 3](#_Toc193641864)

[2. Обработка на операциите 4](#_Toc193641865)

[3. Обновяване на информацията в реално време 4](#_Toc193641866)

[4. Завършване и освобождаване на ресурси 4](#_Toc193641867)

[III. Изпълнение на функционалностите 4](#_Toc193641868)

[Основни функции на класа CashRegister 4](#_Toc193641869)

[Многопоточност и синхронизация 5](#_Toc193641870)

[Последователност на изпълнение 5](#_Toc193641871)

[IV. Експериментални данни 5](#_Toc193641872)

[1. Тестова среда 5](#_Toc193641873)

[2. Тествани сценарии 5](#_Toc193641874)

[V. Приложение 6](#_Toc193641875)

# Анализ на изготвеното приложение

1. Задание

В един магазин служителите имали неприятния навик „да бъркат в касата“ и да взимат пари назаем. Непрекъснато някой взимал някаква сума пари, друг пък връщал. Моделирайте задачата, като отчетете следните ограничения: В един момент на касата може да „работи“ само един служител. Тоест не може двама да теглят пари едновременно, не могат и да внасят едновременно;

Ако в касата няма пари, а някой иска да изтегли, той ще се дръпне встрани и ще изчака, **докато в касата дойдат достатъчно**. Когато това се случи, той ще се вмъкне отново най-отпред на опашката и ще изтегли. Имайте предвид, че информацията за служителите трябва да се обновява постоянно и също така има предварително записана такава.

1. Анализ на заданието

Поставената задачата е за управление на достъпа до касата в магазин, където служите-лите могат да теглят и внасят пари при определени условия:

* Само един служител може да извършва операция в даден момент (не могат да теглят или внасят едновременно).
* Ако няма достатъчно пари в касата, тегленето се блокира, докато не бъдат внесени средства. Ако друг оператор иска да изтегли пари от каса, които са налични, то в касата има **достатъчно пари**, за да му бъдат дадени без значение дали има хора, които чакат на опашка.
* Чакащите служители запазват реда си и първият в опашката получава приоритет, когато в касата са постъпили налични средства.
* Данните за служителите и операциите трябва да се обновяват в реално време.

Задачата има голямо житеско приложение - банкомат. Ако няма пари, потребите-лите не могат да теглят, докато банката не го зареди или даден клиент не внесе пари в банкомата, за да си захрани банковата сметка.

# Функционално описание на приложението

Приложението симулира управление на каса в магазин, като използва **многопоточност** и **синхронизация** в C++. Основната цел е да се осигури **контролиран достъп** до касата и **управление на чакащите заявки**, така че да няма конфликт при едновременни операции.

### **1. Инициализация на системата**

* Зареждане на началните данни за касата (начална наличност на средства).
* Подготовка на необходимите структури за синхронизация и управление на заявки.

### **2. Обработка на операциите**

* **Внасяне на пари:** Служителите могат да добавят суми към касата, като актуализират наличността.
* **Теглене на пари:** Служителите могат да теглят пари, ако има достатъчна наличност.
* **Управление на чакащи заявки:**
  + Ако наличността не е достатъчна, заявката се поставя в опашка.
  + Когато в касата се внесат пари, чакащите заявки се обработват по реда на постъпване (FIFO).

### **3. Обновяване на информацията в реално време**

* Балансът на касата се актуализира след всяка операция.
* Всеки служител получава обратна връзка за статуса на заявката си (успешна, отложена и т.н.).

### **4. Завършване и освобождаване на ресурси**

* Генериране на финален отчет за наличността в касата.
* Коректно освобождаване на използваните ресурси (памет, синхронизационни механизми и т.н.).

Тези функционалности гарантират **безконфликтна работа на касата**, като **предотвратяват едновременен достъп** и осигуряват **справедливо управление на чакащите заявки**.

1. Изпълнение на функционалностите

Програмата се състои от основен клас CashRegister, който управлява процесите, свързани с касата. Основната цел на този клас е да осигури **сигурен достъп** до средствата в касата чрез **многопоточност** и **синхронизация**.

### **Основни функции на класа** CashRegister

1. CashRegister(double initial\_balance)
   * Конструкторът инициализира касата с начална сума.
2. void withdraw(int employee\_id, double amount)
   * Опит за теглене на сума от касата.
   * Ако наличността е достатъчна, балансът се намалява и операцията се маркира като успешна.
   * Ако средствата не достигат, заявката се поставя в **опашка за изчакване**.
3. void deposit(int employee\_id, double amount)
   * Добавяне на сума в касата.
   * След всяко успешно внасяне се проверява **опашката с чакащи заявки** и се обработват тези, за които вече има достатъчно наличност.
4. double getBalance()
   * Връща текущата наличност в касата.
5. void viewPendingTransactions()
   * Извежда списък с чакащите тегления, които все още не могат да бъдат изпълнени поради недостатъчни средства.

### **Многопоточност и синхронизация**

За да се гарантира, че **само един служител** може да тегли или внася пари едновременно, се използват **pthread mutex-и** и **message queue (mq)** за комуникация:

* pthread\_mutex\_t предотвратява едновременен достъп до данните.
* **Message queue (mq)** се използва за предаване на заявки между потоците.
* Чакащите служители **заспиват**, докато не бъдат уведомени, че могат да извършат операцията си.

### **Последователност на изпълнение**

1. **Инициализация:**
   * Създава се обект от CashRegister с начална сума.
   * Стартират се нишки за обработка на заявки.
2. **Обработка на операции:**
   * Служителите подават заявки за **теглене** или **внасяне**.
   * Тегленето се извършва само ако има достатъчно средства.
   * Ако няма достатъчно пари, заявката отива в опашката.
3. **Синхронизация:**
   * Внасянето на пари задейства обработката на чакащите заявки.
   * Веднага след като дадена заявка може да се изпълни, служителят получава достъп до касата.
4. **Завършване:**
   * Извежда се **финален отчет** за състоянието на касата.
   * Освобождават се всички заделени ресурси (нишки, mutex-и, message queue).
5. Експериментални данни

### **1. Тестова среда**

Програмата е компилирана и изпълнена в **WSL (Windows Subsystem for Linux)** с компилатор **g++**. Тестовете са проведени с няколко паралелно работещи нишки, които извършват операции върху касата.

### **2. Тествани сценарии**

#### **Сценарий 1: Стартиране на системата**

* **Входни данни:**
  + Създаване на **CashRegister**
  + Извикване на registerSystem.start();
* **Очакван резултат:**
  + Касата стартира, готова за операции.
* **Реален резултат:**
  + Програмата успешно създава необходимите ресурси.

#### **Сценарий 2: Внасяне и теглене на пари**

##### **Операции, извършени в main():**

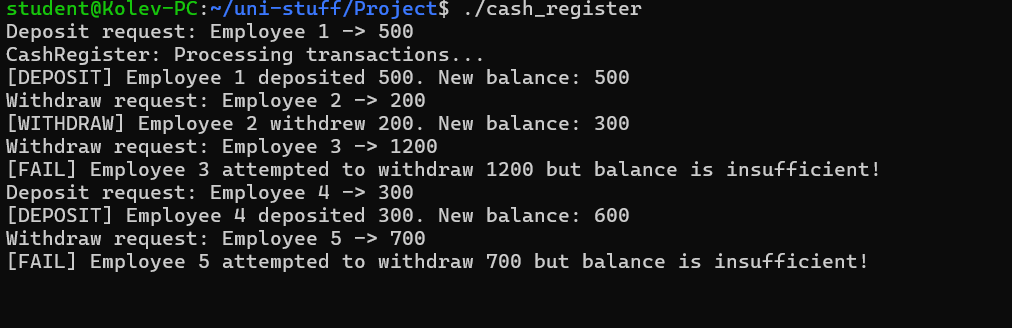
1. **Служител 1** внася **500 лв**
2. **Служител 2** тегли **200 лв**
3. **Служител 3** опитва да тегли **1200 лв** (повече от наличното)
4. **Служител 4** внася **300 лв**
5. **Служител 5** тегли **700 лв**

##### **Очакван резултат:**

* След първите две операции балансът трябва да е **300 лв** (500 - 200 = 300).
* Третата операция трябва да **блокира** (липсват средства).
* След четвъртата операция балансът трябва да е **600 лв** (300 + 300 = 600).
* Петата операция трябва да **успее** (600 - 700 = недостатъчно средства, тегленето се блокира).

##### **Реален резултат:**

* Правилно обработване на внасяния и тегления
* Заявки за теглене, които не могат да се изпълнят, влизат в опашката
* Програмата коректно актуализира баланса след всяка операция (виж фиг. 1.)



Фиг. 1. Резултати

#### **Сценарий 3: Завършване на програмата**

* **Входни данни:**
  + Извикване на registerSystem.stop();
* **Очакван резултат:**
  + Всички ресурси трябва да бъдат освободени
* **Реален резултат:**
  + Програмата приключва коректно без memory leaks

1. Приложение

Кодът на моето приложение може да бъде дотъпен чрез: <https://github.com/dimitarkole/CashRegisterManager>