Косинов Денис Константинович БПИ239.

Вариант 23.

АВС ИДЗ#4.

**Задача:**

Задача о болтунах. N болтунов имеют телефоны. Они либо некоторое (случайное) время ждут звонков, либо звонят друг другу, чтобы побеседовать. Если телефон случайного абонента занят, болтун будет звонить другому случайному абоненту, пока ему кто нибудь не ответит. Побеседовав некоторое время, болтун или ждет звонка, или звонит на другой случайный номер. Создать многопоточное приложение, моделирующее поведение болтунов. Каждый болтун моделируется отдельным потоком.

**Код программы:**

<https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/CSA_1.cpp>

#### **Сценарий поведения в терминах предметной области:**

Задача моделирует поведение группы болтунов, которые совершают звонки друг другу с использованием телефонов:

* Каждый болтун либо ждет звонка на свой телефон, либо самостоятельно пытается кому-то позвонить.
* Телефоны заняты во время разговора и освобождаются, когда разговор заканчивается.
* Если болтун хочет позвонить, но телефон адресата занят, он пытается позвонить другому. Болтун не звонит самому себе.
* Разговор между двумя болтунами происходит в течение случайного времени (1-3 секунды). После завершения разговора телефоны участников освобождаются.
* Состояние телефонов (свободен/занят) проверяется перед началом звонка, чтобы избежать конфликтов.
* Симуляция продолжается до явного завершения пользователем.

#### **Модель параллельных вычислений:**

Используемые примитивы синхронизации: программа реализована с использованием POSIX Threads и мьютексов для управления доступом к общим ресурсам (телефонам).

Параллельность: каждый болтун представлен отдельным потоком. Потоки работают независимо друг от друга, моделируя параллельное поведение болтунов.

Синхронизация:

* Для управления доступом к телефону используется мьютекс (pthread\_mutex). Телефон может быть заблокирован, если он занят разговором.
* Для вывода сообщений используется отдельный мьютекс, чтобы избежать переплетения строк при многопоточном выводе.

#### **Входные данные:**

Программа принимает два способа ввода данных:

1. **Через командную строку**:
   * -n <число\_болтунов>: задается количество болтунов (N).
   * <файл\_вывода>: имя файла, куда записываются результаты симуляции.
2. **Через конфигурационный файл**:
   * -c <имя\_файла>: конфигурационный файл, содержащий значение N на первой строке.
   * <файл\_вывода>: имя файла для записи результатов.

**Ограничения и диапазоны**:

* N > 1 (минимум два болтуна).
* Имя файла для вывода обязательно.

Пример запуска:

g++ -pthread -o program\_name CSA\_1.cpp

./program\_name -n 15 res.txt

или

./program\_name -c input.txt res2.txt

#### **Генераторы случайных чисел:**

* **Использование**:
  + Генератор случайных чисел rand() используется для определения действий болтунов и времени задержки.
  + Генератор инициализируется с помощью srand(time(nullptr) + id), чтобы избежать повторяемости последовательностей в разных потоках.
* **Диапазоны и интерпретация**:
  + Для выбора действия: rand() % 2 (результат 0 - ждать звонка, 1 - звонить).
  + Для выбора адресата: rand() % N (выбор случайного болтуна, исключая самого себя).
  + Для времени задержки (разговора или ожидания): rand() % 3 + 1 (случайное время от 1 до 3 секунд).

#### **Обобщенный алгоритм реализации программы:**

Каждый субъект предметной области (болтун и телефон) был отображен на соответствующие элементы программы, а общий сценарий поведения переведен в пошаговый алгоритм.

#### **Субъекты предметной области и их отображение в программе**

1. **Болтуны (Chatter)**:
   * Представлены потоками pthread\_t.
   * Каждому болтуну соответствует уникальный идентификатор id, который передается в поток.
   * Болтун выполняет действия (ждет звонка или совершает звонок) в бесконечном цикле до завершения симуляции (stop\_simulation).
2. **Телефоны (Phone)**:
   * Каждый телефон связан с конкретным болтуном.
   * Представлен структурой Phone, которая включает:
     + **Мьютекс** (lock) для синхронизации доступа.
     + **Флаг занятости** (is\_busy) для отслеживания состояния телефона.
3. **Вывод информации**:
   * Представлен синхронизированным механизмом вывода в консоль и файл, чтобы избежать перекрытия сообщений из нескольких потоков.

#### **Алгоритм работы программы**

1. **Инициализация**:
   * Считываются параметры симуляции:
     + Либо напрямую через аргументы командной строки.
     + Либо через конфигурационный файл.
   * Создается N объектов Phone, каждому назначается начальное состояние (is\_busy = false) и инициализируется мьютекс.
   * Инициализируется глобальный флаг stop\_simulation = false для управления завершением программы.
2. **Создание потоков**:
   * Для каждого болтуна создается поток pthread\_create().
   * Каждый поток выполняет функцию chatter\_behaviour(), моделирующую поведение болтуна.
3. **Поведение болтуна (поток)**:
   * **Выбор действия**:
     + Генерируется случайное число (rand % 2) для выбора режима:
       - **Ждать звонка**:
         1. Попытка захватить мьютекс телефона болтуна.
         2. Если телефон свободен, выводится сообщение о том, что болтун ждет звонка.
         3. Выполняется случайная пауза (rand % 3 + 1 секунд).
       - **Совершить звонок**:
         1. Выбирается случайный адресат (rand() % n), который не равен текущему болтуну.
         2. Попытка заблокировать мьютексы обоих телефонов (болтуна и адресата).
         3. Если оба телефона свободны:

Телефоны помечаются как занятые.

Выводится сообщение о начале разговора.

Выполняется пауза, моделирующая разговор (1-3 секунды).

Телефоны освобождаются, выводится сообщение о завершении разговора.

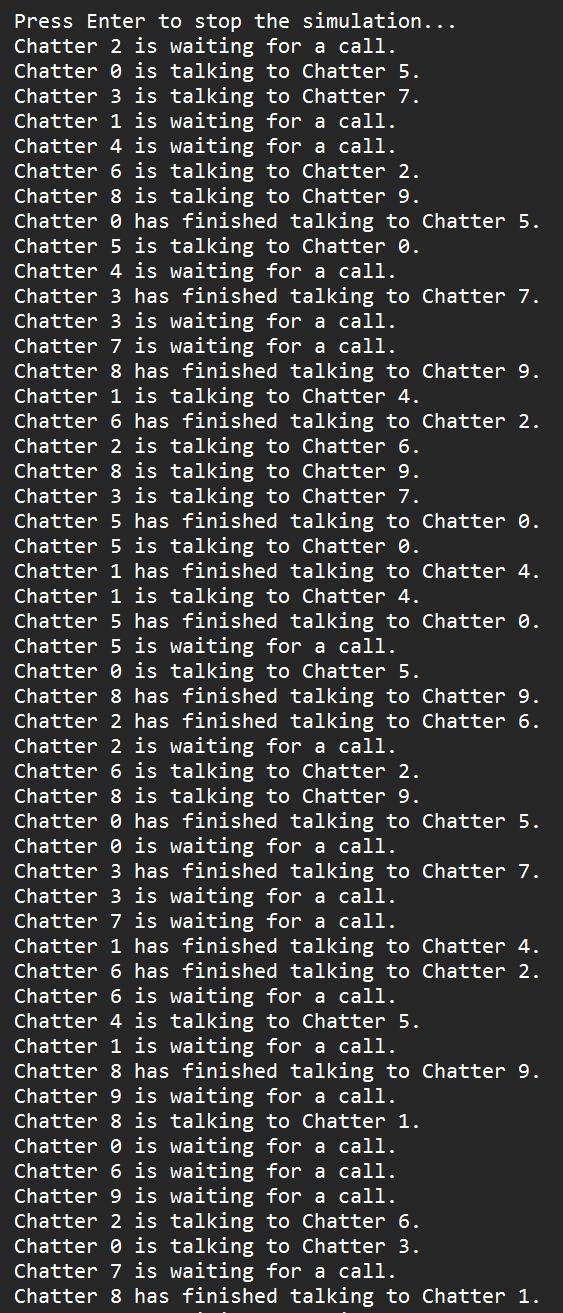
* + - * 1. Если хотя бы один из телефонов занят, мьютексы освобождаются, повторяется выбор адресата.
  + Цикл повторяется до завершения симуляции.

1. **Завершение симуляции**:
   * Пользователь нажимает Enter, глобальный флаг stop\_simulation устанавливается в true.
   * Главный поток ожидает завершения всех потоков через pthread\_join().
   * Освобождаются ресурсы:
     + Уничтожаются мьютексы телефонов и для синхронизации вывода.
     + Закрывается файл для записи результатов.
   * Выводится сообщение о завершении работы программы.

**На 4-5 баллов:**

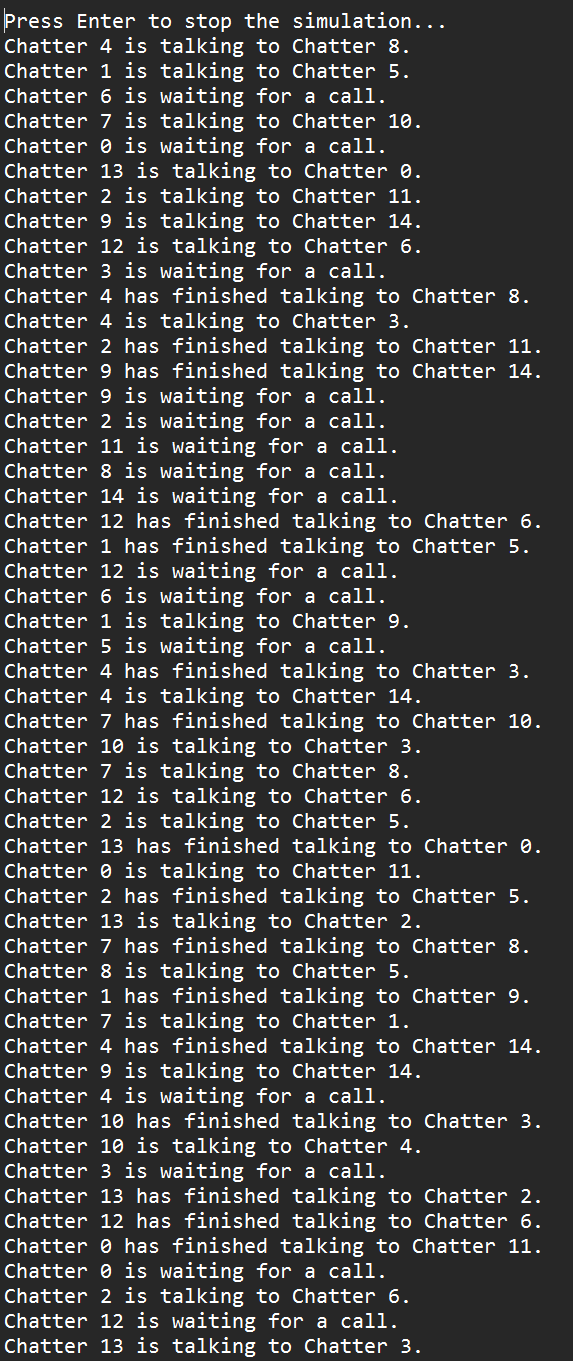
Описан сценарий поведения, модель параллельных вычислений и входные параметры программы. Выполнены все требования на оценку по коду.

**Примеры результатов работы программы:**

****

**res2.txt (https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/res2.txt)**

**(при помощи входного файла test.txt (**[**https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/test.txt**](https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/test.txt)**))**

****

**res.txt (**[**https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/res.txt**](https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/res.txt)**)**

**При входном N = 15.**

**На 6-7 баллов:**

Реализован ввод исходных данных из командной строки при запуске программы. В отчете описан обобщенный алгоритм. В программу добавлена генерация случайных данных.

**На 8 баллов:**

В программу, наряду с выводом в консоль, добавлен вывод результатов в файл. Наряду с вводом исходных данных через командную строку добавлен альтернативный вариант их ввода из файла.

Три варианта входных файлов:

<https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/testing1.txt>

<https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/testing2.txt>

<https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/testing3.txt>

Три варианта выходных файлов:

<https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/results1.txt>

<https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/results2.txt>

<https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/results3.txt>

**На 9 баллов:**

Альтернативное решение использующее другие синхропримитивы: <https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/CSA_2.cpp>

Альтернативное решение использует RWLocks (читатель/писатель блокировки) вместо мьютексов. Это позволяет разделять режим ожидания звонка (чтение) и разговор (запись).

Обе программы корректно моделируют поведение болтунов.

**Отличия в реализации:**

#### Программа на мьютексах:

* Более строгая блокировка: каждый телефон блокируется целиком для звонков и ожидания.
* Возможны краткосрочные задержки из-за конкуренции за мьютексы.

#### Программа на RWLocks:

* Чтение (ожидание звонка) может происходить параллельно для нескольких потоков.
* Запись (разговор) требует полного эксклюзивного доступа.
* Меньше блокировок для случаев ожидания звонков.

RWLocks более эффективны в сценариях, где ожидание звонков (чтение) является частой операцией.

Снижается количество конфликтов при большом количестве болтунов.

**На 10 баллов:**

Программа с использованием OpenMP:

<https://github.com/o7Techno/CSA-HW/blob/main/%D0%90%D0%92%D0%A1%20%D0%98%D0%94%D0%97%234/CSA_3.cpp>