ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук Департамент программной инженерии

Проблема обедающих философов.

Пояснительная записка

Исполнитель:
студент группы БПИ199
Вахитова Д. Р.
«13» декабря 2020 г.

Москва 2020

Оглавление:

1. Текст задания

Условие: Задача об обедающих философах.

Пять философов сидят возле круглого стола. Они проводят жизнь, чередуя приемы пищи и размышления. В центре стола находится большое блюдо спагетти. Спагетти длинные и запутанные, философам тяжело управляться с ними, поэтому каждый из них, что бы съесть порцию, должен пользоваться двумя вилками. К несчастью, философам дали только пять вилок. Между каждой парой философов лежит одна вилка, поэтому эти высококультурные и предельно вежливые люди договорились, что каждый будет пользоваться только теми вилками, которые лежат рядом с ним (слева и справа). Написать многопоточную программу, моделирующую поведение философов с помощью семафоров.

Примечание: Программа должна избегать фатальной ситуации, в которой все философы голодны, ни один из них не может взять обе вилки (например, каждый из философов держит по одной вилки и не хочет отдавать ее). Решение должно быть симметричным, то есть все потоки-философы должны выполнять один и тот же код.

2. Применяемые расчетные методы.

В чем заключается суть проблематики задачи об обедающих философах?

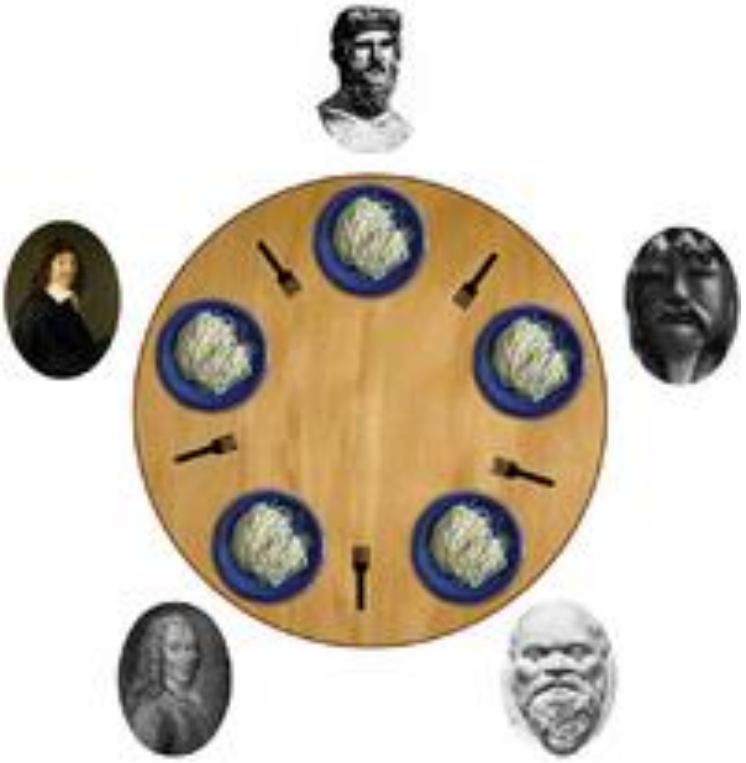

Рисунок 1 – Проблема обедающих философов.

Задача сформулирована таким образом, чтобы иллюстрировать проблему избегания взаимной блокировки (англ. deadlock) — состояния системы, при котором прогресс невозможен.

Например, можно посоветовать каждому философу выполнять следующий алгоритм:

Размышлять, пока не освободится левая вилка. Когда вилка освободится — взять её.
Размышлять, пока не освободится правая вилка. Когда вилка освободится — взять её.
Есть
Положить левую вилку
Положить правую вилку
Повторить алгоритм сначала
Это решение задачи некорректно: оно позволяет системе достичь состояния взаимной блокировки, когда каждый философ взял вилку слева и ждёт, когда вилка справа освободится.

Проблема ресурсного голодания (англ. resource starvation) может возникать независимо от взаимной блокировки, если один из философов не может завладеть левой и правой вилкой из-за проблем синхронизации. Например, может быть предложено правило, согласно которому философы должны класть вилку обратно на стол после пятиминутного ожидания доступности другой вилки, и ждать ещё пять минут перед следующей попыткой завладеть вилками. Эта схема устраняет возможность блокировки (так как система всегда может перейти в другое состояние), но по-прежнему существует возможность «зацикливания» системы (англ. livelock), при котором состояние системы меняется, но она не совершает никакой полезной работы. Например, если все пять философов появятся в столовой одновременно и каждый возьмёт левую вилку в одно и то же время, то философы будут ждать пять минут в надежде завладеть правой вилкой, потом положат левую вилку и будут ждать ещё пять минут прежде, чем попытаться завладеть вилками снова

Как решить проблему? По условию, решение должно быть оформлено через использование семафоров.

Относительно простое решение задачи достигается путём добавления официанта возле стола. Философы должны дожидаться разрешения официанта перед тем, как взять вилку. Поскольку официант знает, сколько вилок используется в данный момент, он может принимать решения

относительно распределения вилок и тем самым предотвратить взаимную блокировку философов. Если четыре вилки из пяти уже используются, то следующий философ, запросивший вилку, вынужден будет ждать разрешения официанта — которое не будет получено, пока вилка не будет освобождена. Предполагается, что философ всегда пытается сначала взять правую вилку, а потом — левую (упрощает логику).

Официантом выступает массив семафоров-вилок. Они работаю по типу частного случая семафоров – мьютексов. То есть, у каждой вилки-семафора есть 2 состояния – занята или нет, ресурс может использоваться одновременно только одним потоком-философом.

Чтобы показать, как это решение работает, предположим, что философы обозначены от А до Д по часовой стрелке. Если философы А и В едят, то заняты четыре вилки. Философ Б сидит между А и В, так что ему недоступна ни одна из вилок. В то же время, философы Г и Д имеют доступ к одной неиспользуемой вилке между ними. Предположим, что философ Г хочет есть. Если он тут же берёт свободную вилку, то становится возможна взаимная блокировка философов. Если вместо этого он спрашивает разрешения у официанта, то тот просит его подождать — и можно быть уверенным в том, что как только пара вилок освободится, то по крайней мере один философ сможет взять две вилки. Таким образом, взаимная блокировка становится невозможной.

3. Список используемых источников.

[1] Инструкция по составлению пояснительной записки [Электронный ресурс]. //URL: http://softcraft.ru/edu/comparch/tasks/mp01/ (Дата обращения: 07.12.2020, режим доступа: свободный)

[2] Статья “Потоки, блокировки и условные переменные в C++11” [Электронный ресурс]. //URL: https://habr.com/ru/post/182610/ (Дата обращения: 07.12.2020, режим доступа: свободный)

[3] Руководство std::mutex [Электронный ресурс]. //URL: https://en.cppreference.com/w/cpp/thread/mutex (Дата обращения: 07.12.2020, режим доступа: свободный)

[4] Multithreading in C++ [Электронный ресурс]. //URL: https://www.geeksforgeeks.org/multithreading-in-cpp/ (Дата обращения: 07.12.2020, режим доступа: свободный)

[5] Статья “Такие удивительные семафоры“ [Электронный ресурс]. //URL: https://habr.com/ru/post/261273/ (Дата обращения: 07.12.2020, режим доступа: свободный)

[6] Статья “Клиент-серверный процессор командной строки“ [Электронный ресурс]. //URL: https://wm-help.net/lib/b/book/3539737877/284 (Дата обращения: 07.12.2020, режим доступа: свободный)

[7] Статья «Задача об обедающих философах» [Электронный ресурс]. // URL https://en.wikipedia.org/wiki/Dining_philosophers_problem (Дата обращения: 14.12.2020 режим доступа: свободный)

4. Описание работы программы.

В программе использованы структуры из библиотеки POSIX thread, дополнительно из заголовка semaphore.h. (рис. 2)

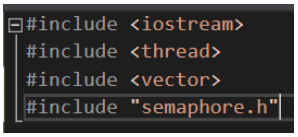

Рисунок 2 – Подключенные библиотеки

Для симуляции жизни философа используются некоторые константы.

```
// Число философов.
const int philosophers_num = 5;
// Количество раз, котоое поест каждый философ.
int eating_num;
// Время размышлений до приема пищи.
const int secsToThink = 3;
// Время приема пищи.
const int secsToEat = 2;
// Время задержки в миллисекундах после вывода текста
const int afterOutputPauseTimeMilliSec = 250;
```

Рисунок 3 – Константы, используемые в программе.

Для того, чтобы время работы программы не было бесконечным, пользователю предлагается ввести через консоль, сколько раз он хочет, чтобы философы поели. При вводе есть небольшая защита от дурака, чтобы пользователь вводил только целые числа в разумных пределах.

```
// Спрашиваем у пользователя, сколько раз он хочет, чтобы философы поели до окончания работы программы.
// Делаем небольшую "защиту от дурака".
double temp;

do {
    cout << "Введите, сколько раз вы хотите, чтобы каждый философ поел. (от 1 до 100)\n";
    cin >> temp;
    int b = static_cast<int>(temp);
    if ((double)b != temp) temp = -1;
} while (temp <= 0 || temp> 100 || static_cast<int>(temp) != temp);
eating_num = temp;
```

Рисунок 4 – Ввод количества приемов пищи.

Основной метод работает по следующему принципу (рис. 5): философы-потоки размышляют 3 секунды. После они пытаются взять вилку слева от них. Если получилось, то пытаются взять вилку справа от них. Необходимость семафора в данной программе в том, что он заставляет поток ждать, когда вилка освободится, если она занята другим философом-потоком. То есть, когда философ берет семафор-вилку, он начинает ожидание, то есть блокирует вилку для остальных потоков до того момента, пока философ-поток не поест и не вернет вилку-семафор на стол. После философ есть в течение 2 секунд, кладет вилки и снова размышляет 3 секунды.

В основной функции main создаются потоки-философы и инициализируется вектор вилок-семафоров. После, по классической схеме, запускается программа для 5 потоков-философов с последующим объединением потоков. (рис. 6)

```
// Метод, симулирующий проблему.
void eat_and_think(int id) {
    // Выбор вилки с наименьшим и наибольшим номером
    int firstForkInd = id;
    int secondForkInd = (id + 1) % philosophers_num;
    if (firstForkInd > secondForkInd)
        swap(firstForkInd, secondForkInd);
    sem_t* firstFork = forks[firstForkInd];
    sem_t* secondFork = forks[secondForkInd];

    for (int i = 0; i < eating_num; ++i) {
        printf("Философ %d размышляет.\n", id+1);
        this_thread::sleep_for(chrono::seconds(secsToThink));

        // Философ сначала берет вилку с наименьшим номером
        printf("Философ %d хочет взять первую вилку под номером %d.\n", id+1, firstForkInd+1);
        sem_wait(firstFork);
        printf("Философ %d взял первую вилку под номером %d.\n", id+1, firstForkInd+1);
        printf("Философ %d хочет взять вторую вилку под номером %d.\n", id+1, secondForkInd+1);
        sem_wait(secondFork);
        printf("Философ %d взял вторую вилку под номером %d.\n", id+1, secondForkInd+1);
        // И только потом уже берет вильку с наибольшим номером

        printf("Философ %d кушает.\n", id+1);
        this_thread::sleep_for(chrono::seconds(secsToEat));

        // Кладет сначала вилку с наибольшим номером, затем с наименьшим
        sem_post(secondFork);
        sem_post(firstFork);
        printf("Философ %d поел и продолжил рассуждать на тему бренности бытия.\n", id+1);
    }
}
```

Рисунок 5 – Метод, симулирующий проблему.

```
do {
    cout << "Введите, сколько раз вы хотите, чтобы каждый философ поел. (от 1 до 100)\n";
    cin >> temp;
    int b = static_cast<int>(temp);
    if ((double)b != temp) temp = -1;
} while (temp <= 0 || temp> 100 || static_cast<int>(temp) != temp);
eating_num = temp;
for (size_t i = 0; i < philosophers_num; i++)
{
    // Семафоры, которые по факту мьютексы.
    forks[i] = new sem_t();
    sem_init(forks[i], 1, 1);
}

thread* philosophers = new thread[philosophers_num];
for (int i = 0; i < philosophers_num; ++i) {
    philosophers[i] = thread(eat_and_think, i);
}

// Объединение потоков.
for (int i = 0; i < philosophers_num; ++i) {
    philosophers[i].join();
}

// Чистим память.
delete[] philosophers;

// Ура.
cout << "Программа закончила выполнение без взаимной блокировки. Ура-ура!" << endl;
return 0;
```

Рисунок 6 – Код main

5. Тестирование программы.

Пример 1: Обработка некорректных данных, которые может ввести пользователь.

```
drvakhitova@DESKTOP-SGFB9B1:/mnt/c/Users/User/Desktop/ABC/Microproject2/Microproject2$ g++ -std=c++11 -pthread -Wall -O2 Microproject2.cpp -o test && ./test
Введите, сколько раз вы хотите, чтобы каждый философ поел. (от 1 до 100)
-10
Введите, сколько раз вы хотите, чтобы каждый философ поел. (от 1 до 100)
0
Введите, сколько раз вы хотите, чтобы каждый философ поел. (от 1 до 100)
101
Введите, сколько раз вы хотите, чтобы каждый философ поел. (от 1 до 100)
5.5
Введите, сколько раз вы хотите, чтобы каждый философ поел. (от 1 до 100)
```

Рисунок 7 – Обход некорректного ввода.

Пример 2: 1 прием пищи.

```
drvakhitova@DESKTOP-SGFB9B1:/mnt/c/Users/User/Desktop/ABC/Microproject2/Microproject2$ g++ -std=c++11 -pthread -Wall
Введите, сколько раз вы хотите, чтобы каждый философ поел. (от 1 до 100)
1
Философ 1 размышляет.
Философ 2 размышляет.
Философ 3 размышляет.
Философ 4 размышляет.
Философ 5 размышляет.
Философ 2 хочет взять первую вилку под номером 2.
Философ 2 взял первую вилку под номером 2.
Философ 2 хочет взять вторую вилку под номером 3.
Философ 2 взял вторую вилку под номером 3.
Философ 2 кушает.
Философ 1 хочет взять первую вилку под номером 1.
Философ 1 взял первую вилку под номером 1.
Философ 4 хочет взять первую вилку под номером 4.
Философ 4 взял первую вилку под номером 4.
Философ 4 хочет взять вторую вилку под номером 5.
Философ 4 взял вторую вилку под номером 5.
Философ 4 кушает.
Философ 5 хочет взять первую вилку под номером 1.
Философ 3 хочет взять первую вилку под номером 3.
Философ 1 хочет взять вторую вилку под номером 2.
Философ 2 поел и продолжил рассуждать на тему бренности бытия.
Философ 1 взял вторую вилку под номером 2.
Философ 1 кушает.
Философ 4 поел и продолжил рассуждать на тему бренности бытия.
Философ 3 взял первую вилку под номером 3.
Философ 3 хочет взять вторую вилку под номером 4.
Философ 3 взял вторую вилку под номером 4.
Философ 3 кушает.
Философ 1 поел и продолжил рассуждать на тему бренности бытия.
Философ 5 взял первую вилку под номером 1.
Философ 5 хочет взять вторую вилку под номером 5.
Философ 5 взял вторую вилку под номером 5.
Философ 5 кушает.
Философ 3 поел и продолжил рассуждать на тему бренности бытия.
Философ 5 поел и продолжил рассуждать на тему бренности бытия.
Программа закончила выполнение без взаимной блокировки. Ура-ура!
drvakhitova@DESKTOP-SGFB9B1:/mnt/c/Users/User/Desktop/ABC/Microproject2/Microproject2$
```

Рисунок 8 – 1 прием пищи

Пример 3: Работа на 2 приемах пищи. Запускаем 2 раза, чтобы наглядно убедиться, что потоки работают по-разному.

```
drvakhitova@DESKTOP-SGFB9B1:/mnt/c/Users/User/Desktop/ABC/Microproject2/
Введите, сколько раз вы хотите, чтобы каждый философ поел. (от 1 до 100)
2
Философ 1 размышляет.
Философ 3 размышляет.
Философ 4 размышляет.
Философ 2 размышляет.
Философ 5 размышляет.
Философ 1 хочет взять первую вилку под номером 1.
Философ 1 взял первую вилку под номером 1.
Философ 1 хочет взять вторую вилку под номером 2.
Философ 1 взял вторую вилку под номером 2.
Философ 1 кушает.
Философ 3 хочет взять первую вилку под номером 3.
Философ 3 взял первую вилку под номером 3.
Философ 3 хочет взять вторую вилку под номером 4.
Философ 3 взял вторую вилку под номером 4.
Философ 3 кушает.
Философ 4 хочет взять первую вилку под номером 4.
Философ 5 хочет взять первую вилку под номером 1.
Философ 2 хочет взять первую вилку под номером 2.
Философ 2 взял первую вилку под номером 2.
Философ 2 хочет взять вторую вилку под номером 3.
Философ 1 поел и продолжил рассуждать на тему бренности бытия.
Философ 1 размышляет.
Философ 2 взял вторую вилку под номером 3.
Философ 2 кушает.
Философ 3 поел и продолжил рассуждать на тему бренности бытия.
Философ 3 размышляет.
Философ 4 взял первую вилку под номером 4.
Философ 4 хочет взять вторую вилку под номером 5.
Философ 4 взял вторую вилку под номером 5.
Философ 4 кушает.
Философ 5 взял первую вилку под номером 1.
Философ 5 хочет взять вторую вилку под номером 5.
Философ 2 поел и продолжил рассуждать на тему бренности бытия.
Философ 2 размышляет.
Философ 4 поел и продолжил рассуждать на тему бренности бытия.
```

Рисунок 9 – Запуск на 2 приемах пищи 1-ый раз, часть 1

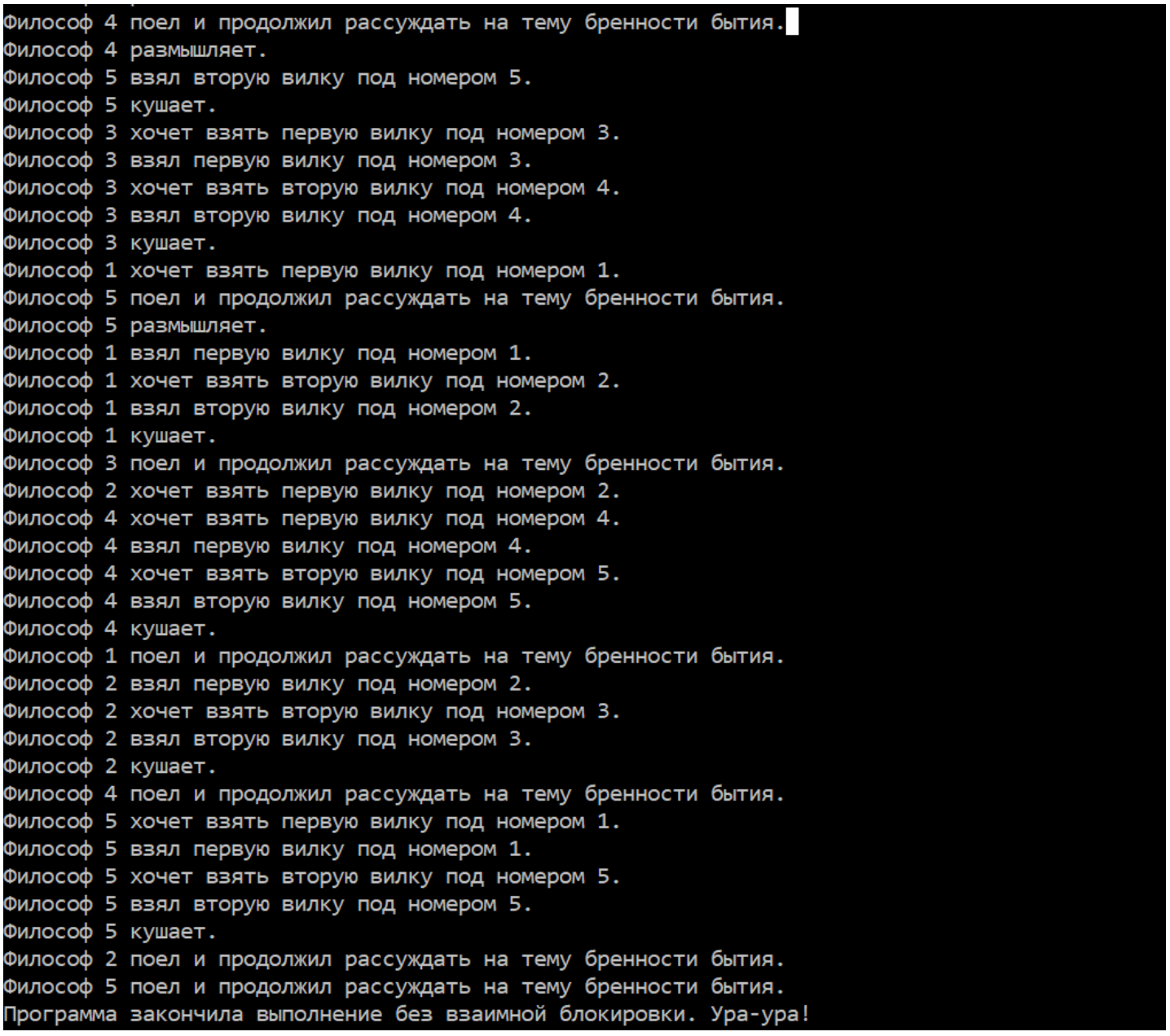

Рисунок 10 – Запуск на 2 приемах пищи 1-ый раз, часть 2

```
drvakhitova@DESKTOP-SGFB9B1:/mnt/c/Users/User/Desktop/ABC/Microproject2/Microproject
Введите, сколько раз вы хотите, чтобы каждый философ поел. (от 1 до 100)
2
Философ 1 размышляет.
Философ 3 размышляет.
Философ 4 размышляет.
Философ 2 размышляет.
Философ 5 размышляет.
Философ 5 хочет взять первую вилку под номером 1.
Философ 3 хочет взять первую вилку под номером 3.
Философ 4 хочет взять первую вилку под номером 4.
Философ 4 взял первую вилку под номером 4.
Философ 4 хочет взять вторую вилку под номером 5.
Философ 5 взял первую вилку под номером 1.
Философ 4 взял вторую вилку под номером 5.
Философ 4 кушает.
Философ 5 хочет взять вторую вилку под номером 5.
Философ 2 хочет взять первую вилку под номером 2.
Философ 2 взял первую вилку под номером 2.
Философ 2 хочет взять вторую вилку под номером 3.
Философ 1 хочет взять первую вилку под номером 1.
Философ 3 взял первую вилку под номером 3.
Философ 3 хочет взять вторую вилку под номером 4.
Философ 4 поел и продолжил рассуждать на тему бренности бытия.
Философ 4 размышляет.
Философ 5 взял вторую вилку под номером 5.
Философ 5 кушает.
Философ 3 взял вторую вилку под номером 4.
Философ 3 кушает.
Философ 3 поел и продолжил рассуждать на тему бренности бытия.
Философ 3 размышляет.
Философ 1 взял первую вилку под номером 1.
Философ 1 хочет взять вторую вилку под номером 2.
Философ 5 поел и продолжил рассуждать на тему бренности бытия.
Философ 5 размышляет.
Философ 2 взял вторую вилку под номером 3.
Философ 2 кушает.
Философ 4 хочет взять первую вилку под номером 4.
Философ 4 взял первую вилку под номером 4.
Философ 4 хочет взять вторую вилку под номером 5.
```

Рисунок 11 – Запуск на 2 приемах пищи 2-ой раз, часть 1

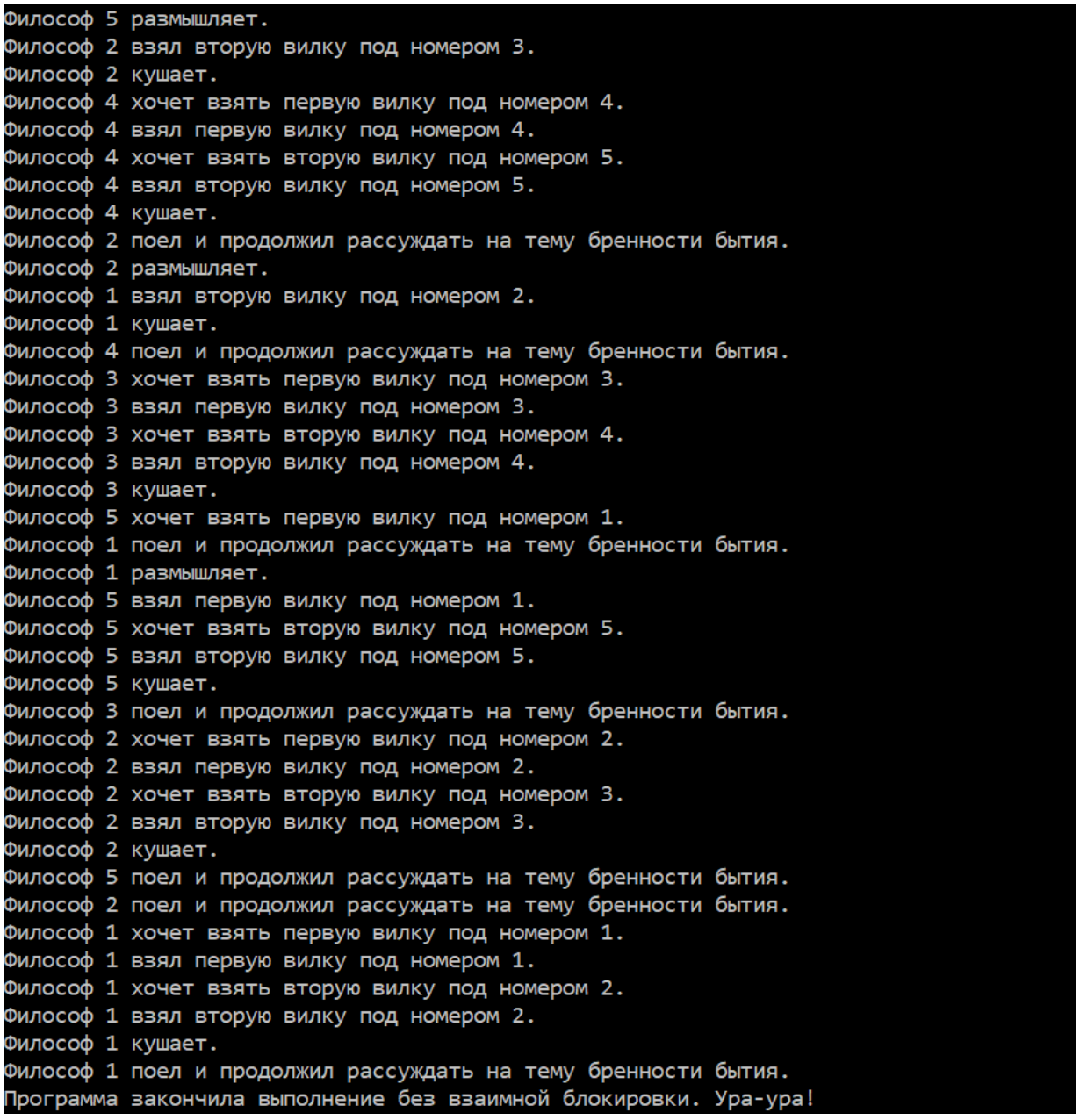

Рисунок 12 – Запуск на 2 приемах пищи 2-ой раз, часть 2

Пример 4: Запустим 100 приемов пищи и убедимся в конце, что программа отработала корректно. Всего работа для 5 философов заняла меньше 9 минут. Если на размышление + прием пищи у 1 философа уходит суммарно 5 секунд, то на сто приемов пиши у него уйдет как раз 8.3 минуты. Все круто!

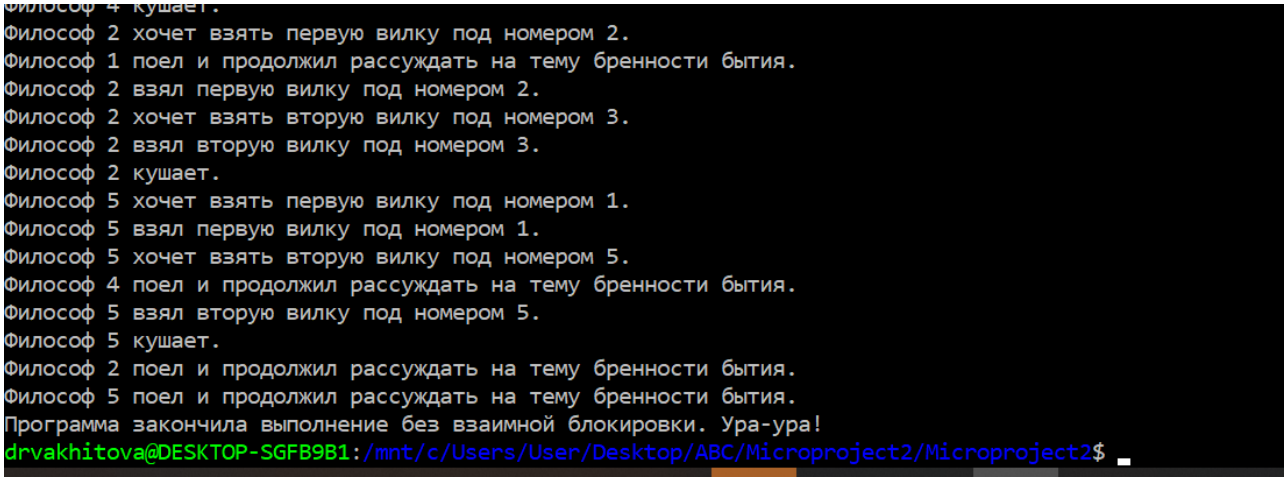

Рисунок 13 – Запуск на 100 приемах пищи.

Код программы:

```
#include <iostream>
#include <thread>
#include <vector>
#include "semaphore.h"

/**
 * Выполнила: студентка БПИ199 Вахитова Диана Рафиковна.
 * ---------------------------------------------------------------------------
 * Условие: Задача об обедающих философах.
 * Пять философов сидят возле
 * круглого стола. Они проводят жизнь, чередуя приемы пищи и размышления.
 * В центре стола находится большое блюдо спагетти. Спагетти длинные и
 * запутанные, философам тяжело управляться с ними, поэтому каждый из них,
 * что бы съесть порцию, должен пользоваться двумя вилками. К несчастью,
 * философам дали только пять вилок. Между каждой парой философов лежит
 * одна вилка, поэтому эти высококультурные и предельно вежливые люди
 * договорились, что каждый будет пользоваться только теми вилками, которые
 * лежат рядом с ним (слева и справа). Написать многопоточную программу,
 * моделирующую поведение философов с помощью семафоров.
 * ---------------------------------------------------------------------------
 * Примечание: Программа должна избегать фатальной ситуации, в которой все философы голодны, но
 * ни один из них не может взять обе вилки (например, каждый из философов
 * держит по одной вилки и не хочет отдавать ее). Решение должно быть
 * симметричным, то есть все потоки-философы должны выполнять один и тот
 * же код.
 */

using namespace std;

// Число философов.
const int philosophers_num = 5;
// Количество раз, котоое поест каждый философ.
int eating_num;
// Время размышлений до приема пищи.
const int secsToThink = 3;
// Время приема пищи.
const int secsToEat = 2;
// Время задержки в миллисекундах после вывода текста
const int afterOutputPauseTimeMilliSec = 250;

// Вектор семафоров, отвечающих за вилки. Работать будут скорее как мутекс - вилка занята/не знаята.
vector<sem_t*> forks = vector<sem_t*>(5);

// Метод, симулирующий проблему.
void eat_and_think(int id) {
    // Выбор вилки с наименьшим и наибольшим номером
    int firstForkInd = id;
    int secondForkInd = (id + 1) % philosophers_num;
    if (firstForkInd > secondForkInd)
        swap(firstForkInd, secondForkInd);
    sem_t* firstFork = forks[firstForkInd];
    sem_t* secondFork = forks[secondForkInd];

    for (int i = 0; i < eating_num; ++i) {
        printf("Философ %d размышляет.\n", id+1);
        this_thread::sleep_for(chrono::seconds(secsToThink));

        // Философ сначала берет вилку с наименьшим номером
        printf("Философ %d хочет взять первую вилку под номером %d.\n", id+1, firstForkInd+1);
        sem_wait(firstFork);
        printf("Философ %d взял первую вилку под номером %d.\n", id+1, firstForkInd+1);
        printf("Философ %d хочет взять вторую вилку под номером %d.\n", id+1, secondForkInd+1);
        sem_wait(secondFork);
        printf("Философ %d взял вторую вилку под номером %d.\n", id+1, secondForkInd+1);
        // И только потом уже берет вильку с наибольшим номером

        printf("Философ %d кушает.\n", id+1);
```

```
        this_thread::sleep_for(chrono::seconds(secsToEat));

        // Кладет сначала вилку с наибольшим номером, затем с наименьшим
        sem_post(secondFork);
        sem_post(firstFork);
        printf("Философ %d поел и продолжил рассуждать на тему бренности бытия.\n", id+1);
    }
}

/**
 * Вызов функций и создание потоков.
 */
int main() {
    // Спрашиваем у пользователя, сколько раз он хочет, чтобы философы поели до окончания работы
программы.
    // Делаем небольшую "защиту от дурака".
    double temp;

    do {
        cout << "Введите, сколько раз вы хотите, чтобы каждый философ поел. (от 1 до 100)\n";
        cin >> temp;
        int b = static_cast<int>(temp);
        if ((double)b != temp) temp = -1;
    } while (temp <= 0 || temp> 100 || static_cast<int>(temp) != temp);
    eating_num = temp;

    for (size_t i = 0; i < philosophers_num; i++)
    {
        // Семафоры, которые по факту мьютексы.
        forks[i] = new sem_t();
        sem_init(forks[i], 1, 1);
    }

    thread* philosophers = new thread[philosophers_num];
    for (int i = 0; i < philosophers_num; ++i) {
        philosophers[i] = thread(eat_and_think, i);
    }

    // Объединение потоков.
    for (int i = 0; i < philosophers_num; ++i) {
        philosophers[i].join();
    }

    // Чистим память.
    delete[] philosophers;

    // Ура.
    cout << "Программа закончила выполнение без взаимной блокировки. Ура-ура!" << endl;
    return 0;
}
```