

## Вариант 8

Задача об обедающих философях. Это классическая задача на взаимодействие параллельных процессов. 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 поест, должен пользоваться двумя вилами. 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 вилами, которые лежат рядом с ним (слева и справа). Написать многопоточную программу, моделирующую поведение философов с помощью семафоров. 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 вилке и не хочет отдавать ее). Время, которое отводится на прием пищи и размышление задается случайно в некотором разумном для наблюдения из вне диапазоне. Решение должно быть симметричным, то есть все потоки-философы должны выполнять один и тот же код (являться равноправными потоками)

## Отчёт программы на 4–5 баллов:

Реализация программы на оценку 4-5 лежит в папке Solution\_1.

- Соблюдены общие требования к отчету.
- В отчете приведен сценарий, описывающий одновременное поведение представленных в условии задания сущностей в терминах предметной области. То есть, описан сценарий, задающий ролевое поведение субъектов и объектов задачи (интерпретация условия с большей степенью детализации происходящего), а не то, как это будет реализовано в программе.

### Сценарий:

Стол за которым сидят пять философов. Каждый философ периодически переходит от состояния размышления к состоянию голода, а

затем — к попыткам поест. Чтобы есть, философу нужны две вилки: правая и левая. Однако вилок всего пять, по одной между каждой парой философов.

Когда философ голоден, он пытается взять сначала одну вилку, затем другую. Если обе доступны — он начинает есть. Пока философ ест, вилки заняты и соседи не могут их взять. Когда философ наелся, он кладет вилки обратно на стол, и они снова становятся свободны для других.

При этом все философы действуют одновременно и равноправно. Введенный блокировщик (ограничение, контролирующее число философов, одновременно пытающихся взять вилки) предотвращает критическую ситуацию, когда все пытаются схватить по одной вилке и застревают.

Так, в каждый момент за столом может наблюдаться разнообразная картина: одни философы думают, другой вдруг становится голодным и ждет вилку, третий уже ест, а еще один только что закончил есть и освободил вилки. Эта динамика непрерывна, пока не истечет время работы программы, моделирующей описанную ситуацию.

- Описана модель параллельных вычислений, используемая при разработке многопоточной программы.

### **Модель:**

В основе реализованной программы лежит модель «Взаимодействующие равные». Каждый философ представлен отдельным потоком, выполняющим один и тот же код (размышление – голод – попытка взять вилки – еда – возвращение к размышлению). При этом отсутствует управляющий поток, а распределение ресурса (вилки) осуществляется динамически с помощью семафоров. Все потоки-философы действуют равноправно и взаимодействуют посредством общих ресурсов, избегая критических ситуаций.

- Описаны входные данные программы, включающие вариативные диапазоны, возможные при многократных запусках.

### **Входные данные:**

Входными данными являются временные параметры, вводимые пользователем при запуске программы с консоли:

Время работы программы (целое число в диапазоне 10–100 секунд)

Минимальное и максимальное время размышления (целые числа в диапазоне 1–10 секунд)

Минимальное и максимальное время приема пищи (целые числа в диапазоне 1–10 секунд)

- Реализовано консольное приложение, решающее поставленную задачу с использованием одного варианта изученных синхропримитивов. В программе используются POSIX-потoki (pthread) для реализации параллельности и семафоры (sem\_t) для обеспечения синхронизации при доступе к разделяемым ресурсам (вилкам) и глобальному контролю количества философов, одновременно пытающихся начать есть. Кроме того, для избежания наложения текстовых сообщений потоков друг на друга при выводе на консоль используется мьютекс, гарантирующий упорядоченный и отдельный вывод информации.
- Ввод данных в приложение реализован с консоли во время выполнения программы (без использования аргументов командной строки).
- Для используемых генераторов случайных чисел описаны их диапазоны и то, как интерпретируются данные этих генераторов.

### Генераторы и диапазоны:

В программе генератор случайных чисел rand(), инициализируемый текущим временем srand((long)time(NULL));, используется для выбора случайного значения в заданных пользователем диапазонах. Например, если минимальное время размышления 1 секунда, а максимальное 10 секунд, то rand() преобразуется в случайное целое число от 1 до 10, которое затем интерпретируется как количество секунд, в течение которых философ будет размышлять или есть.

- Вывод программы информативный, отражает все ключевые протекающие в ней события в терминах предметной области.

- В программе присутствуют комментарии, поясняющие выполняемые действия и описание используемых объектов и переменных.
- Результаты работы программы представлены в отчете.

## Отчёт:

```
Введите время работы программы (10-100 сек): 10
Введите минимальное и максимальное время размышления (1-10 сек): 4 8
Введите минимальное и максимальное время приема пищи (1-10 сек): 5 7
Программа будет работать 10 секунд
Философ 0 думает в течение 6 секунд.
Философ 3 думает в течение 5 секунд.
Философ 2 думает в течение 5 секунд.
Философ 4 думает в течение 5 секунд.
Философ 1 думает в течение 6 секунд.
Философ 3 голоден и ждет разрешения брать вилки.
Философ 3 пытается взять левую вилку 3.
Философ 3 пытается взять правую вилку 4.
Философ 3 ест в течение 6 секунд.
Философ 2 голоден и ждет разрешения брать вилки.
Философ 2 пытается взять левую вилку 2.
Философ 2 пытается взять правую вилку 3.
Философ 2 ест в течение 6 секунд.
Философ 4 голоден и ждет разрешения брать вилки.
Философ 4 пытается взять левую вилку 4.
Философ 4 пытается взять правую вилку 0.
Философ 4 ест в течение 7 секунд.
Философ 0 голоден и ждет разрешения брать вилки.
Философ 0 пытается взять левую вилку 0.
Философ 0 пытается взять правую вилку 1.
Философ 0 ест в течение 6 секунд.
Философ 1 голоден и ждет разрешения брать вилки.
Философ 1 пытается взять левую вилку 1.
Философ 1 пытается взять правую вилку 2.
Философ 1 ест в течение 5 секунд.

Время работы программы истекло. Ожидание завершения потоков...
Философ 3 закончил есть и кладет вилки назад на стол.
```

```
Философ 2 закончил есть и кладет вилки назад на стол.
Философ 0 закончил есть и кладет вилки назад на стол.
Философ 1 закончил есть и кладет вилки назад на стол.
Философ 4 закончил есть и кладет вилки назад на стол.
Программа завершена.
```

```
Process finished with exit code 0
```

## Отчёт программы на 6–7 баллов:

Реализация программы на оценку 6-7 лежит в папке Solution\_2.

- В отчете подробно описан обобщенный алгоритм, используемый при реализации программы исходного словесного сценария. В котором показано, как на программу отображается каждый из субъектов предметной области.

### Обобщенный алгоритм:

#### 1. Инициализация ресурсов:

В программе модель философов представлена следующим образом:

Пять философов — это пять потоков POSIX (pthread).

Пять вилок — это пять семафоров (sem\_t), по одному на каждую вилку.

Ввод параметров с консоли задает временные интервалы размышления и приема пищи, а также время работы программы.

Инициализируется блокировщик — глобальный семафор, ограничивающий число философов, которые одновременно пытаются начать есть.

#### 2. Запуск потоков-философов:

Каждый философ (субъект предметной области) представлен функцией-потоком (философ), которой передаются параметры (указатели на семафоры вилок, блокировщика, и заданные интервалы времени).

Потоки стартуют и начинают выполняться параллельно.

#### 3. Цикл работы каждого философа (каждый поток):

Размышление: Философ (поток) выбирает случайное время в заданных пределах и думает. В коде программы поток просто вызывает sleep().

Переход к еде: После размышления философ голоден и хочет взять вилки. Прежде чем попытаться взять вилки, философ обращается к блокировщику — в программе это означает вызов sem\_wait на глобальном семафоре. Если семафор блокирует — философ ждет.

Захват вилок: После получения разрешения от блокировщика, философ пытается взять две вилки. Если вилка занята, поток ждет освобождения (блокируется на sem\_wait). Таким образом, в коде семафор моделирует доступ к общему ресурсу (вилке).

Прием пищи: Получив обе вилки, философ ест. В коде это снова имитация через sleep() на случайно выбранное время.

Освобождение ресурсов: По окончании еды философ кладет вилки на место (`sem_post` для обоих вилочных семафоров) и сообщает блокировщику о том, что место освободилось (`sem_post` на глобальном семафоре).

#### 4. Завершение работы:

Программа работает в течение заданного времени. По истечении этого времени глобальная переменная сигнализирует потокам о завершении.

Каждый философ, закончив текущий цикл (или находясь в состоянии ожидания), завершает работу.

Главный поток (`main`) дожидается завершения всех потоков (путем `pthread_join`), освобождает ресурсы и завершает программу.

- В программу добавлена генерация случайных данных в допустимых диапазонах.

```
int minThink = 1 + rand() % 10;  
int maxThink = minThink + rand() % (11 - minThink);  
int minEat = 1 + rand() % 10;  
int maxEat = minEat + rand() % (11 - minEat);  
int simulationTime = 10 + rand() % 91;
```

`minThink`, `minEat` от 1 до 10;

`maxThink`, `maxEat` от значений `minThink`, `minEat` до 10;

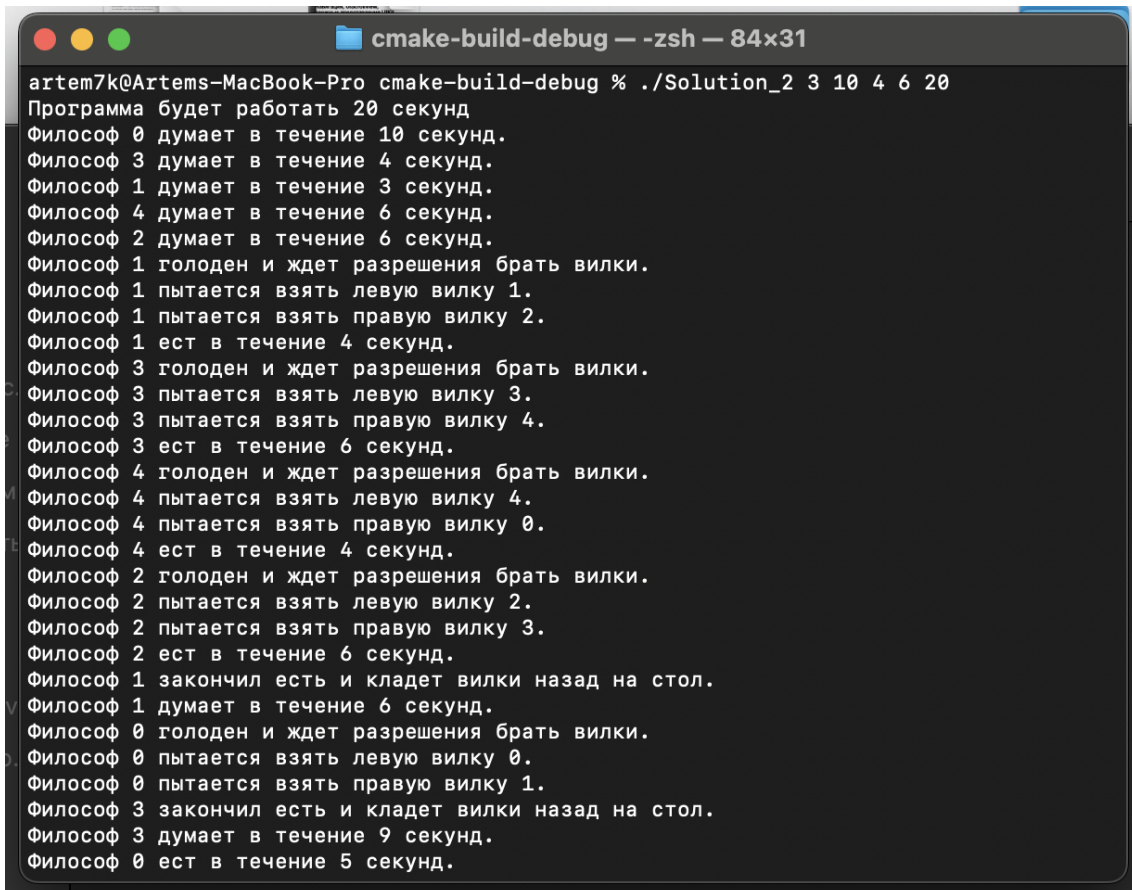
`simulationTime` от 10 до 100.

- Реализован ввод исходных данных из командной строки при запуске программы вместо ввода параметров с консоли во время выполнения программы.

```
int main(int argc, char* argv[]) {
    if (argc < 6) {
        std::cerr << "Использование: " << argv[0] << " minThink maxThink minEat maxEat simulationTime\n";
        return 1;
    }

    int minThink = std::atoi(argv[1]);
    int maxThink = std::atoi(argv[2]);
    int minEat = std::atoi(argv[3]);
    int maxEat = std::atoi(argv[4]);
    int simulationTime = std::atoi(argv[5]);

    // Проверка корректности диапазонов
    if (simulationTime < 10 || simulationTime > 100 ||
        minThink < 1 || minThink > 30 || maxThink < 1 || maxThink > 30 || minThink > maxThink ||
        minEat < 1 || minEat > 30 || maxEat < 1 || maxEat > 30 || minEat > maxEat) {
        std::cerr << "Неправильные значения\n";
        return 1;
    }
}
```



```
artem7k@Artems-MacBook-Pro cmake-build-debug % ./Solution_2 3 10 4 6 20
Программа будет работать 20 секунд.
Философ 0 думает в течение 10 секунд.
Философ 3 думает в течение 4 секунд.
Философ 1 думает в течение 3 секунд.
Философ 4 думает в течение 6 секунд.
Философ 2 думает в течение 6 секунд.
Философ 1 голоден и ждет разрешения брать вилки.
Философ 1 пытается взять левую вилку 1.
Философ 1 пытается взять правую вилку 2.
Философ 1 ест в течение 4 секунд.
Философ 3 голоден и ждет разрешения брать вилки.
Философ 3 пытается взять левую вилку 3.
Философ 3 пытается взять правую вилку 4.
Философ 3 ест в течение 6 секунд.
Философ 4 голоден и ждет разрешения брать вилки.
Философ 4 пытается взять левую вилку 4.
Философ 4 пытается взять правую вилку 0.
Философ 4 ест в течение 4 секунд.
Философ 2 голоден и ждет разрешения брать вилки.
Философ 2 пытается взять левую вилку 2.
Философ 2 пытается взять правую вилку 3.
Философ 2 ест в течение 6 секунд.
Философ 1 закончил есть и кладет вилки назад на стол.
Философ 1 думает в течение 6 секунд.
Философ 0 голоден и ждет разрешения брать вилки.
Философ 0 пытается взять левую вилку 0.
Философ 0 пытается взять правую вилку 1.
Философ 3 закончил есть и кладет вилки назад на стол.
Философ 3 думает в течение 9 секунд.
Философ 0 ест в течение 5 секунд.
```

Команда: `./Solution_2 3 10 4 6 20`

Как видим, всё работает!

## Отчёт программы на 8 баллов:

Реализация программы на оценку 8 лежит в папке Solution\_3.

- В программу, наряду с выводом в консоль, добавлен вывод результатов в файл. Имя файла для вывода данных задается в командной строке как один из ее параметров.

```
./Solution_3 -c minThink maxThink minEat maxEat simulationTime  
output_file
```

- Результаты работы программы выводятся на экран и записываются в файл.
- Наряду с вводом исходных данных через командную строку добавлен альтернативный вариант их ввода из файла, который по сути играет роль конфигурационного файла. Имя этого файла задается в командной строке вместо параметров, которые в этом случае не вводятся. Управление вводом в командной строке осуществляется с использованием соответствующих ключей.

```
./Solution_3 -f config_file output_file
```

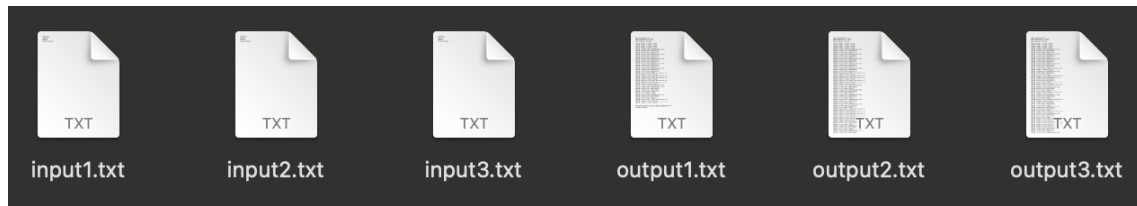
```
artem7k@Artems-MacBook-Pro cmake-build-debug % ./Solution_3  
Использование:  
1. С параметрами командной строки:  
./Solution_3 -c minThink maxThink minEat maxEat simulationTime output_file  
2. С конфигурационным файлом:  
./Solution_3 -f config_file output_file  
artem7k@Artems-MacBook-Pro cmake-build-debug %
```

```
artem7k@Artems-MacBook-Pro cmake-build-debug % ./Solution_3 -f input3.txt output3.txt  
Начальная конфигурация:  
Время размышления: 1-10 секунд  
Время приема пищи: 4-8 секунд  
Время симуляции: 60 секунд
```

Как видим, всё работает! Также программа корректно обрабатывает неправильный формат конфигурации файлов, неправильное название файла и тд.



- Приведено три варианта входных и выходных файлов с различными исходным данными и результатами выполнения программы.



- Ввод данных из командной строки расширен с учетом введенных изменений.

## Отчёт программы на 9 баллов:

Реализация программы на оценку 9 лежит в папке Solution\_4.

- Разработано альтернативное решение. Вместо семафоров используются условные переменные, `std::mutex` для синхронизации. Добавлен класс `ForkObserver` (наблюдатель), который контролирует состояние вилки, управляет доступом философов к вилкам, использует условные переменные для уведомления философов, предотвращает взаимные блокировки.

Входные данные такие же как и в прошлой программе. Выходные данные представлены путём прогона через новую реализацию.



- Для сравнения функционала, предоставлены выходные данные для программы на оценку 8 и оценки 9. Они обе выполняют одну и ту же поставленную задачу. Однако не очень понятно, что значит "идентичности поведения при одинаковых данных" так как идентичных результатов не может быть, из-за использования генератора случайных чисел `srand((unsigned int)time(nullptr));`. При запуске программы на одних и тех же диапазонах, и так не будет "идентичных" выходных данных, так как каждый раз они генерируются уникальными.

## Отчёт программы на 10 баллов:

Реализация программы на оценку 10 лежит в папке Solution\_5.

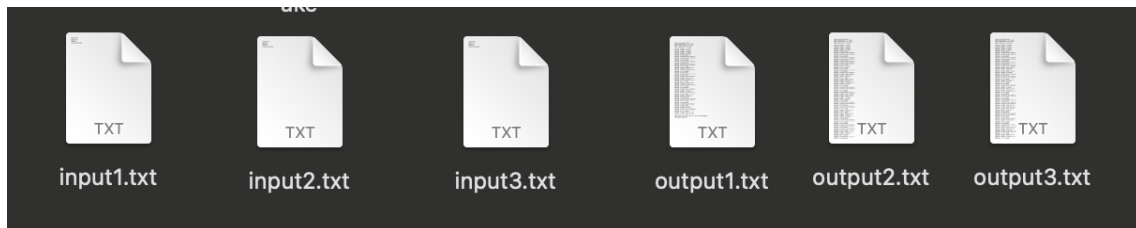
- Разработана еще одна программа с использованием OpenMP.

Используется модель OpenMP, а для синхронизации доступа к ресурсам применяются OpenMP-замки (omp\_lock\_t). Таким образом, каждый философ – это поток OpenMP, запускаемый внутри параллельной области.

```
[artem7k@Artems-MacBook-Pro cmake-build-debug % ./Solution_5 -f input1.txt output1.txt]
[
Начальная конфигурация:
Время размышления: 3-10 секунд
Время приема пищи: 4-6 секунд
Время симуляции: 20 секунд

Философ 0 думает 9 секунд.
Философ 4 думает 4 секунд.
Философ 3 думает 9 секунд.
Философ 2 думает 4 секунд.
Философ 1 думает 3 секунд.
Философ 1 проголодался.
Философ 1 пытается взять вилку 1.
Философ 1 пытается взять вилку 2.
```

Как мы видим, всё работает!



Конфигурации и результаты программы можно найти в папке Solution\_5/cmake-build-debug.

**СПАСИБО ЗА ВНИМАНИЕ!**

