**«Проблема обедающих философов»** — классический пример, используемый в информатике для иллюстрации проблем синхронизации при разработке параллельных алгоритмов и техник решения этих проблем.

Проблема была сформулирована в 1965 году Эдсгером Дейкстрой как экзаменационное упражнение для студентов. В качестве примера был взят конкурирующий доступ к ленточному накопителю. Вскоре проблема была сформулирована Ричардом Хоаром в том виде, в каком она известна сегодня.

**Постановка задачи:**

Пять безмолвных философов сидят вокруг круглого стола, перед каждым философом стоит тарелка спагетти. Вилки лежат на столе между каждой парой ближайших философов.

Каждый философ может либо есть, либо размышлять. Приём пищи не ограничен количеством оставшихся спагетти — подразумевается бесконечный запас. Видимо, эти спагетти настолько скользкие, что философ может их есть только двумя вилками, причем взять он их может только справа и слева, т.е. нельзя тянуться через весь стол (альтернативная формулировка проблемы подразумевает миски с рисом и палочки для еды вместо тарелок со спагетти и вилок).

Жизнь каждого их этих пяти философов состоит из чередования размышлений и приемов пищи. Каждый философ может взять ближайшую вилку (если она доступна), или положить — если он уже держит её. Взятие каждой вилки и возвращение её на стол являются раздельными действиями, которые должны выполняться одно за другим.

Суть проблемы заключается в том, чтобы разработать модель поведения (параллельный алгоритм), при котором ни один из философов не будет голодать, то есть будет вечно чередовать приём пищи и размышления.

**Самое просто и очевидно решение такое:**

void philosopher(int i) /\* i: номер философа (от 0 до 4) \*/

{

while (TRUE) {

think( ); /\* философ размышляет \*/

take\_fork(i); /\* берет левую вилку \*/

take\_fork((i+1) % N); /\* берет правую вилку; \*/ /\* ( % - оператор деления по модулю )\*/

eat(); /\* ест спагетти \*/

put\_fork(i); /\* кладет на стол левую вилку \*/

put\_fork((i+1) % N); /\* кладет на стол правую вилку \*/

}

}

К сожалению, этот алгоритм не является хорошим алгоритмом, т.к. может возникнуть ситуация, когда все философы берут левую вилку. Получается, что дальше они уже не могут взять правую вилку, а значит находятся в бесконечном её ожидании. Это состояние можно назвать состоянием взаимной блокировки.

Можно модифицировать задачу следующим образом: после того, как философ берет левую вилку, программа проверяет, доступна ли правая вилка. Если нет, философ должен положить левую вилку на место, а через некоторое время повторить попытку. Это исключит возможность взаимной блокировки, но по-прежнему остается возможность зацикливания. Например, при некоторой доле невезения, все философы одновременно берут левую вилку. Тогда через t1 минут они положат её, подождут ещё t2 минут, опять возьмут, и т.д. бесконечно. Т.е. наши философы опять будут голодать.

Взаимное исключение является основной идеей «Проблемы обедающих философов». Эта проблема представляет собой общий, абстрактный сценарий, позволяющий объяснить проблемы этого типа. Ошибки философов наглядно демонстрируют те трудности, которые возникают в реальном программировании, когда нескольким программам требуется исключительный доступ к совместно используемым ресурсам. Эти вопросы изучаются в области параллельных вычислений.

**Решение задачи**

**Официант**

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 левую вилку, а потом — правую (или наоборот), что упрощает логику. Официант работает, как семафор — понятие, введённое Дейкстрой в 1965 году.[[5]](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D0%B0_%D0%BE%D0%B1%D0%B5%D0%B4%D0%B0%D1%8E%D1%89%D0%B8%D1%85_%D1%84%D0%B8%D0%BB%D0%BE%D1%81%D0%BE%D1%84%D0%BE%D0%B2#cite_note-EWD-123-5)

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

### Иерархия ресурсов

Другое простое решение достигается путём присвоения частичного порядка ресурсам (в данном случае вилкам) и установления соглашения, что ресурсы запрашиваются в указанном порядке, а возвращаются в обратном порядке. Кроме того, не должно быть двух ресурсов, не связанных порядком, используемых одной рабочей единицей.

Пусть ресурсы (вилки) будут пронумерованы от 1 до 5, и каждая рабочая единица (философ) всегда берёт сначала вилку с наименьшим номером, а потом вилку с наибольшим номером из двух доступных. Далее, философ кладёт сначала вилку с бо́льшим номером, потом — с меньшим. В этом случае, если четыре из пяти философов одновременно возьмут вилку с наименьшим номером, на столе останется вилка с наибольшим возможным номером. Таким образом, пятый философ не сможет взять ни одной вилки. Более того, только один философ будет иметь доступ к вилке с наибольшим номером, так что он сможет есть двумя вилками. Когда он закончит использовать вилки, он в первую очередь положит на стол вилку с бо́льшим номером, потом — с меньшим, тем самым позволив другому философу взять недостающую вилку и приступить к еде.

Данное решение было предложено Дейкстрой.

В то время, как иерархия ресурсов позволяет избежать взаимных блокировок, данное решение не всегда является практичным, в особенности когда список необходимых ресурсов неизвестен заранее. Например, если рабочая единица удерживает ресурс 3 и 5 и решает, что ей необходим ресурс 2, то она должна выпустить ресурс 5, затем 3, после этого завладеть ресурсом 2 и снова взять ресурс 3 и 5. Компьютерные программы, которые работают с большим количеством записей в базе данных, не смогут работать эффективно, если им потребуется выпускать все записи с верхними индексами прежде, чем завладеть новой записью. Это делает данный метод непрактичным.

**Решение на основе монитора**

Пример ниже показывает решение, где вилки не представляются явно. Философы могут есть, если ни один из их соседей не ест. Аналогично системе, где философы, которые не могут взять вторую вилку, должны положить первую вилку до того, как они попробуют снова.

В отсутствии блокировок, связанных с вилками, философы должны обеспечивать то, что начало принятия пищи не основывается на старой информации о состоянии соседей. Например: Если философ B видит, что A не ест в данный момент времени, а потом поворачивается и смотрит на C, A мог начать есть, пока философ B смотрит на C. Используя одну взаимоисключающую блокировку, можно избежать этой проблемы. Эта блокировка не связана с вилками, но она связана с решением процедур, которые могут изменить состояние философов. Это обеспечивается монитором.

Алгоритм монитора реализует схему «проверить, взять и положить» и совместно использует взаимоисключающую блокировку. Заметьте, что философы, желающие есть, не будут иметь вилок.

Если монитор разрешает философу, желающему есть, действовать, то философ снова завладевает первой вилкой, прежде чем взять уже свободную вторую.

По окончании текущего приёма пищи философ оповещает монитора о том, что обе вилки свободны.

Стоит заметить, что этот алгоритм монитора не решает проблемы голодания. Например, философ B может бесконечно ждать своей очереди, если у философов A и C периоды приёма пищи всё время пересекаются. Чтобы гарантировать также, что ни один философ не будет голодать, можно отслеживать, сколько раз голодный философ не ел, когда его соседи положили вилки на стол. Если количество раз превысит некий предел, такой философ перейдёт в состояние Голодания и алгоритм монитора форсирует процедуру завладения вилками, выполняя условие недопущения голодания ни одного из соседей.

Философ, не имеющий возможности взять вилки из-за того, что его сосед голодает, находится в режиме полезного ожидания окончания приёма пищи соседом его соседа. Эта дополнительная зависимость снижает параллелизм. Увеличение значения порога перехода в состояние Голодание уменьшает этот эффект.