8 Многозадачность и параллельное программирование

8.1 Напоминания

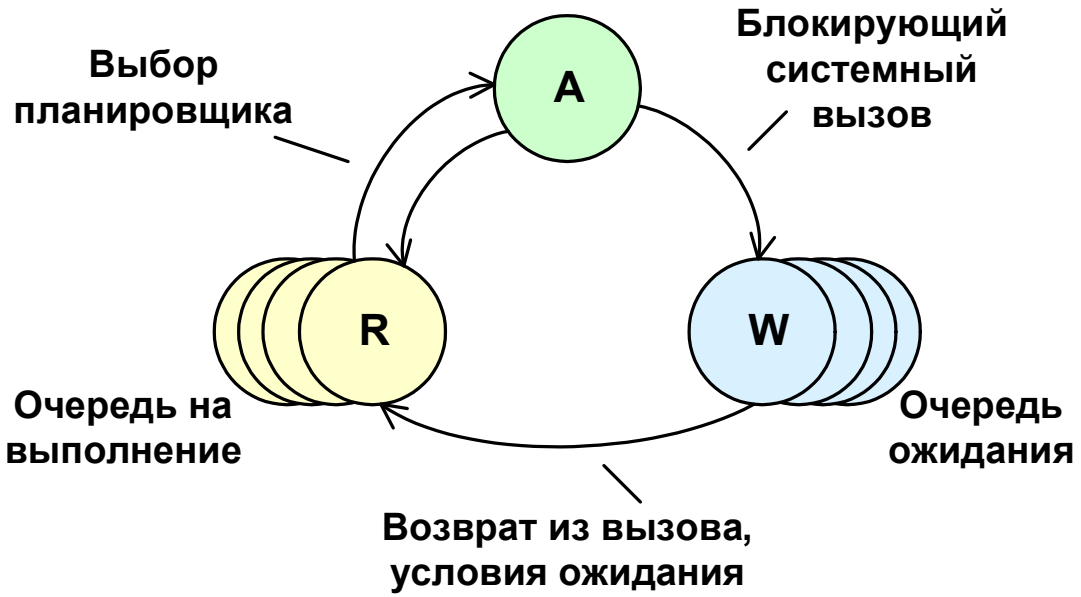
Многозадачность – способность системы выполнять параллельно (псевдопараллельно) несколько задач (программ), отличая их друг от друга (требуется идентификация задач) и влияя на порядок выполнения (имеются средства управления задачами)

Вычислительный ***процесс*** (упрощенно) – работающая программа. Как правило, в системе определен тип объекта «процесс».

Вычислительный ***поток*** (thread) – последовательность инструкций, выполняемая независимо и асинхронно по отношению к другим аналогичным последовательностей. Отражение в иерархии объектов – объект «поток» (thread).

Теория параллельных алгоритмов и параллельного программирования обычно оперирует понятием «процесс», однако в современных системах планирование выполнения программ происходит обычно на уровне потоков)

Основные состояния потоков: ***Активность*** (Active, ***A***), ***Готовность*** (Ready, ***R***), ***Ожидание*** (Wait, ***W***).



Основные состояния потока (процесса)

Вычислительные процессы (задачи) по наличию и активности взаимодействия между ними:

– не связанные – выполняются независимо, возможны лишь неявные коллизии при использовании общесистемных ресурсов; удобны для распараллеливания

– слабо связанные – к общим ресурсам обращаются сравнительно редко; достаточно удобны для распараллеливания, в т.ч. на многомашинных конфигурациях при условии наличия быстрых интерфейсов обмена данными

– сильно связанные – интенсивно обращаются к общим ресурсам, что приводит к частым коллизиям; распараллеливание возможно на многопроцессорных (многоядерных) конфигурациях, но проблематично

– не поддающиеся распараллеливанию

8.2 Задачи и проблемы взаимодействия

Наличие в системе многозадачности, параллелизма – неизбежно выход на другой уровень проблем, которые в обычном однозадачном программировании не встречаются.

Проблема ***взаимного исключения*** – исключение одновременного совершения несколькими потоками действий, которые по определению не должны выполняться одновременно. (Пример –вывод документа на печать: при смешивании данных двух документов получаем не третий, объединенный, а некорректный «испорченный» результат.)

Проблема ***синхронизации*** – установление необходимого правильного порядка выполнения различных «шагов» взаимодействующих потоков (процессов), передача управления между ними.

Проблема ***обмена данными*** вытекает из концепции изолированных адресных пространств – процессы не имеют возможности напрямую обращаться «внутрь» друг друга. Потоки одного процесса действуют в едином адресном пространстве, но в общем случае во взаимодействии могут участвовать потоки разных процессов.

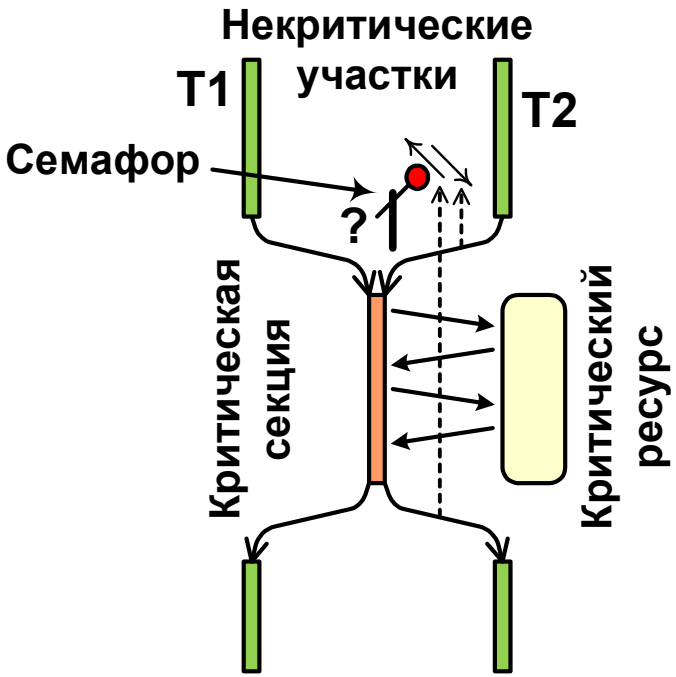
Решением всех задач становится использование специальных системных объектов – объектов ***IPC*** (***I***nter-***P***rocess ***C***ommunication) и, как частный случай, ***ISO*** (***I***nter-process ***S***ynchronization ***O***bjects)

8.3 Критический ресурс и критическая секция

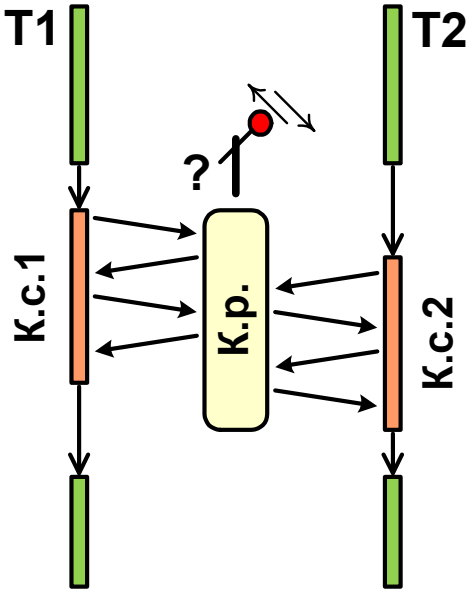
Базовые понятия в параллельном программировании.

Критический ресурс – такой ресурс системы, который не может использоваться одновременно более чем заданным числом пользователей. Часто речь идет о доступности его не более чем одному пользователю.

Критическая секция – та часть алгоритма, где происходят обращения к критическим ресурсам. Очевидно, число потоков, находящихся в критической секции, ограничивается в соответствии с характеристиками критического ресурса (часто – одним потоком).



Потоки с общей критической секцией



Потоки с отдельными критическими секциями

Критическая секция (с вариантами)

Требования к критическим секциям:

– В критической секции не должны находиться одновременно более одного потока (или иного заранее заданного их числа)

– Решение о входе в критическую секцию (выбор одного из конкурирующих потоков) должно приниматься за конечное время; недопустимо бесконечное ожидание перед входом

– Время пребывания потока внутри секции должно быть конечно, а его состояние должно быть только активным

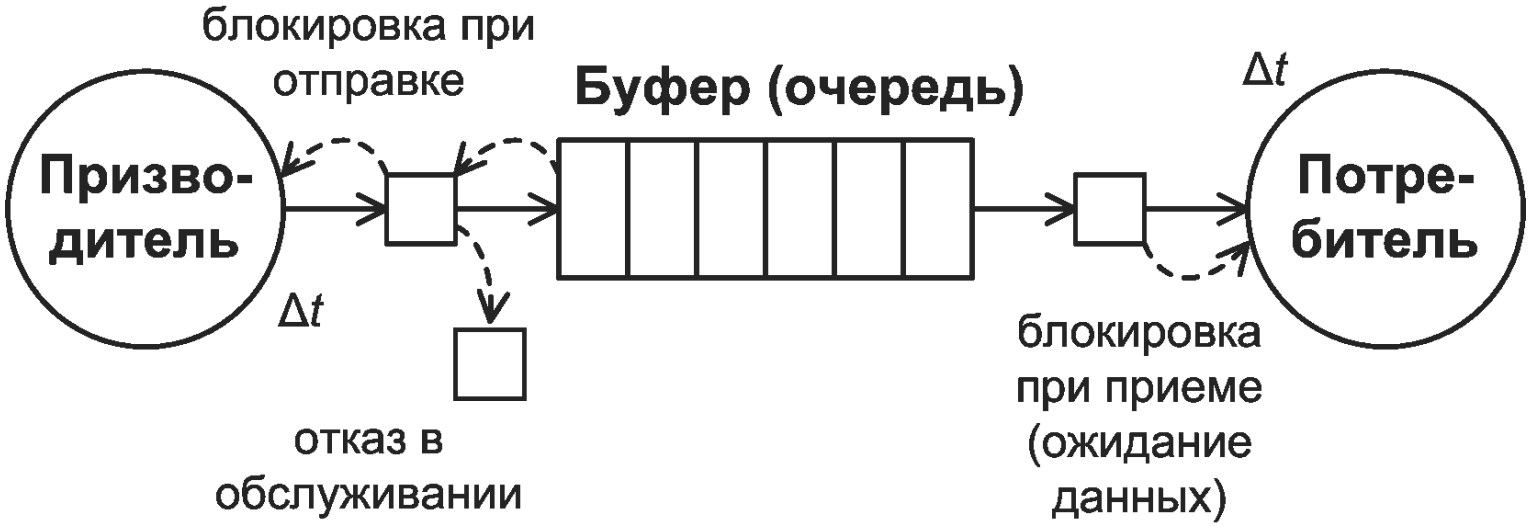
– Внутри критической секции не действуют приоритеты

– Потоки вне критической секции не могут блокировать другие потоки, в т.ч. и фактом своего останова

– Нельзя рассчитывать на какие-либо конкретные количественные параметры системы, например на точно определенное быстродействие и количество процессоров.

8.4 Модели ситуаций взаимодействия

8.4.1 Модель «Производитель – потребитель»



Модель «Производитель – потребитель»

Пример: «Производитель» – получение кадров видеопотока из сети (или программная генерация кадров), «Потребитель» – отображение кадров.

Проблемы:

– согласование скоростей производства и потребления, в т.ч. обеспечение оптимальных условий для одной из сторон;

– предотвращение «гонок» (столкновений) при доступе к буферу;

– взаимное управление и настройка

Буферизация – сглаживание пиковых значений времени обработки. Не может решить проблему постоянной разницы в производительности (один из участников после исчерпания или опустошения буфера будет заблокирован).

Критическая секция – прием «ресурсов» в очередь и изъятие их из очереди (проверка свободного места, модификация указателей и счетчиков, и т.д.). Минимизация времени блокировки.

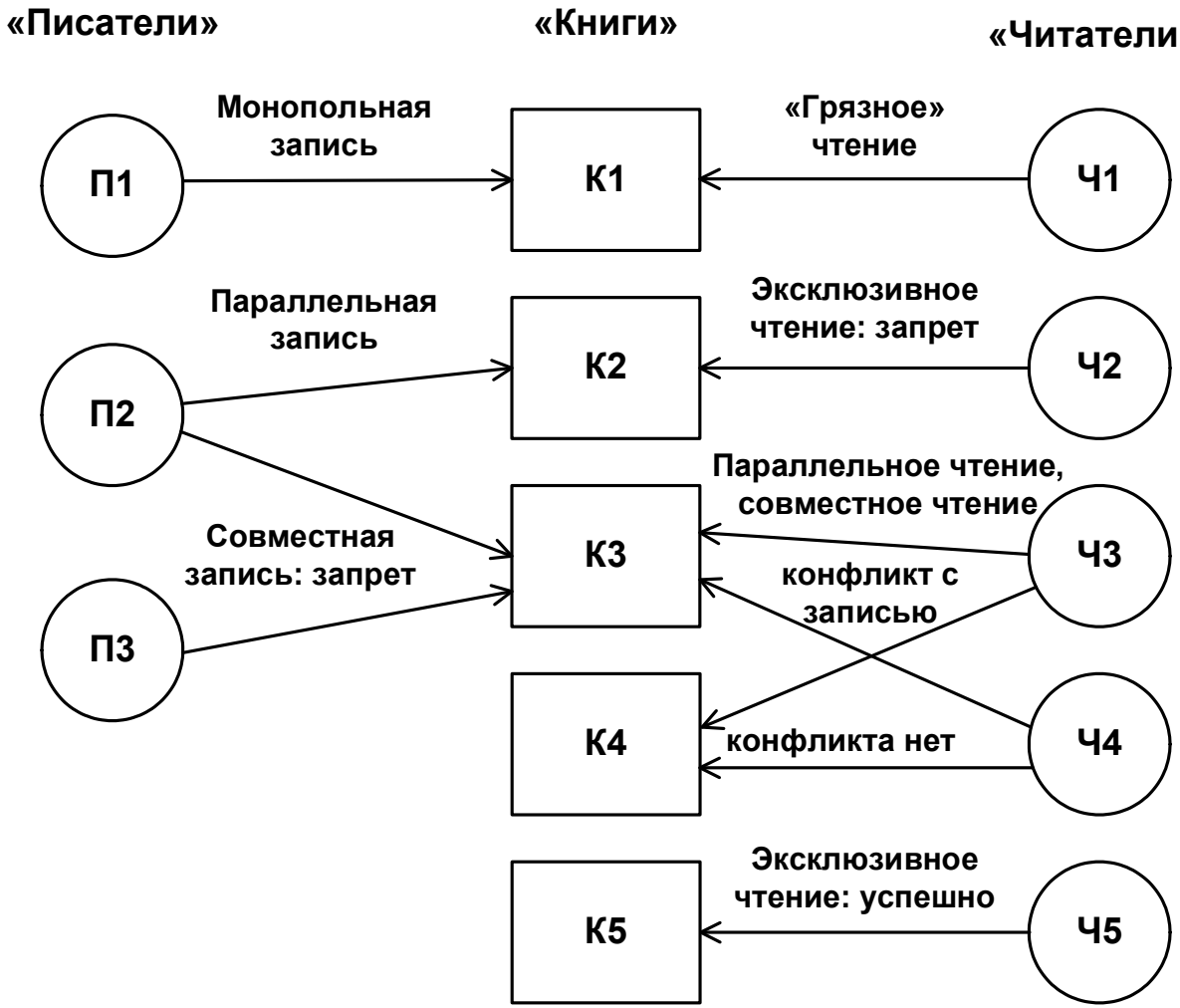
Способы разрешения коллизий:

– Детерминированная передача права на доступ к буферу производителю и потребителю попеременно

– Встречный поток пустых сообщений-«квитанций»

– Семафоры либо мьютексы для защиты критической секции

8.4.2 Модель «Писатели и читатели»



Модель «Писатели и читатели»

Примеры: доступ к БД; совместно используемые файлы конфигурации; файлы проекта в совместном доступе.

Сочетания видов доступа:
– монопольная (эксклюзивная) запись;
– совместная запись – как правило, запрещена;
– монопольное (эксклюзивное) чтение;
– «грязное» чтение (dirty read) – параллельно с записью;
– совместное чтение (но не одновременно с записью!) – как правило, разрешено;
– любое чтение не подготовленного ресурса («книга не написана») – как правило, запрещено.

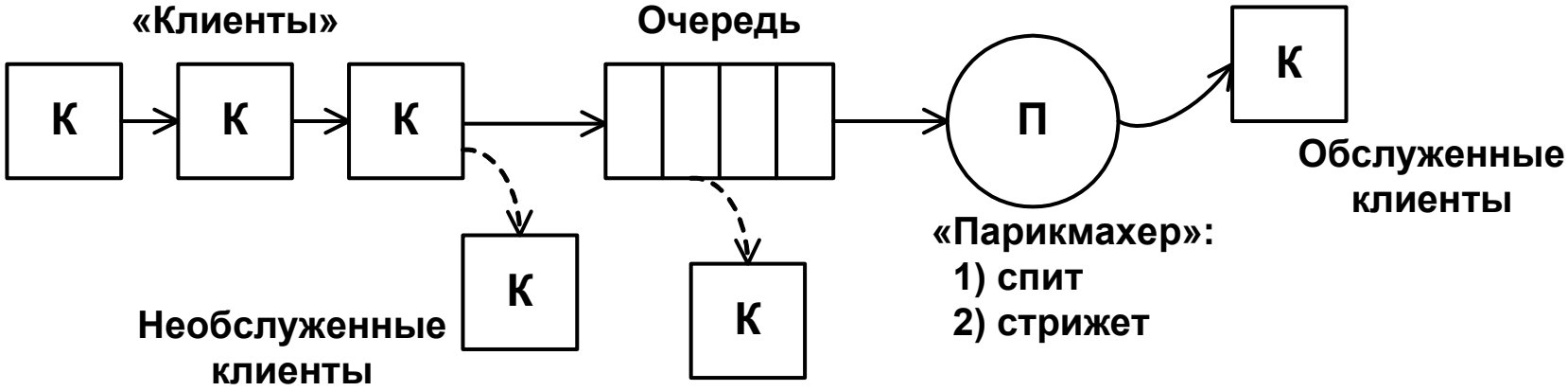
«Грязное» чтение – повышение производительности (снижение или устранение вероятности блокировки), но допущение нарушения целостности считываемых данных.

Регулирование доступа – семафоры или мьютексы, отражающие состояние каждого ресурса («книги»). Критическая секция – захват ресурса для доступа соответствующего вида. Разный уровень захвата критической секции («читателем» или «писателем»).

Типичная проблема: перехват «читателями» права доступа ко всем ресурсам (число «читателей» обычно больше, и интенсивность чтения ресурсов выше).

Решение: двойные семафоры, учет заявок на доступ, приоритет заявок. Наличие заявки на запись не выталкивает актуальных читателей, но блокирует появление новых. Заявка на запись имеет преимущество перед заявкой на чтение, но не перед самим процессом чтения.

8.4.3 Модель «Парикмахерская»



Модель «Парикмахерская»

Упрощенная система массового обслуживания с одной ступенью обработки.

Заявки – «клиенты»

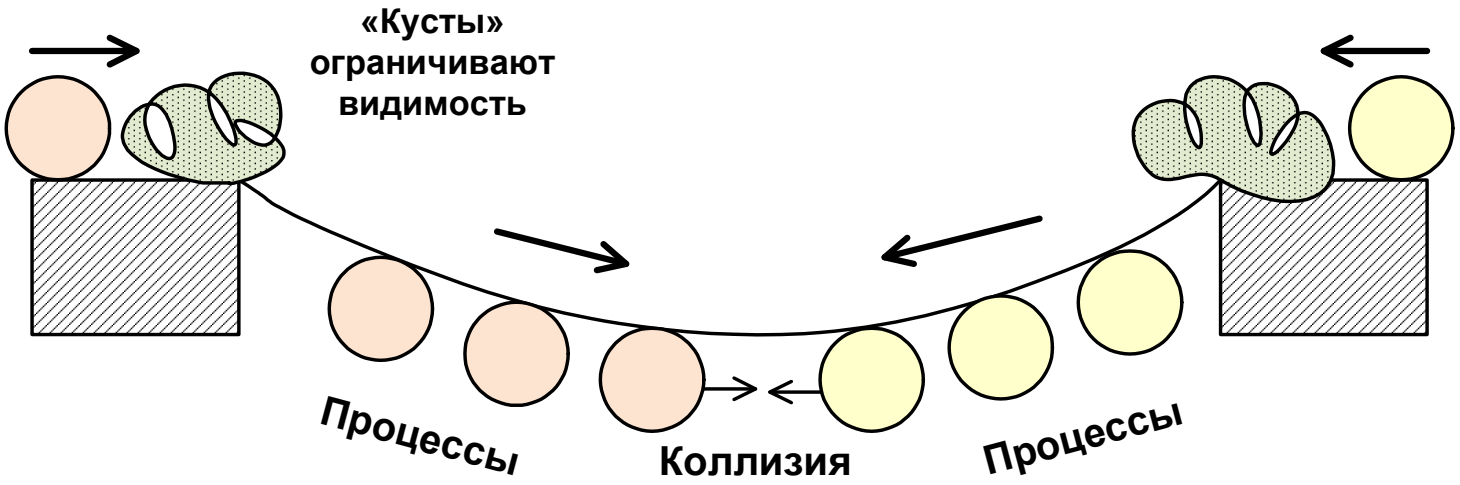
Обрабатывающий процесс – «парикмахер»

Очередь клиентов – с ограничением числа мест и времени ожидания в ней.

Состояния процесса:
– «сон» (в отсутствие клиентов)
– «работа» (при наличии клиента)

Основная задача – оптимизация загрузки системы: обрабатывающий процесс практически не бывает «спящим», но очередь практически не заполнена, потерь клиентов нет.

8.4.4 Модель – «Бабуины на канате»



Модель «Бабуины на канате»

Процессы – «бабуины»

Две категории процессов – движутся влево или вправо. Смена направления невозможна.

Дополнительные обстоятельства:
– грузоподъемность «канатного моста» ограничена
– видимость происходящего вблизи концов «моста» ограничена

Разрешение коллизий – путем «сброса» процессов с «каната»

Цель моделирования – организовать движение процессов с максимальной пропускной способностью и по возможности без потерь.

Пример применения – конвейерная обработка, взаимодействие в распределенных системах, коммуникационные протоколы.

8.4.5 Модель – «Обедающие философы»

Модель предложена Э. Дейкстрой в 1965 г., неоднократно модифицирована и адаптирована (например, Р. Хоаром, 1985 г.). Хорошо описывает функционирование многозадачной системы с разделяемыми ресурсами.

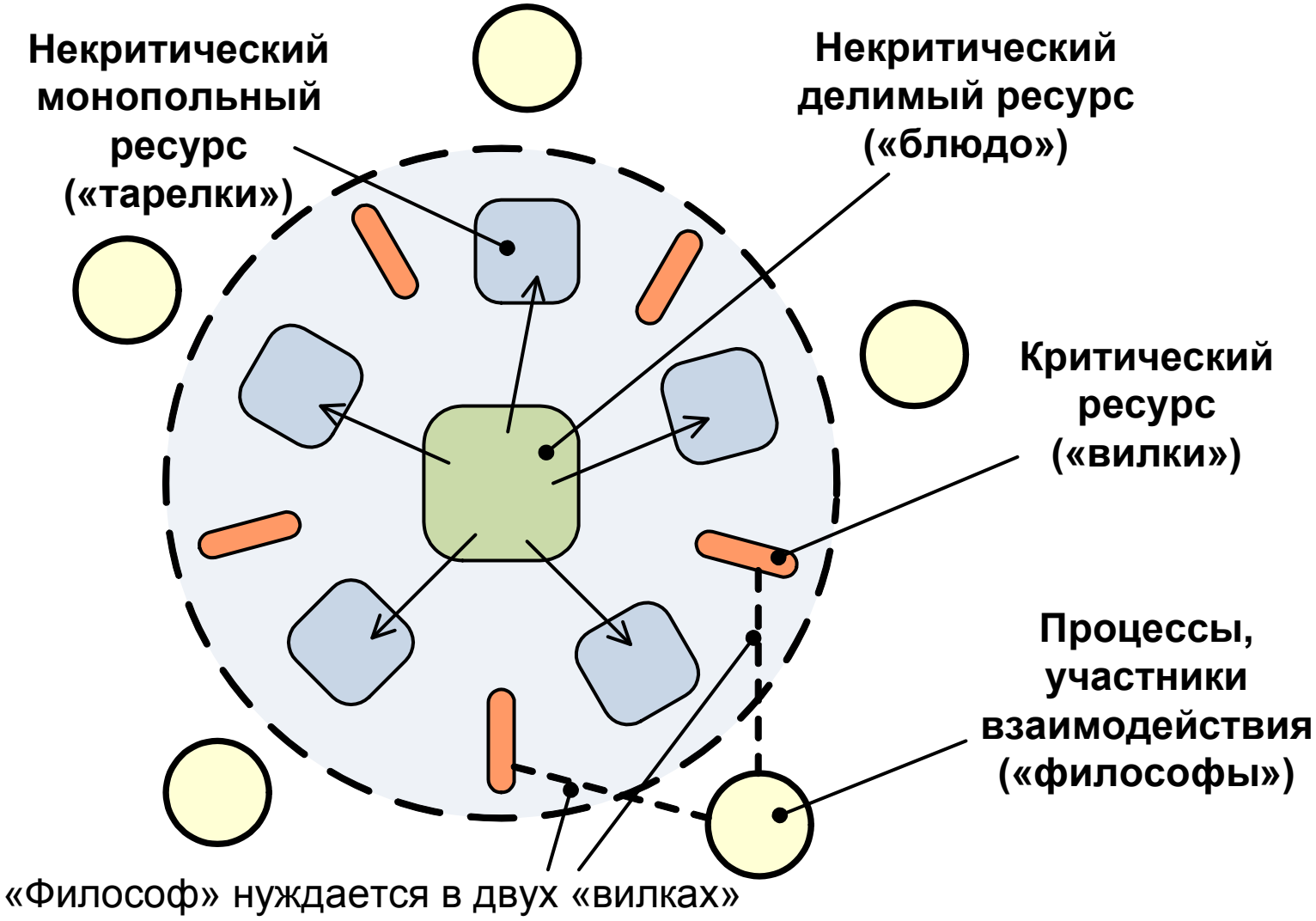
Элементы (участники) модели:

– общее «***блюдо***» – некритический разделяемый ресурс (предполагается бесконечным), существенного влияния не оказывает (можно моделировать задержки предоставления доступа)

– персональные «***тарелки***» – некритический монопольный ресурс, влияния не оказывают

– «***вилки***» – критический ресурс; для еды требуются две вилки одновременно

– «***философы***» – процессы, конкурирующие за ресурсы



Модель «Обедающие философы»

Общее число «вилок» равно числу «философов», причем каждому потенциально доступны только две ближайшие «вилки». Брать «вилки» через стол или отбирать их у соседа нельзя. Прямого общения между «философами» нет (изоляция процессов в многозадачной системе)

Состояния «философов»:

– «***еда***» (использование ресурса)

– «***размышления***» (ожидание, но не из-за невозможности получить «еду»)

– «***голодание***» (ожидание получения «еды»)

Критические секции – периоды пользования критическими ресурсами («вилками»).

Алгоритм действия «философа» (повторяется в цикле):

– «размышления» сытого «философа» (без вилок)

– последовательный захват двух ближайших «вилок» (с ожиданием, если они заняты)

– «еда» (возможно, с паузами на «размышления» еще голодного философа)

– освобождение «вилок»

Первоначальная постановка задачи – программная реализация модели на пять «философов», функционирующая без взаимной блокировки.
Более широкая постановка задачи: обеспечить безаварийное (без тупиков) и по возможности оптимальное (эффективное) функционирование коллектива «философов» и справедливый доступ их к «еде».

Возникающие проблемы:

Тупик (взаимная блокировка, ***deadlock***) – каждый из процессов владеет частью необходимых ресурсов и не может выполняться, поскольку другая часть необходимых ресурсов занята соседями. Тупик является естественным следствием попытки входа более чем в одну критическую секцию одновременно. Находясь в тупике, процессы не могут самостоятельно разрешить эту ситуацию.

Зависание (***livelock***) – бесконечное повторение всеми «философами» попыток завладеть ресурсами.

Вечное голодание (***starvation***) – невозможность для процесса получить доступ к «еде». Может быть следствием дисбаланса в интенсивности «еды» и «размышлений» между соседями, либо результатом «***заговора соседей***»

Возможные решения:

Естественный («наивный») подход: ***прямое моделирование*** элементов системы: «вилкам» соответствуют семафоры или мьютексы, отражающие свободность/занятость. Каждый «философ» пытается «присваивать» соответствующие мьютексы и блокируется, если те оказываются уже заняты. Проблемы не решаются, но будут добросовестно продемонстрированы.

«Классическое» эффективное решение: семафоры связан не с «вилками», а с «философами», и передают им сигналы о наличие свободной «вилки». «Философ», захватывая свой семафор, пытается понизить его сразу на 2 единицы – это возможно только тогда, когда семафор был открыт со стороны обоих соседей. Закончив «еду», философ разблокирует семафоры соседей, но лишь на 1 единицу каждый. Таким образом, каждый «философ» получает право выполняться от соседей и передает его соседям. Фактически это один из способов реализовать «залповое» выделение ресурса, избегая вхождения более чем в одну критическую секцию.
Есть проблема корректного старта.

Общая ***блокировка*** всех остальных «философов» на время «еды» одного из них. Достигается единственным семафором или мьютексом.
Недостаток – значительное снижение производительности.

Усовершенствование: общая блокировка только новых попыток захвата «вилок», т.е. от момента взятия одним из «философов» первой вилки и до получения им второй остальные «философы» брать вилки не могут, но могут их освобождать или продолжать использовать. Требуется один дополнительный семафор (общий для всех «вилок»).
Сравнительно простое решение со сравнительно небольшой потерей производительности.

Таймауты ожидания процессами получения ресурсов.
«Философ» должен освободить уже захваченную «вилку», если не может получить вторую в течение ограниченного времени.
Проблема – необходимо обеспечивать строгое соблюдение правил всеми «философами».

Иерархия ресурсов и упорядочение их получения.
Дополнительные ограничения на алгоритм поведения «философов».
Решение относится к методам избегания тупиков при распределении ресурсов.

Случайный порядок захвата ресурсов и случайная задержка перед повторной попыткой после неудачи (в сочетании с тайм-аутами). Эффективность не гарантируется, но вероятность длительного зависания снижается до приемлемой. Широко используется в реальных протоколах взаимодействия, в т.ч. в сетях.
Как и предыдущие решения, требует обеспечить обязательное соблюдение единых правил всеми «философами».

Детерминированные (***маркерные***) методы: передача разрешений воспользоваться «вилками» по кругу, предоставление «философам» отдельных временных «слотов» и т.п.
Характерны гарантированное предоставление доступа и ограниченная, зато предсказуемая производительность.

Строго одновременный «***атомарный***» захват только обеих «вилок» и ожидание «философом» этого момента; такой подход к выделению ресурсов называют также ***залповым***.
Результат – уход от наличия у каждого процесса двух критических секций (и, соответственно, от первоначальной постановки задачи).

Усложнение процедур захвата и освобождения с дополнительными промежуточными состояниями (например, «грязные» использованные «вилки»).
Так или иначе, при усложнении логики желательно отделить алгоритм решения от «философов» (прикладных процессов).

Наличие «***официанта***» (реализованного в виде ***семафора***) как способ обеспечения атомарности – недопущение захвата «вилки», если второй свободной в этот момент нет. Захват «вилки» требует разрешения от «официанта», который, зная состояние всех «вилок», при необходимости блокирует «философа» еще до обращения его к критическому ресурсу.

Наличие ***монитора*** – функции (набора функций) в качестве процедурного интерфейса (API) распределения ресурсов.
«Монитор вилок» контролирует состояние «вилок» и определяет возможность/невозможность снабжения ими конкретного «философа», который и блокируется при обращении к монитору, если запрос не может быть удовлетворен. Функции монитора гарантированно атомарны.
Реализация собственно «философов» сводится к вызовам функций монитора.
Отличие о предыдущего решения – семафор не предоставляется «философу» непосредственно, а скрыт внутри монитора.

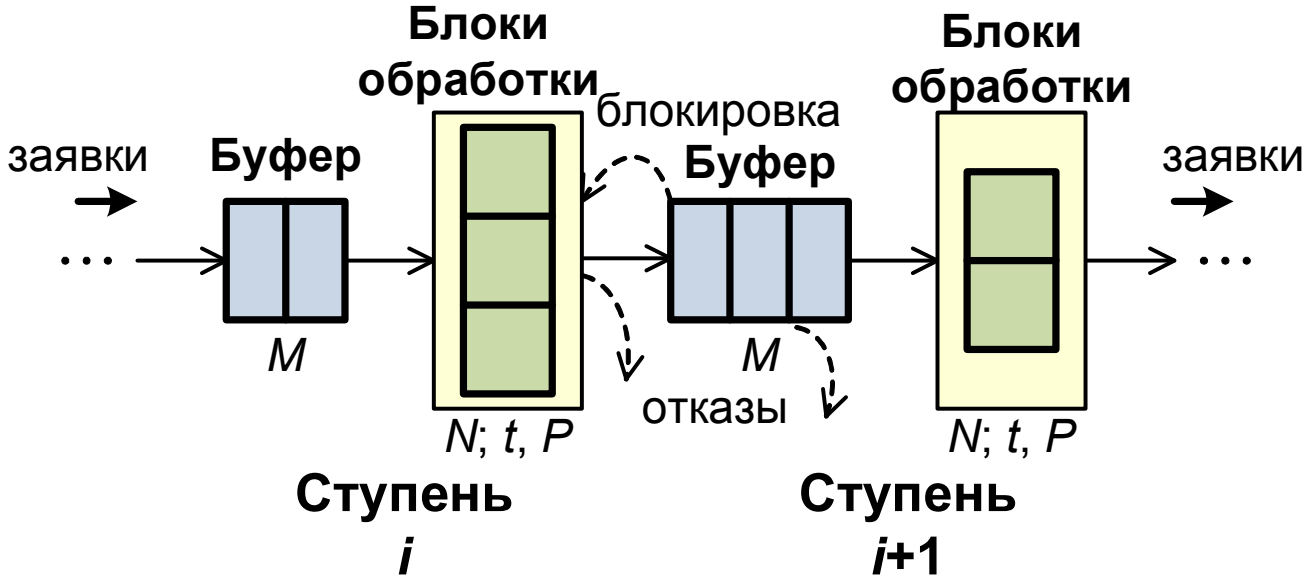
Использование дополнительного процесса – «***слуги***», который должен сопроводить «философа» к столу, снабдить его двумя «вилками» и (это важно!) проследить, чтобы тот начал «есть», и при этом не отвлекался бы на «размышления» с «вилками» в руках. Также «слуга» обеспечивает выполнение прочих требований регламента («правил поведения»).
От предыдущих отличается расширенными функциями и полномочиями «слуги». Аналогия в реальных системах – сервисы-планировщики.

Избыточность ресурсов: добавление одной дополнительной «вилки» в комплект приборов на столе (либо удаление одного лишнего «философа» из-за стола) исключает возникновение тупика (полной взаимной блокировки), а двойной комплект «вилок» – также и «голодание».
Это решение не представляет большого интереса для теории, но эффективно на практике, с поправкой на его стоимость.

8.4.6 Системы массового обслуживания (СМО)

Абстрактная система, обрабатывающая поток входных ***заявок*** – запросов на выполнение некоторых действий (обработку).

СМО состоит из последовательности ***ступеней***. Ступени включают один или несколько исполнительных узлов (***блоков обработки***) и ***буфер*** для заявок.



Фрагмент системы массового обслуживания (СМО)

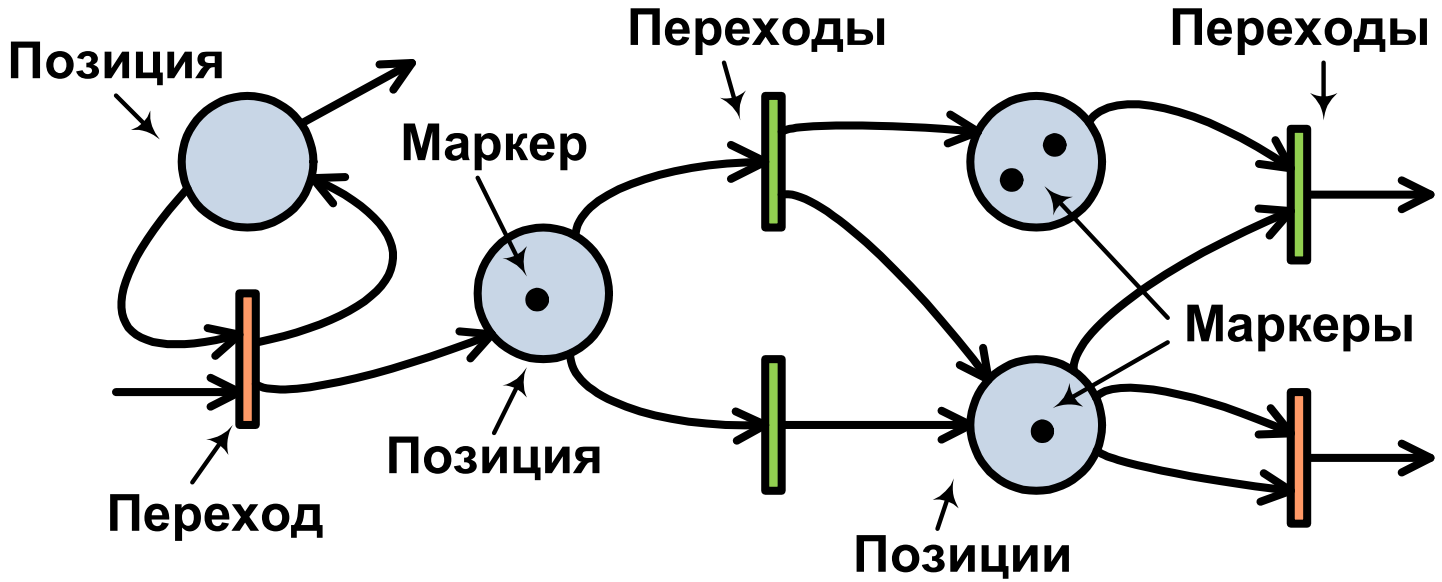
Примеры описываемых систем:
– производственный конвейер
– процессор, выполняющий поток инструкций
– разработчики и тестировщики, занятые проектами
– эшелонированная система ПВО
и т.д.

Многочисленные разновидности СМО:
– с непрерывным (Q-схемы) или дискретным (P-схемы) модельным временем, смешанные («агрегатные» схемы)
– с отказами, блокировками, накоплением заявок
и т.д.

СМО различного типа описываются соответствующими математическими моделями – аналитическими и имитационными. Широко используются в теории массового обслуживания.

8.4.7 Сети Петри

Модель системы с множеством процессов, протекающих параллельно и асинхронно.



Фрагмент сети Петри

Сеть Петри представляет собой ориентированный двудольный граф с вершинами двух типов:

– «***Позиции***» – можно рассматривать как состояния некоторой части системы

– «***Переходы***» – управляют изменениями состояний.

Кроме того, модель включает в себя дуги и маркеры (метки, токены, «фишки»).

«***Маркеры***» – отражают количественную (дискретную) характеристику состояния позиций. Маркеры передаются между позициями посредством переходов.

Переходы срабатывают асинхронно и независимо друг от друга, в соответствии с заданными правилами: с некоторой вероятностью, через интервалы времени с заданным законом распределения и т.д.

Условие срабатывания: наличие маркеров на каждой входящей дуге перехода.

Результат срабатывания: каждая входящая дуга изымает маркер из соответствующей входной позиции, каждая исходящая дуга добавляет маркер в соответствующую выходную позицию. Таким образом, может происходить «размножение» и «поглощение» маркеров.

Разметка (***маркировка***) сети – ее состояние: текущее количество маркеров во всех позициях (и, соответственно, разрешение или запрет срабатывания каждого из переходов).

Тупик – разметка, при которой нет ни одного разрешенного перехода.

Процесс моделирования: начальная разметка и последовательность шагов имитационного моделирования с наблюдением состояния сети. В ряде случаев возможна аналитическая модель.

Типичные решаемые на сети Петри задачи (доказываемые свойства сети):

– ***Достижимость*** состояния – возможность достигнуть заданной разметки из текущей

– «***Живость***» сети – отсутствие тупиков при любой возможной эволюции текущей разметки
и др.

Сети Петри хорошо формализуются, имеют хорошо наработанный математический аппарат. Обычно используются модификации сетей Петри: разнородные вершины, задержки срабатывания, отдельные управляющие входы, и т.д.