Конфликты на ресурсах

Совокупность параметров системы, на изменение которых сформулированы некоторые ограничивающие условия, называется *ресурсом*. *Обозначим R*.

Захват ресурса R процессом Z означает получение разрешения процессу Z изменять значения параметров $q \subset R$

Конфликт на ресурсе есть возникновение ситуации, когда тому или иному процессу отказано в захвате ресурса до момента выполнения некоторого заданного условия.

(т.е. необходимо добиться согласования процессов в этих объектах R)

Способы разрешения конфликтов

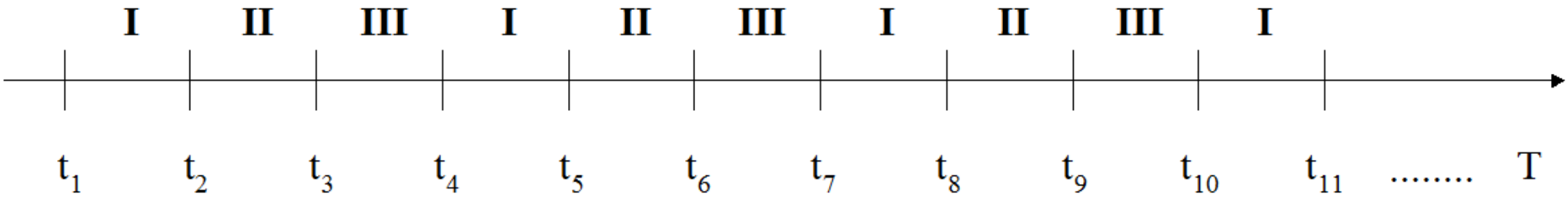
Синхронизация (time-sharing)

- это способ построения согласованных процессов в виде разнесения во времени их интервалов определения для объекта R.

Для этого задаются (возможно периодически повторяющиеся) интервалы времени захвата ресурса R для каждого претендующего на него процесса.

Алгоритм реализуется так: каждой заявке отводится определенное количество квантов времени (например,3 мс), в течение которых заявка может монопольно занимать процессорное время. После того как заданный интервал времени истекает, управление передается следующей заявке.

Здесь I, II, III обозначены процессы Z1, Z2, Z3



Способы разрешения конфликтов

Семафор *(логический, кооперативность)*

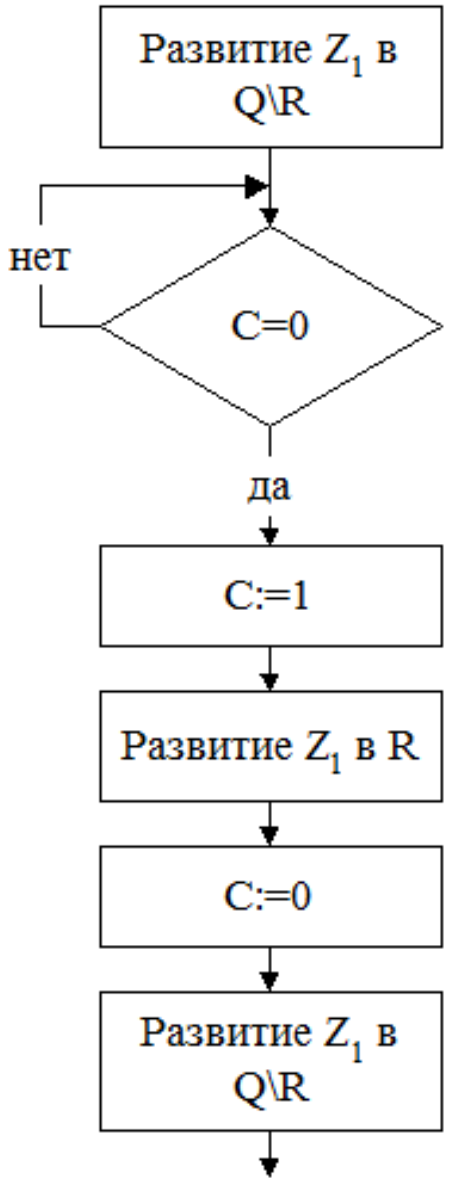
- это простая логическая переменная однозначно соответствующая ресурсу.

Будем полагать, что значение семафора ‘0’ означает, что ресурс может быть захвачен процессом, значение семафора ‘1’ запрещает захват ресурса.

Процесс, получивший управление ресурсом, выполняется до тех пор, пока сам по своей инициативе не передаст управление другому процессу *(кооперативная многозадачность).*

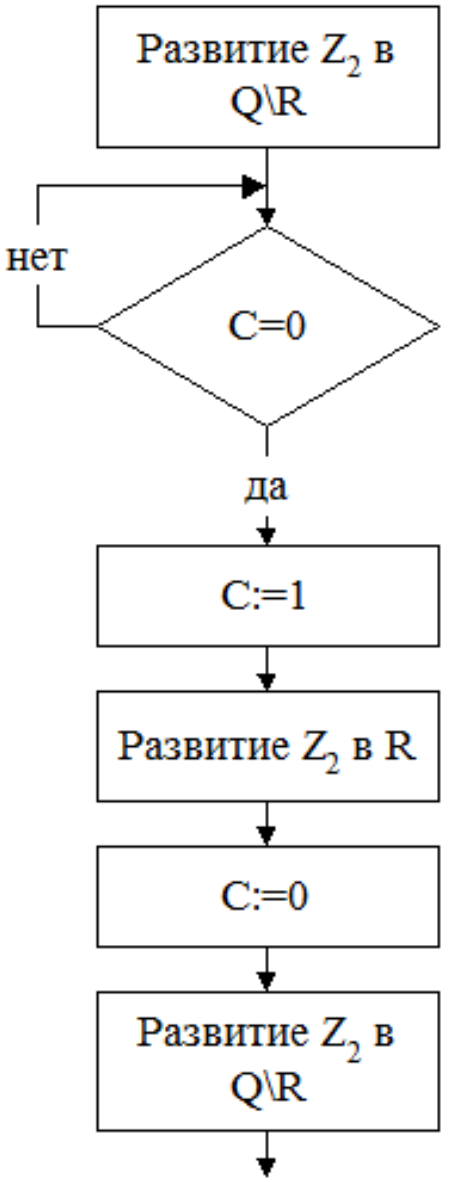
Применение семафоров для управления захватом ресурсов широко используется в системах управления.

Способы разрешения конфликтов



Семафор

критическая секция



Способы разрешения конфликтов

Семафор

- Бинарные семафоры только открывают или закрывают вход в критическую секцию (например, mutex);

- Арифметические семафоры позволяют определить, сколько процессов могут одновременно войти в критическую секцию:

    Если счетчик = 0, то вход в критическую секцию не возможен;

    Если счетчик > 0, то процесс может войти в свою критическую секцию.

На семафоре обычно реализуют активное ожидание, т.е. процесс на семафоре не блокируется, а продолжает проверять, не открылся ли семафор.

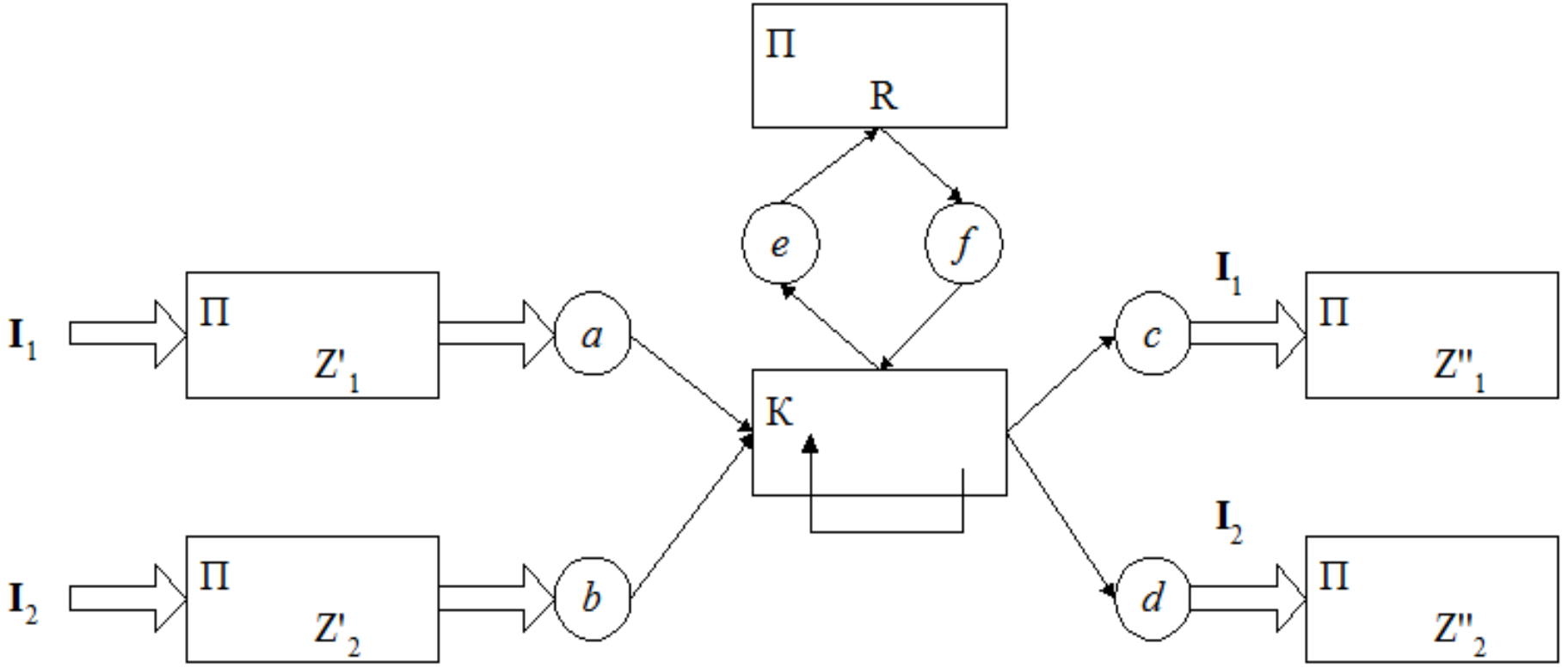
Семафоры живут как разделяемые глобальные переменные *(внимание!)*

Семафоры используются как для решения задачи взаимного исключения, так и для координации действий (синхронизации).

Способы разрешения конфликтов

Контроллер *(логический)*

- это способ управления процессами при захвате ресурсов, который состоит в создании соответствующих К-блоков. Использование К-блоков является универсальным способом управления множеством процессов при использовании ресурсов.



Варианты алгоритмов Контроллера Ресурсов

Кольцевая система очередей (round robin, карусель)

- очереди с независимыми источниками заявок обслуживаются «по кругу», например:

• по одной заявке из каждой непустой очереди;

• до исчерпания текущей очереди;

• до исчерпания заявок, скопившихся в текущей очереди к *моменту начала* ее обслуживания; т.е. следует запоминать момент начала обслуживания текущей очереди и сравнивать его с моментами прибытия находящихся в ней заявок.

Варианты алгоритмов Контроллера Ресурсов

Статическая относительная приоритизация

- в этом случае организуют отдельную очередь по каждому классу приоритета.

При завершении обслуживания очередная заявка выбирается из головы непустой очереди с наивысшим приоритетом

(*т.е. низкоприоритетные задачи могут никогда не получить управление!)*

Обобщением этой стратегии являются *альтернирующие* приоритеты, когда текущая очередь обслуживается до ее полного исчерпания.

Варианты алгоритмов Контроллера Ресурсов

Динамическая приоритизация

В случае динамических (переменных во времени) приоритетов обслуживание заявок обычно ведется без прерываний, а диспетчерские приоритеты заявок растут по времени ожидания, как правило, линейно с коэффициентами $\{\beta_j\}$ и стартовыми вкладами $\{\gamma_j\}$, убывающими по мере понижения базового приоритета.

В этом случае следует хранить неупорядоченную общую очередь, но в локальной среде каждой заявки фиксировать ее тип и момент прибытия.

При освобождении ресурса для всех заявок очереди вычисляется диспетчерский приоритет и при необходимости обновляются значение и позиция текущего максимума.

По окончании просмотра заявка с максимальным диспетчерским приоритетом выбирается на обслуживание, а остальные за ней в очереди сдвигаются вперед.

Мультиресурсные приоритетные системы

При моделировании мультиресурсных систем необходимо учитывать число и номера занятых ресурсов, а также возможную их дифференциацию по типам принимаемых заявок и характеристикам распределения времени обслуживания.

Учет моментов освобождения ресурсов ведется аналогично обработке моментов поступления заявок неоднородного рекуррентного потока общего вида.

При однородных ресурсах простейшей дисциплиной выбора является «первый свободный», выявляемый по моменту его освобождения.

При нескольких свободных ресурсах может потребоваться *равновероятный* выбор из них. Например, если таких каналов *m*, то для определения выбираемого канала следует вычислить $N = целое[m*U] +1$, где U - случайное число [0;1). Далее выполняется просмотр моментов освобождения ресурсов, и для каждого свободного в счетчик добавляется единица. Заявка передается в тот ресурс, для которого содержимое счетчика равняется N.

Мультиресурсные приоритетные системы

Многоуровневое квантованное обслуживание (прерывание)

- в этом случае появляется новый тип события: исчерпание кванта текущего обслуживания. Соответственно прерванная заявка перемещается в конец следующей, менее приоритетной очереди, где она будет дожидаться предоставления большего кванта.

Приоритетная многозадачность с вытеснением (прерывание)

- для новой заявки генерируется длительность обслуживания в момент ее прибытия. Приоритет вновь прибывшей заявки сравнивается с приоритетом обслуживаемой и определяется необходимость прерывания. Прерванная заявка помещается в начало очереди своего приоритета с новым временем (режим дообслуживания).

Мультиресурсные приоритетные системы

В регулировании систем с приоритетным обслуживанием важную роль играют *периоды непрерывной занятости* (**ПНЗ**) системы заявками более высоких приоритетов – поскольку на это время система недоступна для обслуживания заявок более низких приоритетов.

Для управления такими ситуациями при прибытии в *n*-канальную систему высокоприоритетной заявки, увеличивающей их текущее количество до *n*, фиксируется начало **ПНЗ** /=значение таймера/.

При завершении обслуживания и уменьшении текущего числа заявок до *n-1* отмечается окончание **ПНЗ** и к счетчикам (для вычисления моментов) добавляется его длительность . Счётчик событий **ПНЗ** инкрементируется.

Мультиресурсные контроллеры

Иногда для начала обслуживания заявки необходимо одновременное наличие ***ресурсов нескольких видов***, имеющихся в ограниченном объеме.

В подобных случаях возможна ситуация типа известных из теории операционных систем «смертельных объятий» (***deadlocks***): когда несколько заявок из очереди частично обеспечили себя, захватив ресурсы разных видов. Но они взаимно заблокировались, что приводит к параличу всех процессов обслуживания.

Для предотвращения таких блокировок можно, например -

1) **исключить назначение неполных комплектов**;
2) **ввести приоритеты выделяемых ресурсов**.

Способы разрешения конфликтов

В сложной системе алгоритмы диспетчеризации *могут меняться* в процессе функционирования.

Существуют системы, где каждая отдельная задача может участвовать в одном из трех алгоритмов планирования или их комбинации (*прерывание, разделение времени, кооперативность*).

Для решения проблем «застоя» в операционных системах офисного назначения применяется «*разгон приоритетов*»: операционная система периодически выбирает из «хвоста» очереди давно не выполняющиеся задачи и присваивает им на один квант времени максимальный приоритет.
В системах реального времени применяется «*наследование приоритетов*»: всем задачам, конкурирующим за общий ресурс, кратковременно присваивается один и тот же приоритет, равный максимальному среди всех таких задач.

Нестационарные процессы

Значительная часть проблем систем массового обслуживания относится к нестационарным режимам, в том числе с *периодическими* режимами.

Многие задачи управления связаны с суточными ритмами (транспортные потоки, клиенты, дорожно-транспортные происшествия, работа энергосетей).

Календарный ритм имеют процессы развития заболеваний, лесных пожаров, наводнений, горных лавин, обслуживания клиентов, ледовой проводки судов, обработки судов в портах.

Непериодическими нестационарными процессами являются двусторонние боевые действия (морской бой, ракетная дуэль), оказание медицинской помощи в ходе масштабной антитеррористической операции, борьба с эпидемией, аварийно-спасательная деятельность при землетрясении (в частности, с учетом «афтершоков») и т. д. Численно-аналитические методы расчета подобных ситуаций известны только для системы M/G/1 и чрезвычайно сложны.

Пример задачи разрешения конфликтов

Пять безмолвных философов сидят вокруг круглого стола, перед каждым философом стоит тарелка спагетти. Палочки для еды лежат на столе между каждой парой философов. Каждый философ может либо есть, либо размышлять. Приём пищи не ограничен количеством оставшихся спагетти — подразумевается бесконечный запас. Тем не менее, философ может есть только тогда, когда держит две палочки — взятую справа и слева.

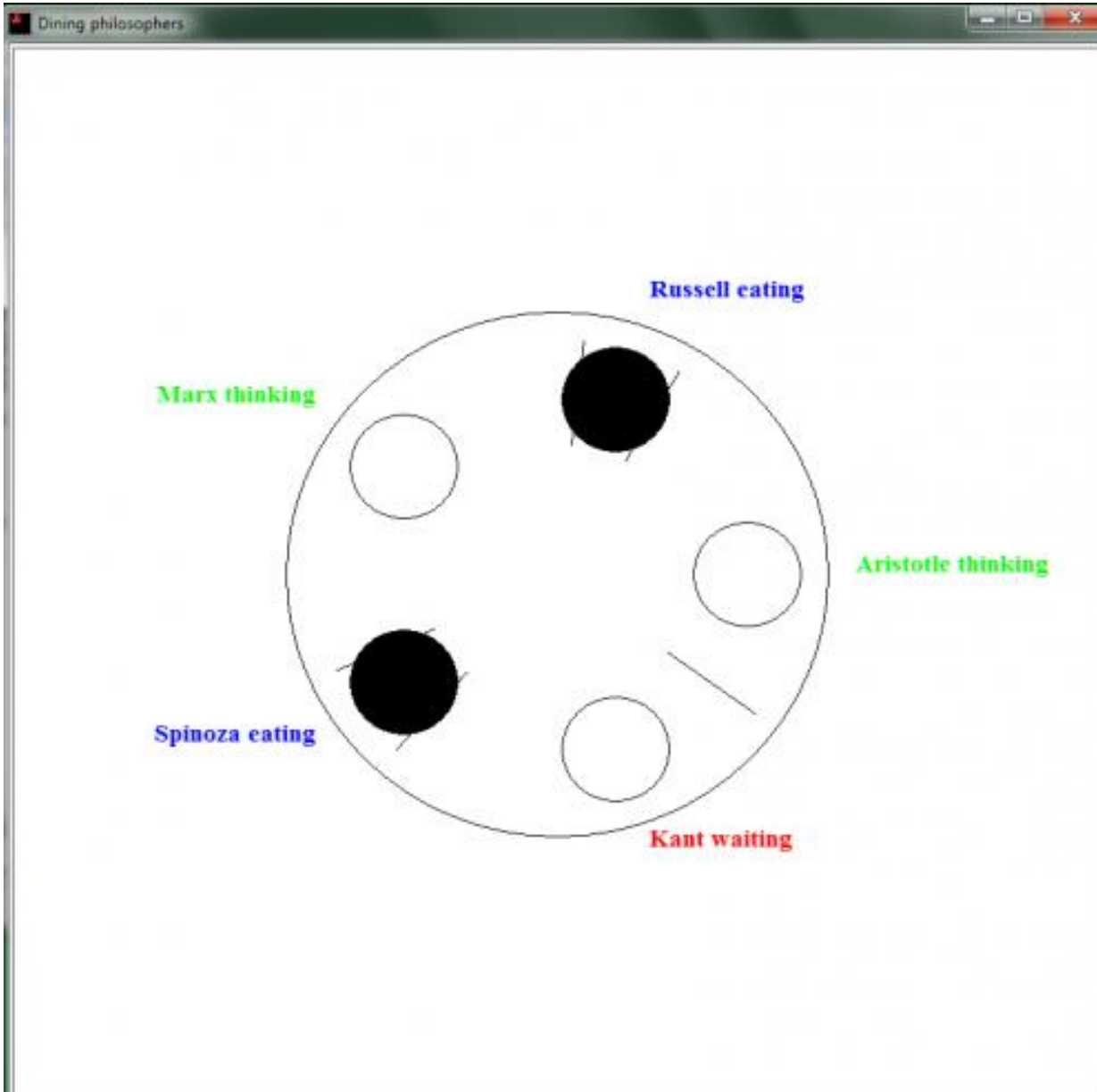
Каждый философ может взять ближайшую палочку (если она доступна) или положить, если он уже держит её. Взятие и возвращение на стол являются раздельными действиями, которые должны выполняться одно за другим. Временные интервалы размышления и еды случайны, действия философов не синхронизированы.

Требуется разработать параллельный алгоритм, при котором ни один из философов не будет голодать, т.е. будет чередовать приём пищи и размышления.

Пример задачи разрешения конфликтов

Например, можно каждому философу дать выполнять следующий алгоритм:

1) Размышлять некоторое время;
2) Ждать пока не освободится палочка слева и взять её;
3) Ждать пока не освободится палочка справа и взять её;
4) Есть спагетти некоторое время;
5) Положить левую палочку;
6) Положить правую палочку;
7) Повторить алгоритм сначала.

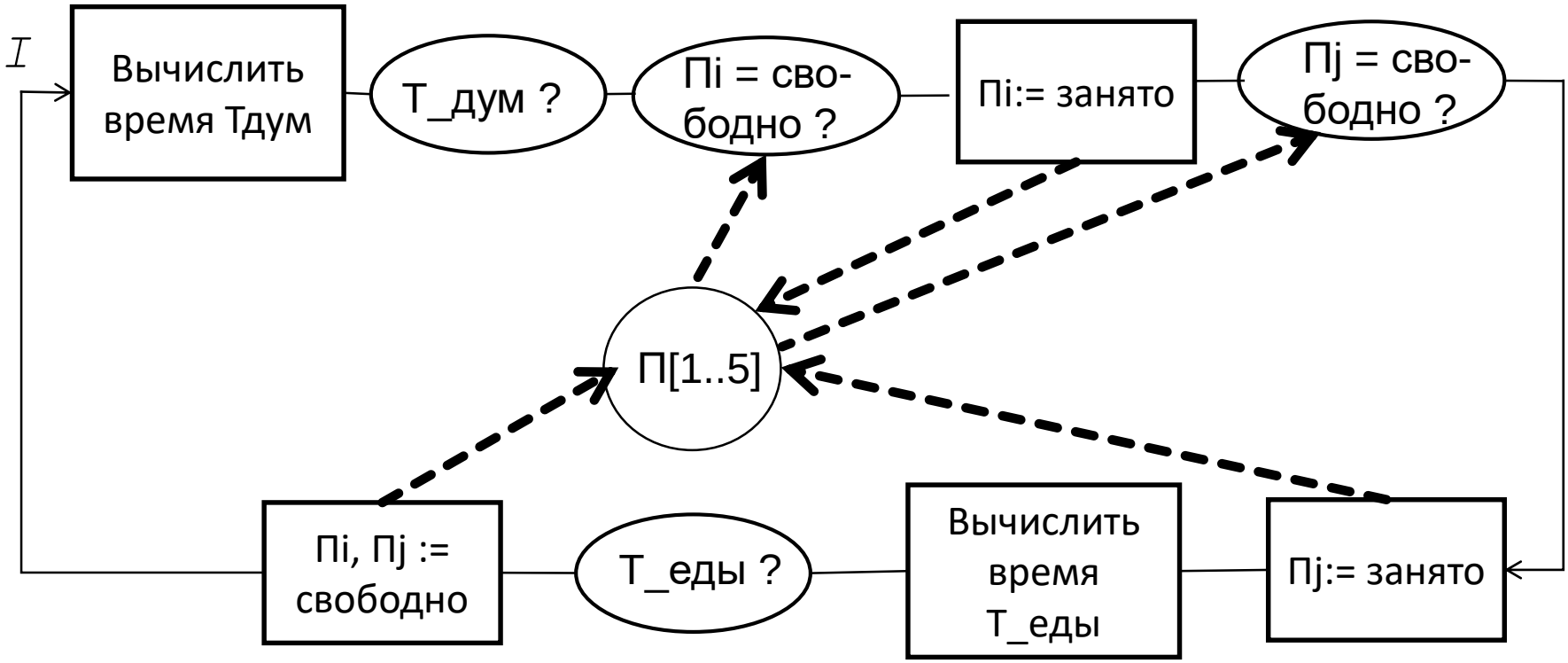


Dining philosophers problem

by E. W. Dijkstra

О П С блока Философ

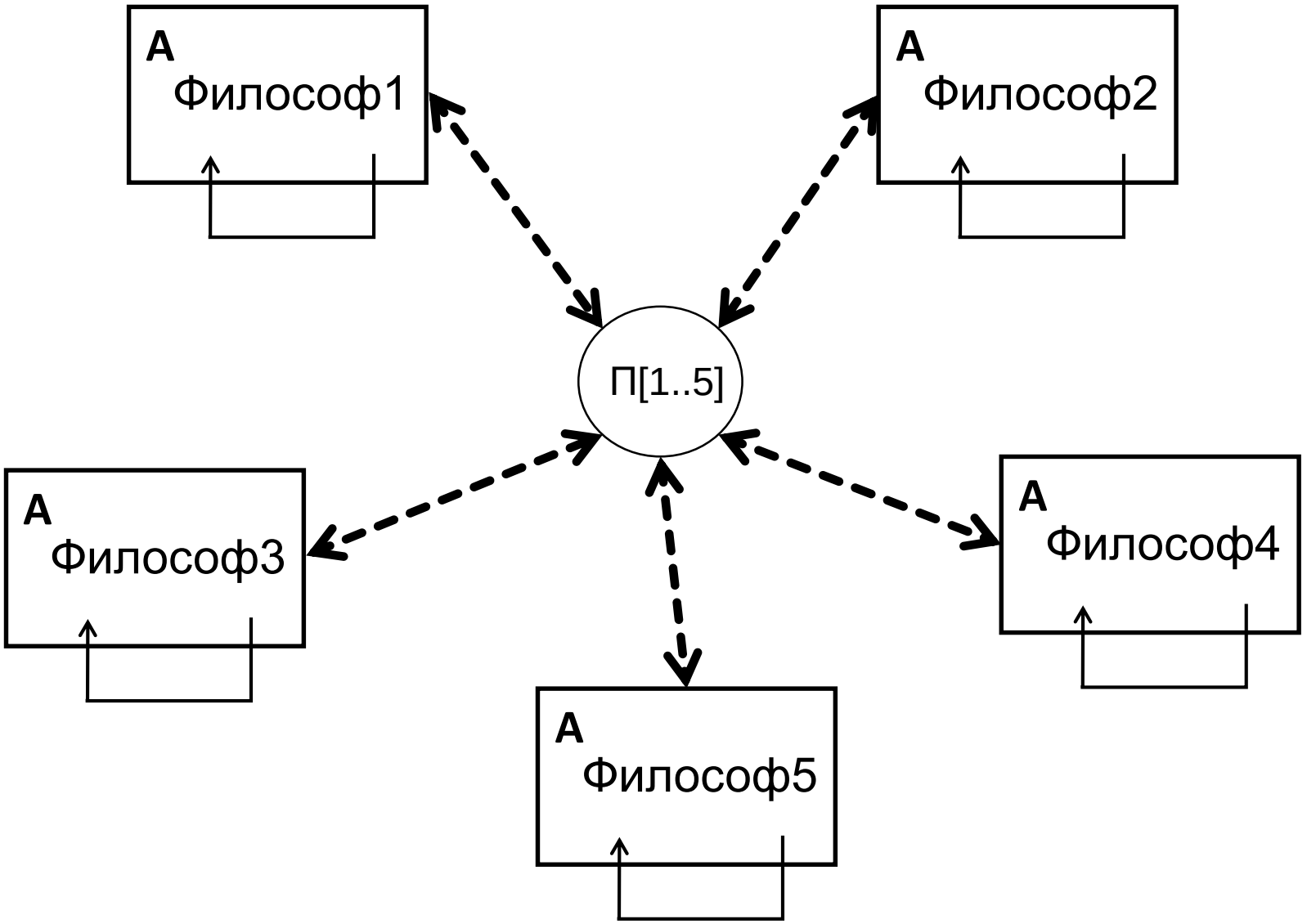
Философ(*i*) *i=1...5*



Пi ∈ {*свободно; занято*}

j = 1...5
если i= 1 *то* *j=5*
иначе *j = i - 1*

Блочная схема алгоритма модели



Способы разрешения конфликтов

Такое решение задачи некорректно: оно позволяет системе достичь состояния взаимной блокировки, *тупика* (**deadlock**), когда каждый философ взял инструмент слева и ждёт, когда инструмент справа освободится.

Проблема ресурсного голодания, *застоя* (**resource starvation**) может возникать независимо от взаимной блокировки, если один из философов не сможет завладеть левой и правой палочкой из-за проблем синхронизации.

Тупик представляет собой ситуацию более опасную, чем *застой*: процессы, попавшие в *тупик*, удерживают при этом системные ресурсы. Если *тупик* не глобальный, система продолжает работать с уменьшенным объемом ресурсов, с пониженной производительностью. *Застой* одного процесса может и не повлиять на среднюю пропускную способность системы, но влияет на общие показатели обслуживания.

Способы разрешения конфликтов

Для предупреждения проблем можно предложить правило, согласно которому философы должны класть инструмент обратно на стол после пятиминутного ожидания доступности другого инструмента, и ждать ещё пять минут перед следующей попыткой завладеть инструментами.

Такая схема устраняет возможность блокировки, но здесь существует возможность «зацикливания» системы (***livelock***), при котором состояние системы меняется, но она не совершает никакой полезной работы.

Например, если все пять философов одновременно возьмут левый инструмент в одно и то же время, то философы будут ждать пять минут в надежде завладеть правым, потом положат левый и будут ждать ещё пять минут прежде, чем попытаться завладеть инструментом снова.

Способы разрешения конфликтов

Относительно простое решение задачи достигается путём добавления «официанта возле стола» и доступности всем всех инструментов.

Философы должны дожидаться разрешения официанта перед тем, как взять инструмент. Поскольку официант знает (считает), сколько инструментов используется в данный момент, он может «принимать решения» относительно распределения инструмента, и тем самым, предотвратить взаимную блокировку философов. Если четыре из пяти уже используются, то следующий философ, запросивший инструмент, вынужден будет ждать разрешения официанта. Оно не будет получено, пока какой-либо инструмент не будет освобожден. Предположим, что философ всегда пытается сначала взять инструмент слева, а потом — справа.

Тогда «Официант» реализуется как арифметический семафор.

Способы разрешения конфликтов

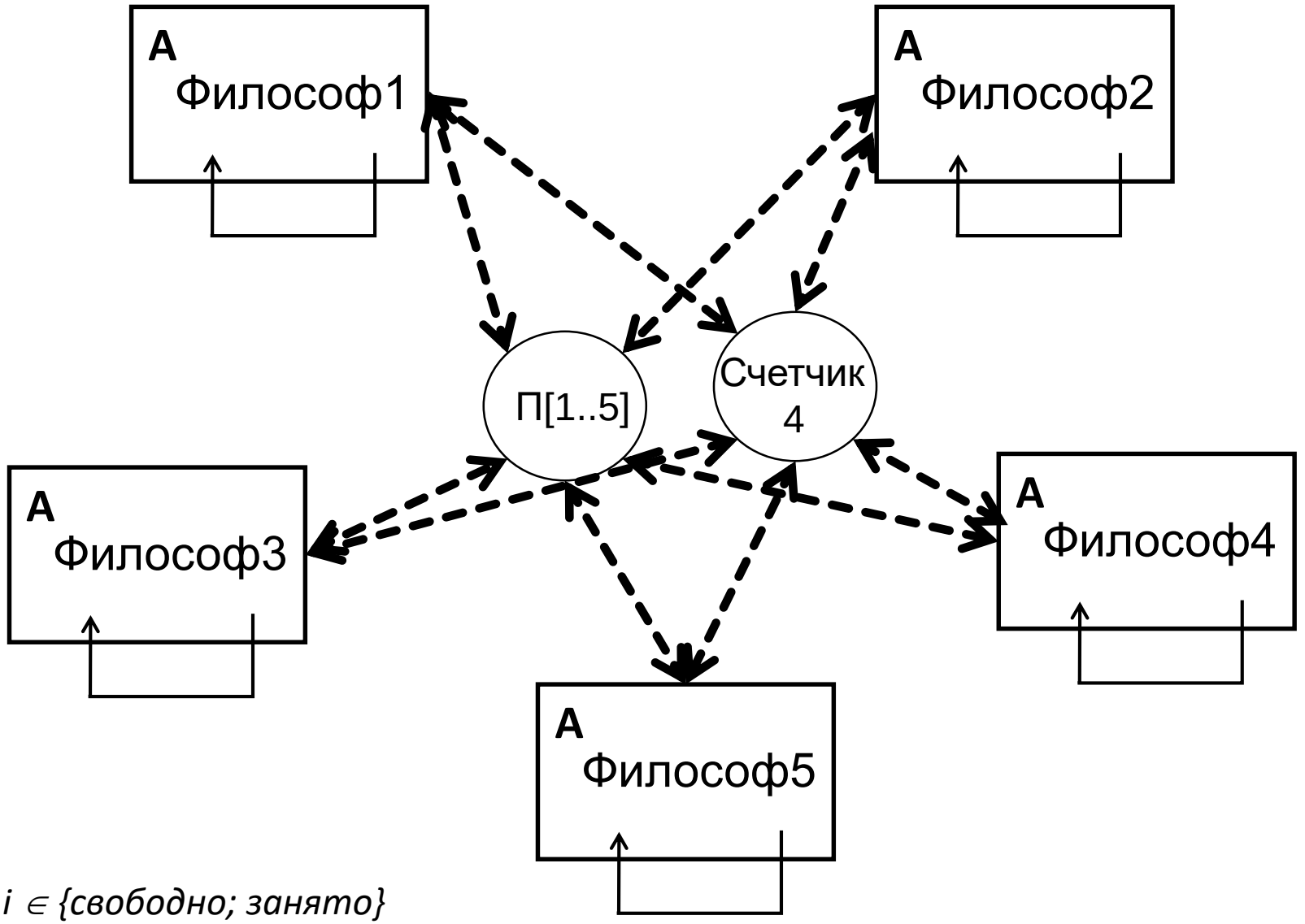
Предположим, что философы обозначены от А до Д по часовой стрелке. Если философы А и В едят, то заняты четыре инструмента.

Философ Б сидит между А и В, так что ему недоступен ни один инструмент. В то же время, философы Г и Д имеют доступ к одному неиспользуемому инструменту между ними.

Предположим, что философ Г хочет есть. Если он тут же берёт свободный инструмент, 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 инструментов освободится, то по крайней мере один философ сможет взять два инструмента.

Таким образом, взаимная блокировка становится невозможной.

Схема модели с семафором



П*i* $\in$ *{свободно; занято}*

Счетчик4 $\in$ *{0;1;2;3;4}*