Операционные системы

Межпроцессное взаимодействие

Виды межпроцессного взаимодействия (IPC)

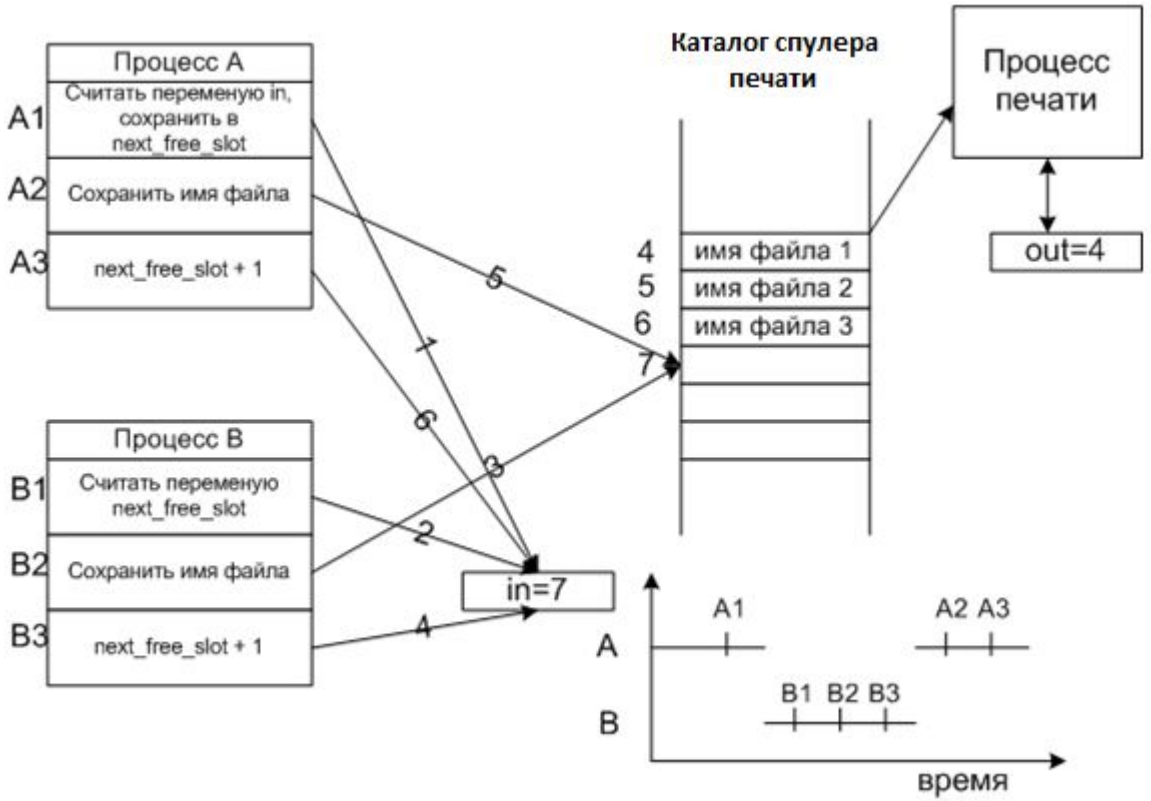
- Предотвращение критических ситуаций
- Синхронизация процессов
- Передача информации от одного процесса другому

Межпроцессное взаимодействие

Предотвращение критических ситуаций и средства синхронизации процессов

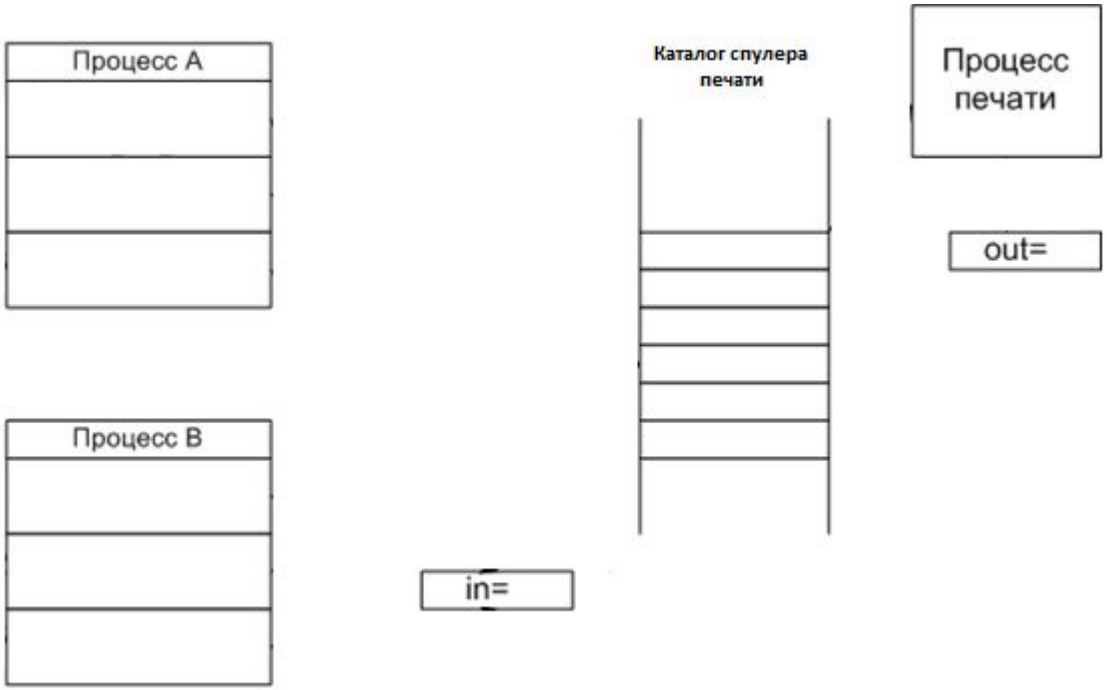
Пример возникновения гонок (состязаний)

- Рассмотрим в качестве примера возникновения гонок (состязаний) ситуацию, когда несколько процессор имеют общий доступ на чтение и запись к некоторым данным.



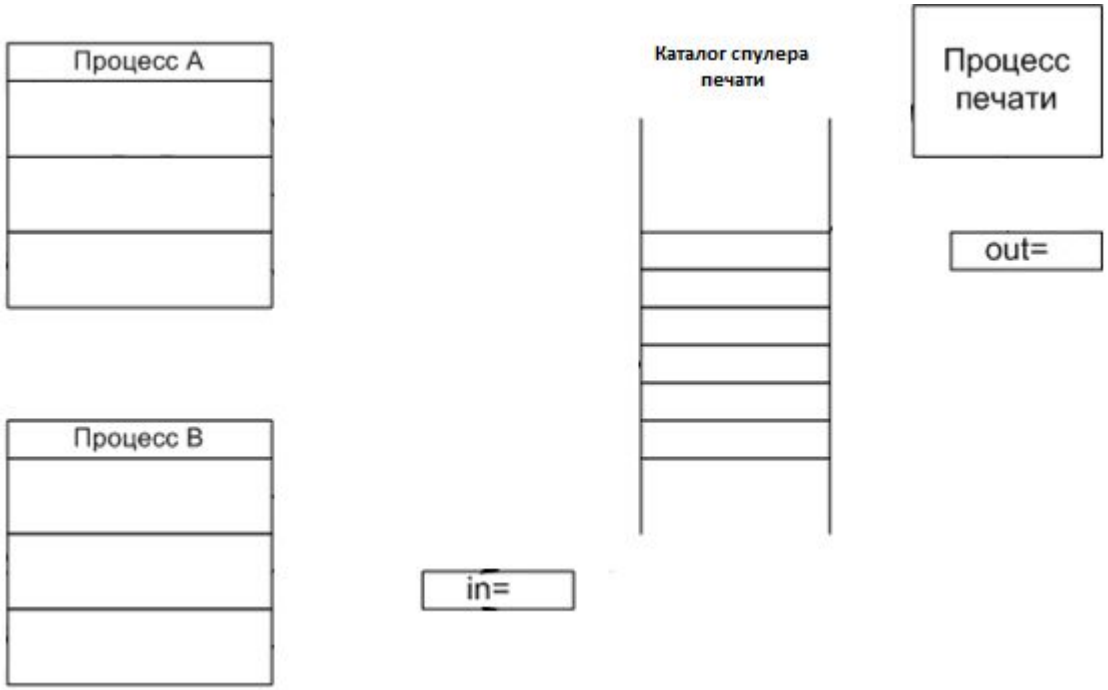
Пример возникновения гонок – каталог спулера печати

- Процессы A и B выполняют задание на печать файлов, размещая имена файлов в специальном каталоге спулера печати.



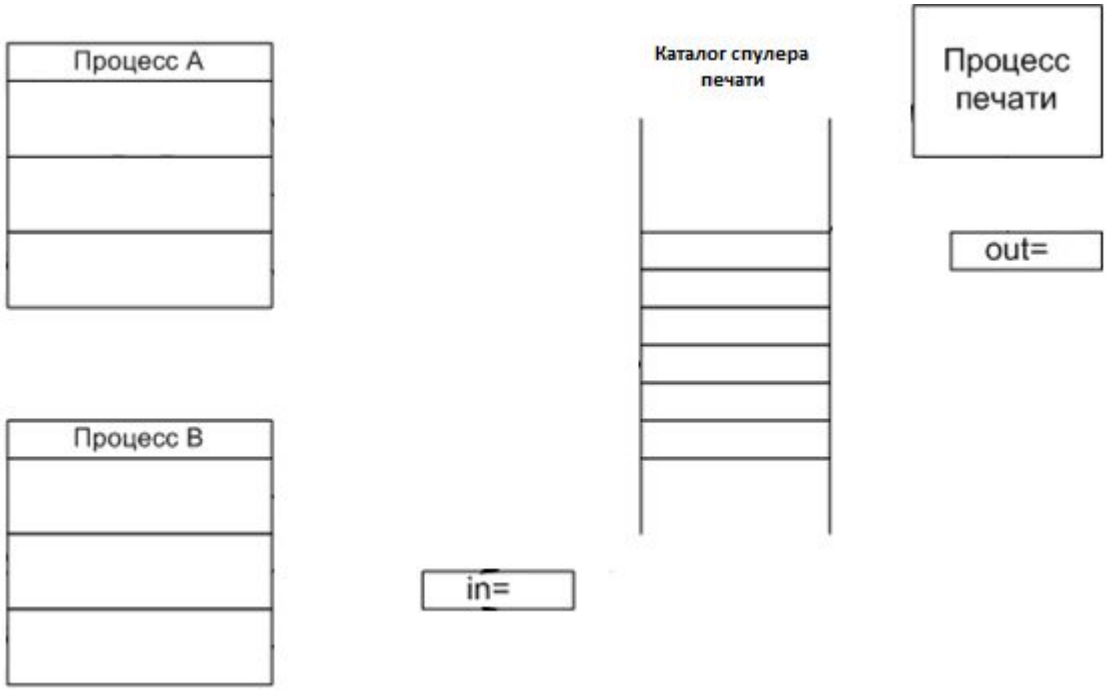
Пример возникновения гонок – процесс печати

- Процесс печати периодически проверяет наличие файлов в каталоге спулера, которые нужно печатать, печатает их и удаляет из каталога.



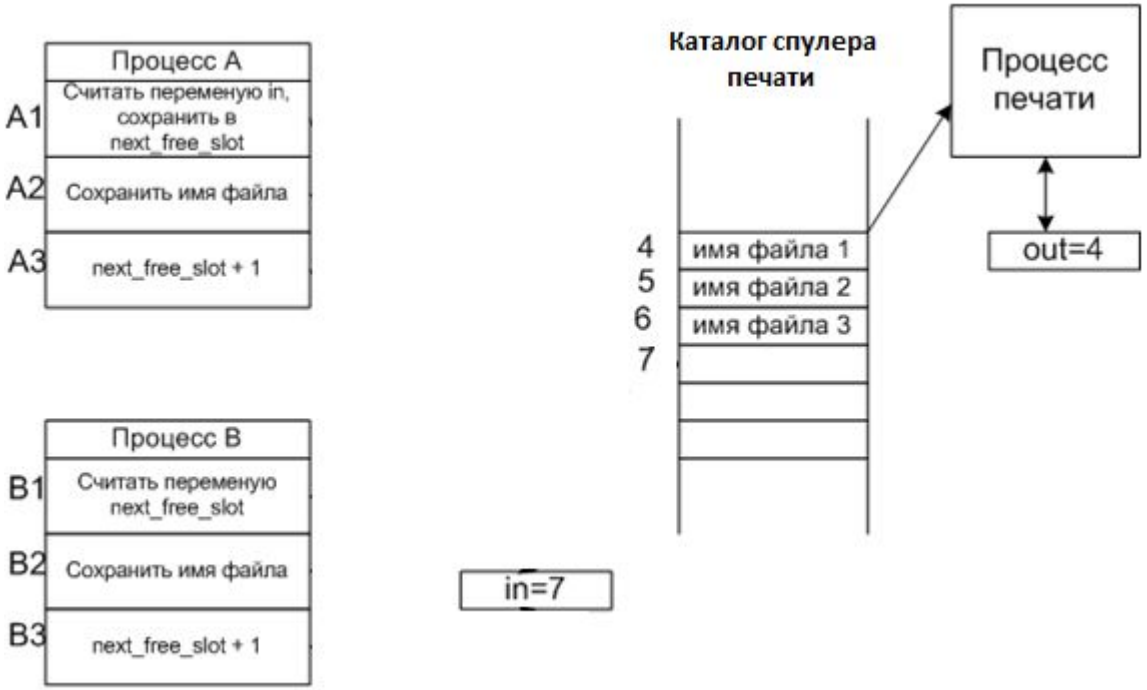
Пример возникновения гонок – общие переменные

- Две совместно используемые переменные хранят индексы следующего файла для печати (*out*) и следующего свободного сегмента (*in*).



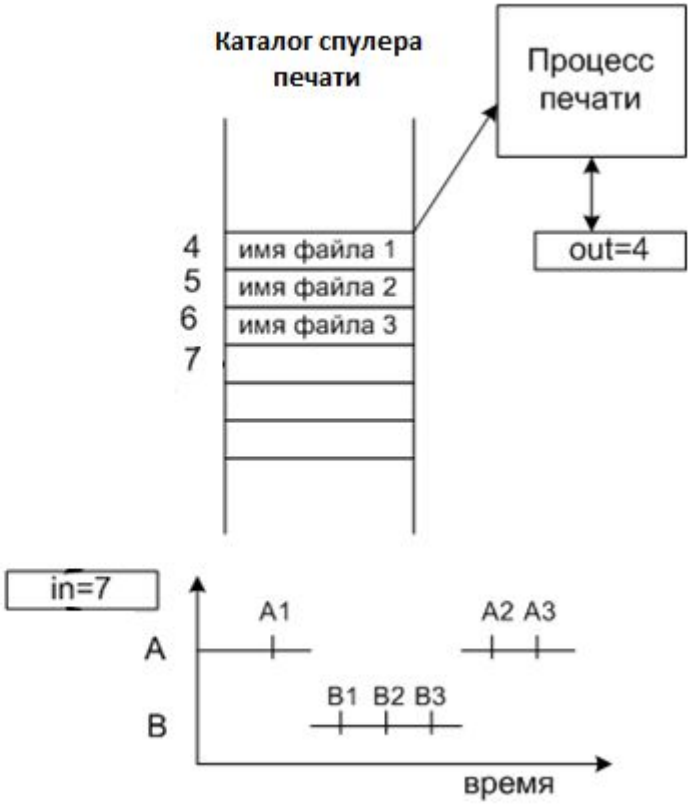
Пример возникновения гонок – описание задачи

- Пусть сегменты каталога с 0 по 3 пусты, а сегменты 4-6 заняты файлами, которые ждут печати, поэтому **in =7**, а **out =4**.



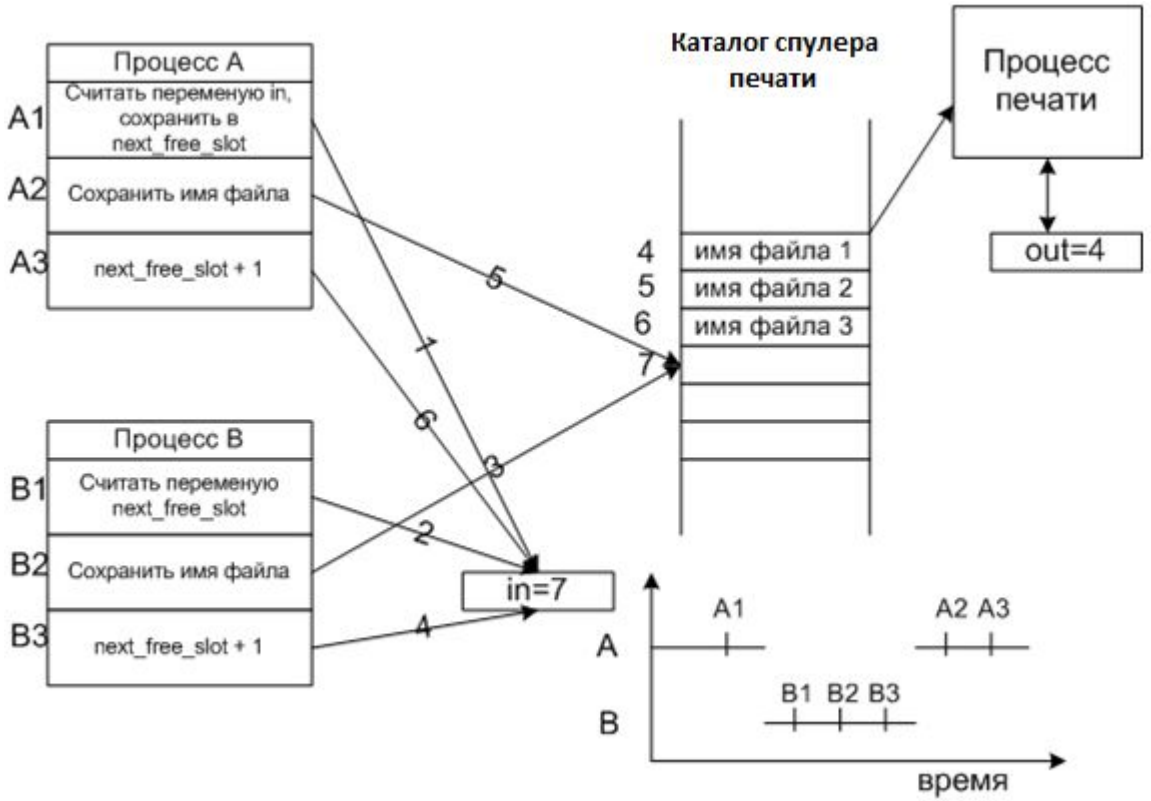
Пример возникновения гонок – выполнение

- Процессы A и B начинают выполнять свои команды A1-A3 и B1-B3, как это показано на графике.



Пример возникновения гонок – результат

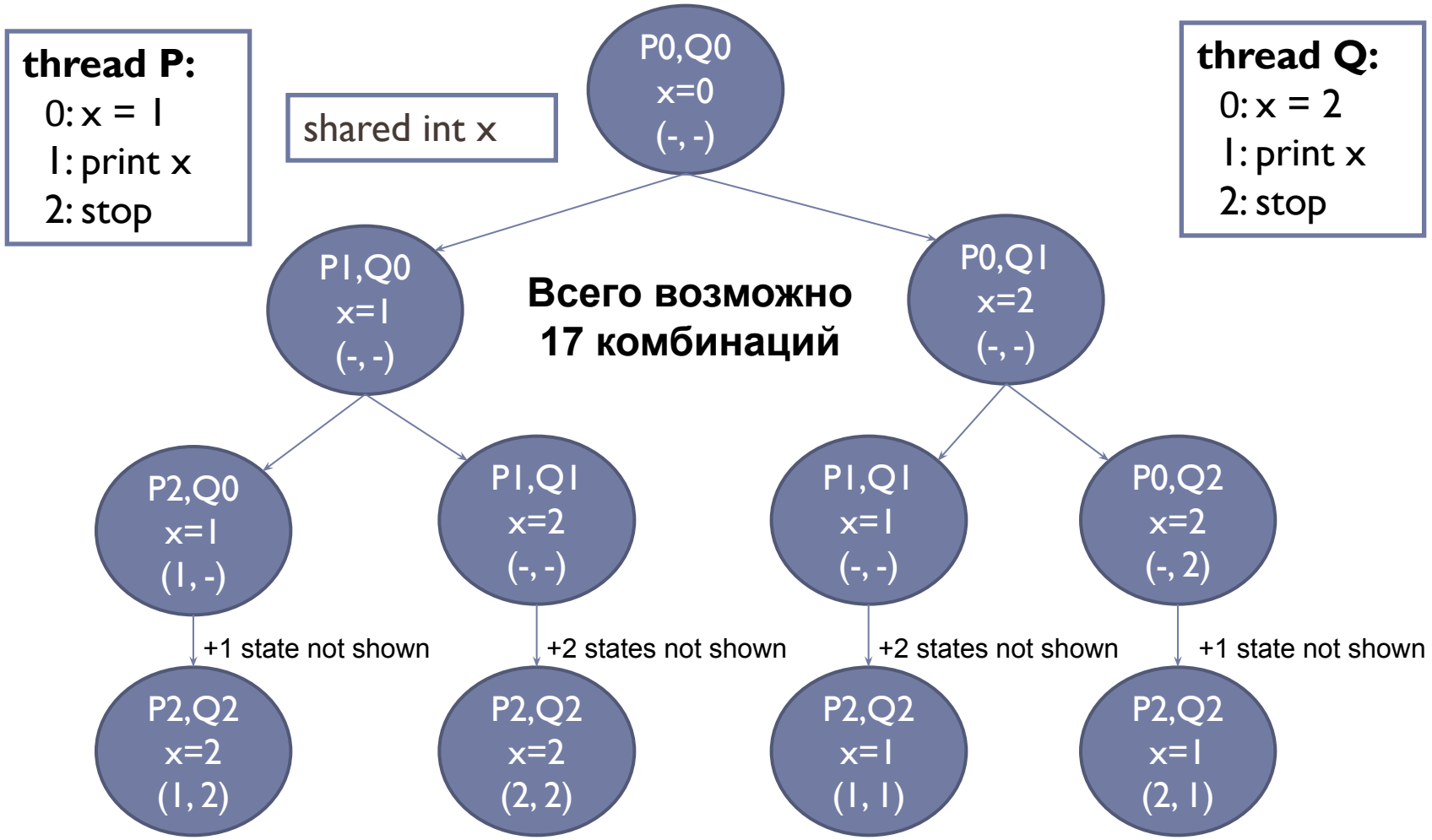
- При выполнении своего кода процессы A и B записывают имена своих файлов в один и тот же сегмент 7, в результате файл процесса B напечатан не будет.



Пример возникновения гонок – комментарии

- Процесс A считывает значение (7) переменной in и сохраняет его в локальной переменной next_free_slot. После этого происходит прерывание по таймеру, и процессор переключается на процесс B. Процесс B, в свою очередь, считывает значение переменной in и сохраняет его (опять 7) в своей локальной переменной next_free_slot.
- Теперь оба процесса считают, что следующий свободный сегмент – седьмой.
- Процесс B сохраняет в каталоге спулера имя файла и заменяет значение in на 8, затем продолжает заниматься своими задачами, не связанными с печатью.
- Наконец управление переходит к процессу A, и он начинает с того места, на котором остановился. Он обращается к переменной next_free_slot, считывает ее значение и записывает в седьмой сегмент имя файла (удаляя значение, записанное процессом B).

Еще один пример гонок



Критические секции и данные

- Критическая секция – это часть программы, результат выполнения которой может непредсказуемо меняться, если переменные, относящиеся к этой части программы, изменяются другими потоками в то время, когда выполнение этой части еще не завершено.
- Критические данные – общие переменные, которые изменяются двумя или более потоками в состязательном режиме.
- В общем случае в разных потоках критическая секция может состоять из разных последовательностей команд.

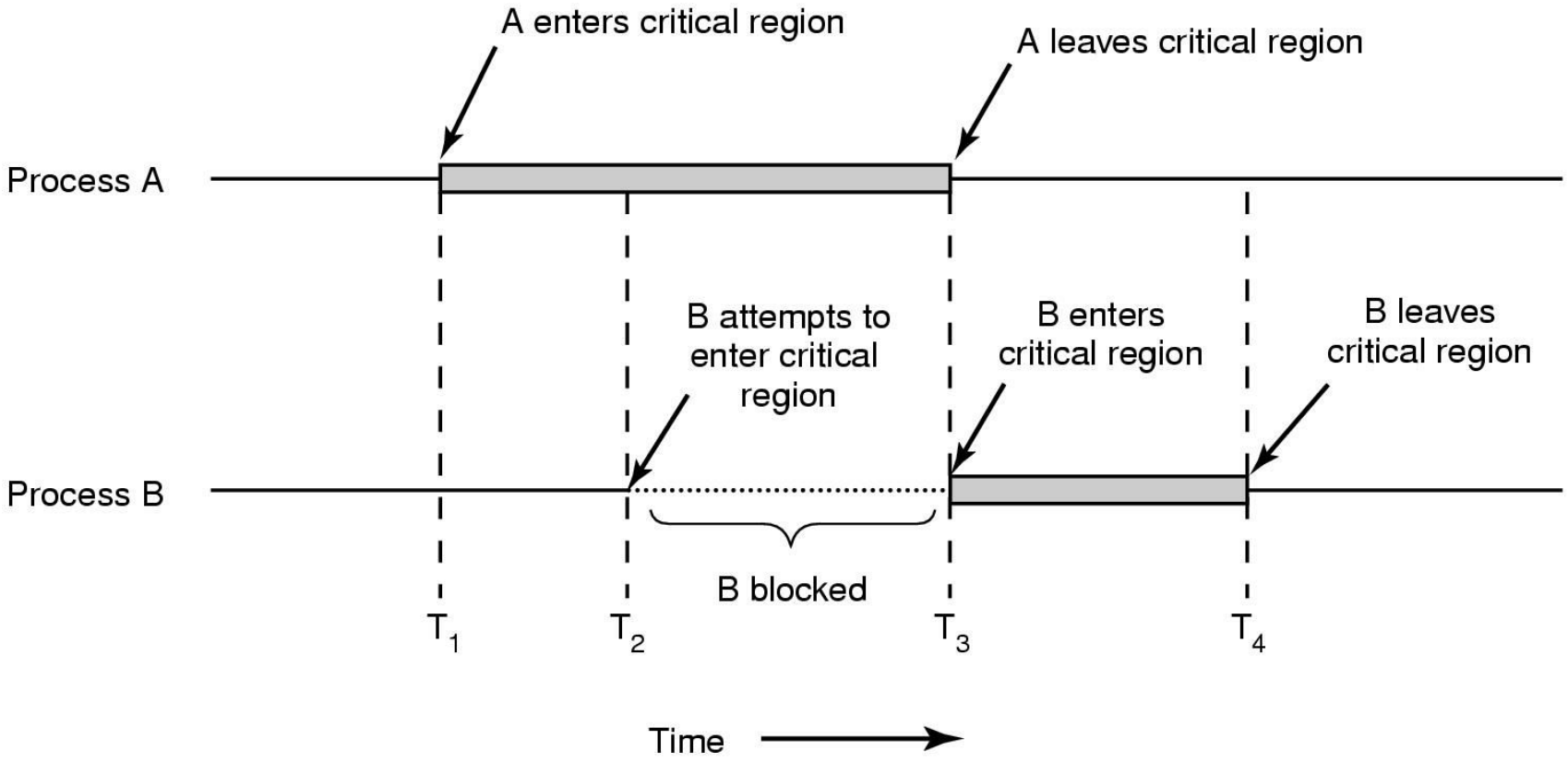
Простейшее решение проблемы возникновения гонок

- Самый простой и в то же время самый неэффективный способ обеспечения взаимного исключения состоит в том, что ОС позволяет потоку запрещать любые прерывания на время его нахождения в критической секции.
- Подобный способ редко применяется, т.к. по сути он схож с невытесняющей многозадачностью и грозит нарушением работоспособности всей вычислительной системы, если случится сбой пользовательского потока во время запрета прерываний.

Условия взаимного исключения гонок со стороны процессов

- В каждый момент времени в критических секциях, связанными с одними критическими данными, не должно находиться более одного процесса.
- Процесс вне критической секции не может блокировать другие процессы.
- Должна быть невозможна ситуация, когда процесс вечно ждет попадания в критическую секцию.
- В программе не должно быть предположений о скорости ее выполнения или количестве процессоров.

Пример решения взаимного исключения с использованием критических секций



Алгоритм Петерсона

- Алгоритм Петерсона – программная реализация механизма взаимного исключения без запрещения прерываний.
- Алгоритм Петерсона имеет смысл в системах на базе модели вычислений Фон-Неймана.
- Алгоритм Петерсона предложен в 1981 году Гарри Петерсоном из университета Рочестер (США). В основу алгоритма Петерсона был положен алгоритм Деккера.

Описание алгоритма Петерсона

- Алгоритм использует следующие общие переменные: у каждого процесса есть собственная переменная flag[i] и есть общая переменная turn. В переменной flag[i] запоминается факт попытки захвата ресурса, в переменной turn – номер процесса, которому предлагается захватить ресурс.
- При вступлении в критическую секцию процесс Pi заявляет о своей готовности выполнить критический участок (flag[i]) и одновременно предлагает второму процессу приступить к его выполнению (turn).
- Если процессы подошли к вступлению в КС практически одновременно, то они оба объявят о своей готовности и предложат захватить ресурс друг другу. При этом одно из предложений всегда следует после другого. Работу в критическом участке продолжит процесс, которому было сделано последнее предложение.

Пример реализации алгоритма Петерсона для двух процессов

```
flag[0]    = 0;
flag[1]    = 0;
turn       = 0;
```

```
P0: flag[0] = 1;
turn = 1;
while( flag[1] && turn == 1 );
// ждем
// начало критической секции
...
// конец критической секции
flag[0] = 0;
```

```
P1: flag[1] = 1;
turn = 0;
while( flag[0] && turn == 0 );
// ждем
// начало критической секции
...
// конец критической секции
flag[1] = 0;
```

Ограничения алгоритма Петерсона

- Алгоритм рассчитан только на 2 процесса (от этого ограничения свободен следующий алгоритм – алгоритм пекарни Bakery algorithm).
- При ожидании ресурса процессы не снимаются с очереди на обслуживание и впустую тратят процессорное время (активное ожидание).
- Алгоритм Петерсона учитывает отсутствие атомарности в операциях чтения и записи переменных и может применяться без использования команд управления прерываниями.

Алгоритм пекарни

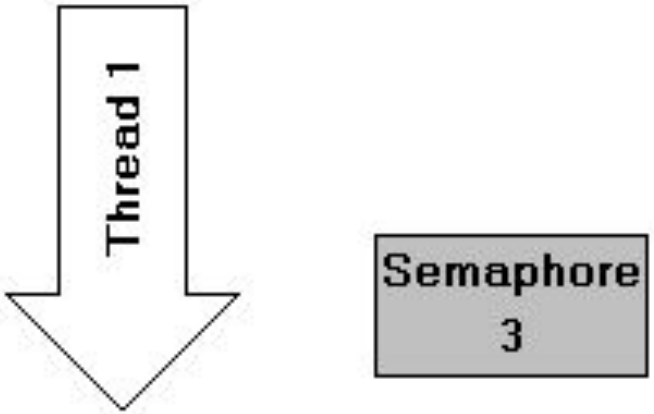
- Алгоритм пекарни – программная реализация механизма взаимного исключения без запрещения прерываний. Алгоритм пекарни имеет смысл в системах на базе модели вычислений Фон-Неймана. В отличии от алгоритма Деккера и Петерсона отсутствует ограничение на число процессов.
- Алгоритм состоит в том, что при входе каждый «клиент» получает табличку с уникальным номером. В один момент времени производится обслуживание только одного клиента. При выходе «клиент» табличку отдает. Первым обслуживается вошедший «клиент», имеющий минимальный номер. Так как операции не атомарные, одинаковый номер могут получить несколько «клиентов». В таком случае можно выбрать приоритетность «клиентов», например, по имени процесса.

Семафоры

- Дийкстра (Dijkstra) предложил использовать для синхронизации вычислительных процессов семафоры.
- Семафор – неотрицательная целая переменная S >= 0, которая может изменяться и проверяться только посредством двух примитивов:
  - V(S): переменная S увеличивается на 1 единым неделимым действием. К переменной S нет доступа другим потокам во время выполнения этой операции.
  - P(S): уменьшение S на 1, если это возможно. Если S=0 и невозможно уменьшить S, оставаясь в области целых неотрицательных значений, то в этом случае поток, вызывающий операцию P, ждет, пока это уменьшение станет возможным. Успешная проверка и уменьшение также являются неделимой операцией.

Иллюстрация работы семафора

- Пример демонстрирует использование семафора для ограничения доступа потоков к объекту синхронизации на основании их количества.



- Исходное состояние семафора равно 3, после доступа к семафору первых трех потоков четвертый поток будет заблокирован.

Использование семафоров

- Таким образом, семафоры позволяют эффективно решать задачу синхронизации доступа к ресурсным пулам, таким, например, как набор идентичных в функциональном назначении внешних устройств (модемов, принтеров, портов), или набор областей памяти одинаковой величины, или информационных структур.
- Во всех этих и подобных им случаях с помощью семафоров можно организовать доступ к разделяемым ресурсам сразу нескольких потоков.

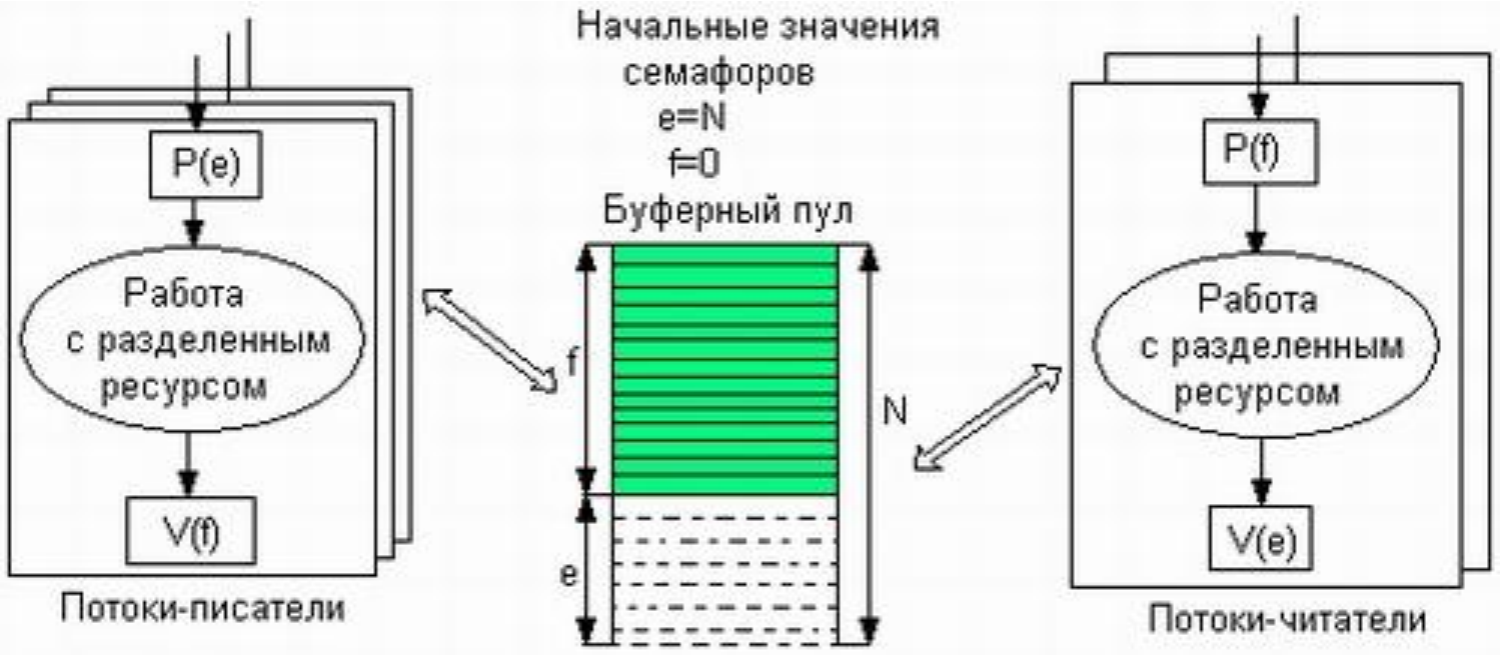
Мьютексы

- Иногда используется упрощенная версия семафора – мьютекс (mutex, mutual exclusion – взаимное исключение). Иногда называют еще двоичным семафором.
- Мьютекс – переменная, которая может находиться в одном из двух состояний: блокированном или неблокированном.
- Если процесс хочет войти в критическую секцию – он вызывает примитив блокировки мьютекса.
- Если мьютекс не заблокирован, то запрос выполняется и процесс попадает в критическую секцию.

Задача о читателях и писателях

- Рассмотрим использование семафоров на классическом примере взаимодействия двух выполняющихся в режиме мультипрограммирования потоков, один из которых пишет данные в буферный пул, а другой считывает их из буферного пула.
- Пусть буферный пул состоит из N буферов, каждый из которых может содержать одну запись. В общем случае поток-писатель и поток-читатель могут иметь различные скорости и обращаться к буферному пулу с переменой интенсивностью. В один период скорость записи может превышать скорость чтения, в другой – наоборот.
- Для правильной совместной работы поток-писатель должен приостанавливаться, когда все буферы оказываются занятыми, и активизироваться при освобождении хотя бы одного буфера. Напротив, поток-читатель должен приостанавливаться, когда все буферы пусты, и активизироваться при появлении хотя бы одной записи.

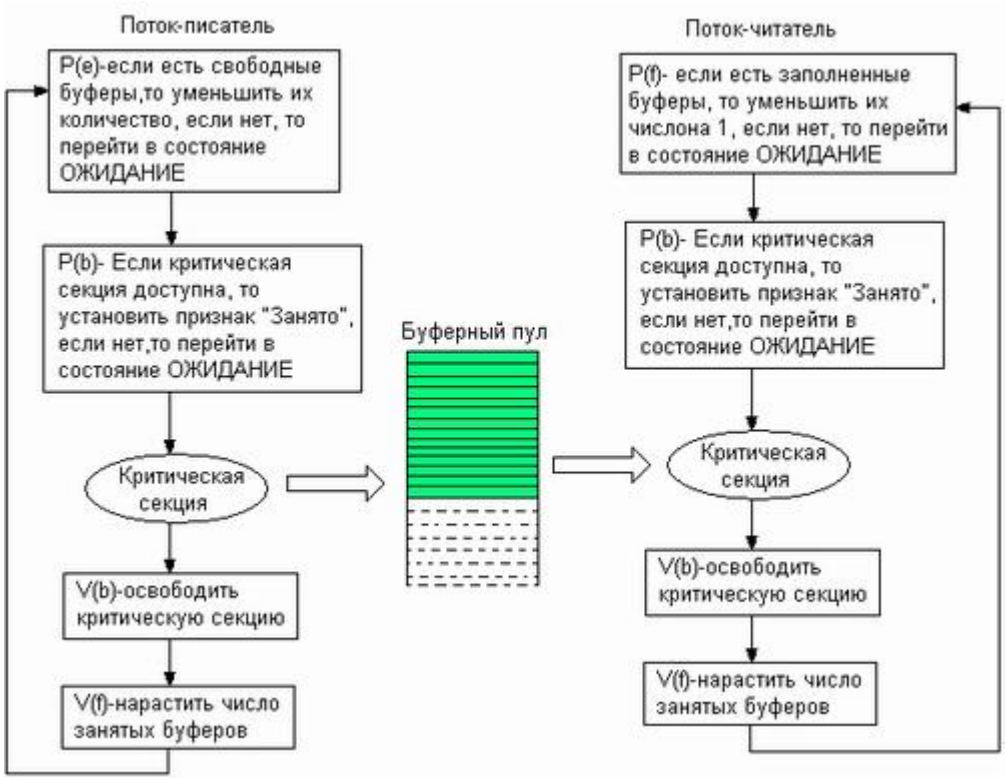
Задача о читателях и писателях



Семафоры: e – число пустых буферов, и f – число заполненных буферов, причем в исходном состоянии e = N, а f = 0.

Задача о читателях и писателях

- Для исключения гонок при работе с разделяемой областью памяти, будем считать, что запись в буфер и считывание из буфера являются критическими секциями, взаимное исключение при доступе к которым будем обеспечивать с помощью мьютекса *b*.



Достоинства и недостатки семафоров

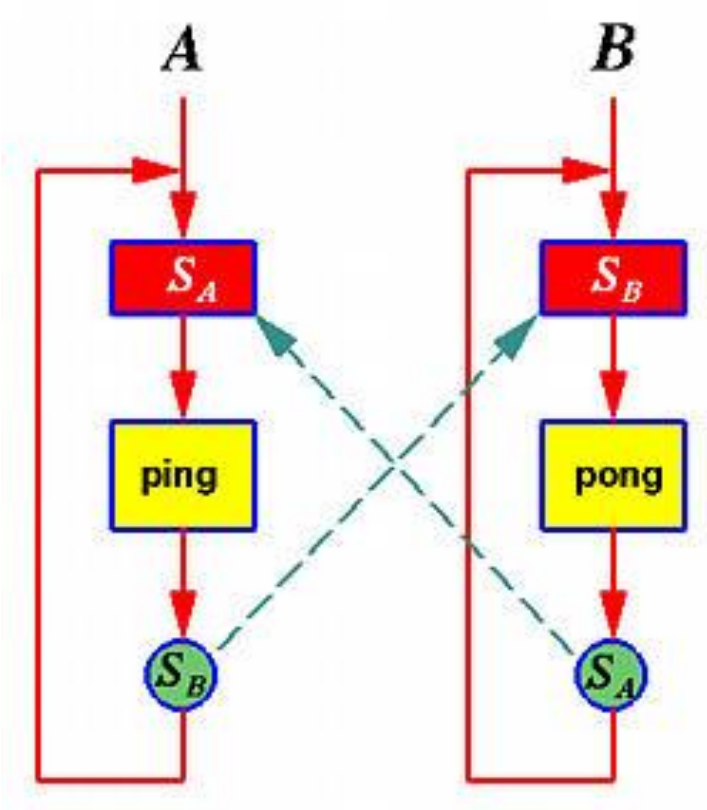
- Достоинства семафоров:
  - простота
  - отсутствие «активного ожидания»
  - независимость от количества процессов
- Недостатки:
  - семафор не «привязывается» к конкретному синхроусловию или критическому ресурсу
  - сложность доказательства корректности программы
  - неправильное использование может привести к нарушению работоспособности системы (возможны тупиковые ситуации)

Взаимная блокировка (тупики)

- Взаимная блокировка, тупик, клинч, дедлок (deadlock) – ситуация, которая может возникнуть в системе, выполненной на безе модели вычислений «сеть процессов», при которой несколько процессов находятся в состоянии бесконечного ожидания ресурсов, захваченных этими процессами.
- В качестве ресурсов могут в том числе выступать семафоры и мьютексы, которые используются для решения задачи взаимного исключения.

Пример взаимной блокировки

- Пусть имеются 2 процесса A и B, которым перед началом работы предоставлены ресурсы S_A и S_B, соответственно.
- В какой-то момент времени процессу A понадобился S_B, а процессу B – S_A, но они их не получат, т.к. удерживаются предыдущими процессами.



Условия возникновения тупиков

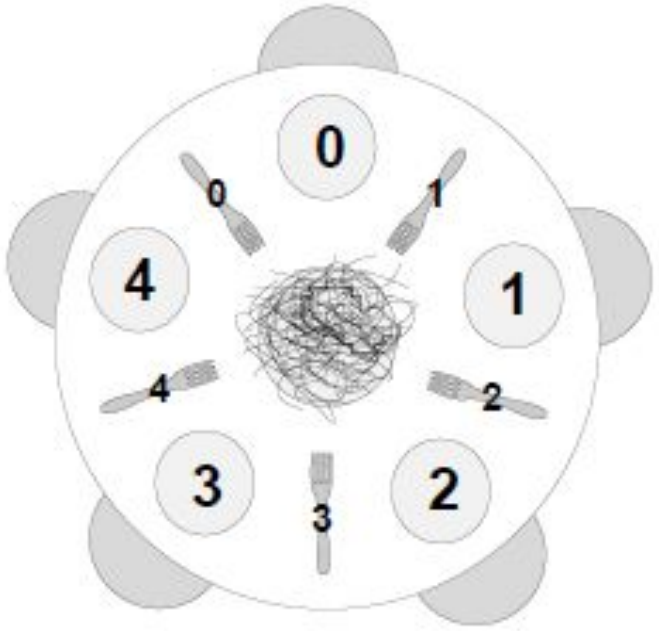
- потоки требуют предоставления им права монопольного управления ресурсами, которые им выделяются (*условие взаимоисключения);*
- потоки удерживают за собой ресурсы, уже выделенные им, ожидая в тоже время выделения дополнительных ресурсов (условие *ожидания ресурсов);*
- ресурсы нельзя отобрать у потоков, удерживающих их, пока эти ресурсы не будут использованы для завершения работы (условие *не перераспределяемости);*
- существует кольцевая цепь потоков, в которой каждый поток удерживает за собой один или более ресурсов, требующихся следующему потоку цепи (условие *кругового ожидания).*

Классические задачи на взаимную блокировку

- Известен ряд классических задач на взаимную блокировку конкурирующих процессов (потоков):
  - Задача об обедающих философах
  - Задача о курильщиках
  - …
  - Задача о Санта-Клаусе
- Решение задач:
  - Требуется разработка алгоритмов синхронизации конкурирующих процессов (потоков).

Задача об обедающих философах

- Пять философов сидят возле круглого стола, в центре которого находится большое блюдо спагетти.
- Философы проводят жизнь, чередуя приемы пищи и размышления
- Чтобы съесть свою порцию, философ должен пользоваться двумя вилками (вариант - палочками).
- К несчастью, философам дали всего пять вилок. Между каждой парой философов лежит одна вилка и философы могут пользоваться только теми вилками, которые лежат рядом с ним (слева и справа).



- **Задача – написать программу, моделирующую поведение философов.**

Решение задачи об обедающих философах (1)

- Решение задачи должно предотвратить наступление тупиковой ситуации, в которой все философы голодны и ни один из них не может взять обе вилки – например, когда каждый из них держит по одной вилке и не хочет отдавать ее.
- Вилки можно представить массивом семафоров, инициализированных значением 1. Взятие вилки имитируется операцией P для соответствующего семафора, а освобождение – операцией V.
- Действия (процессы), выполняемые философами, идентичны друг другу. Например, каждый процесс может сначала взять левую вилку, потому правую. Однако это может привести ко взаимной блокировке процессов. Например, если все философы возьмут свои левые вилки, то они навсегда останутся в ожидании возможности взять правую вилку.

Решение задачи об обедающих философах (2)

- Чтобы избежать взаимной блокировки, достаточно сделать так, чтобы один из философов сперва брал правую вилку, а потом уже левую.

```
sem fork[5] = {1, 1, 1, 1, 1};

process Philosopher[i = 0 to 3] {
  while (true) {
    P(fork[i]); P(fork[i+1]); #взять левую вилку, потом правую
    поесть;
    V(fork[i]); V(fork[i+1]);
    поразмыслить;
  }
}

process Philosopher[4] {
while (true) {
    P(fork[0]); P(fork[4]); #взять правую вилку, потом левую
    поесть;
    V(fork[0]); V(fork[4]);
    поразмыслить;
  }
}
```

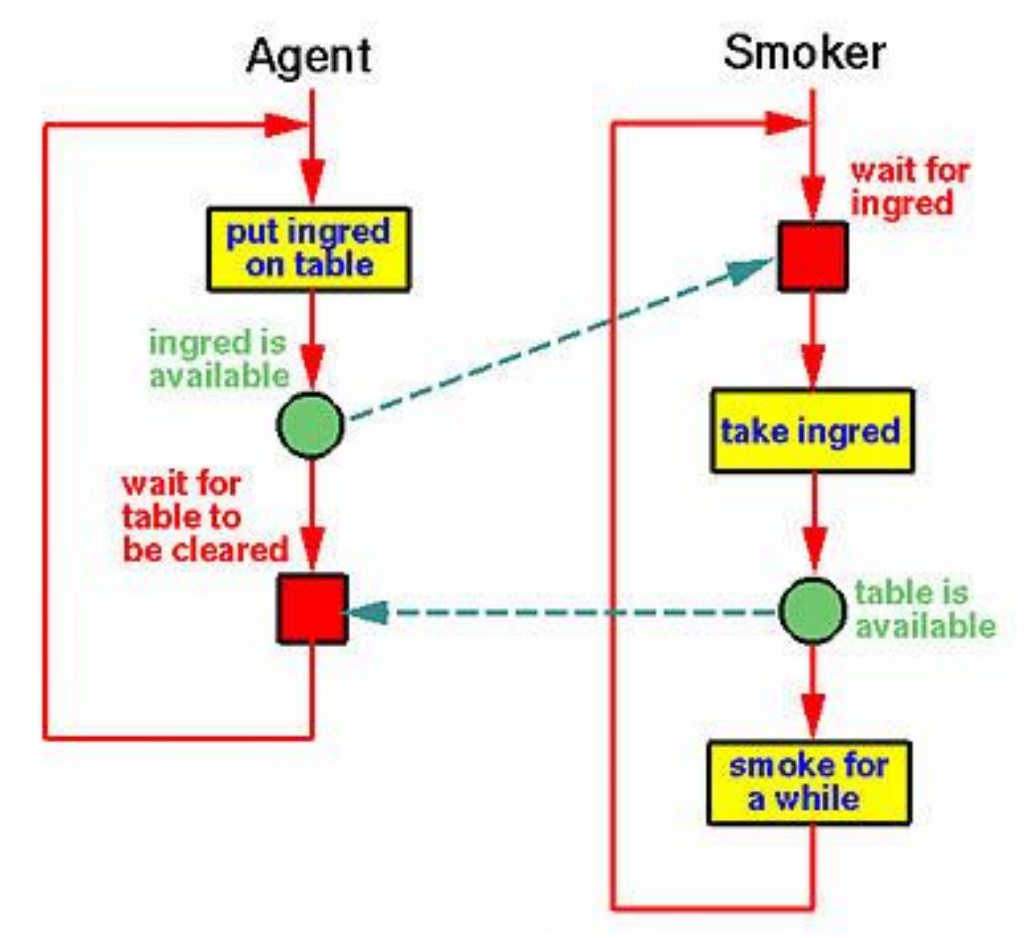
Задача о курильщиках

- Дано:
  - Изначально есть три заядлых курильщика, сидящих за столом.
  - Каждому из них доступно бесконечное количество одного из трёх компонентов: у одного курильщика – табака, у второго – бумаги, у третьего – спичек. Для того чтобы делать и курить сигары, необходимы все три компонента.
  - Также, кроме курильщиков, есть некурящий слуга (дилер), помогающий им делать сигареты.
- Правила:
  - Слуга случайным образом выбирает двух курильщиков, берёт у них по одному компоненту из их запасов и кладёт их на стол.
  - Третий курильщик забирает ингредиенты со стола и использует их для изготовления сигареты, которую он курит некоторое время.
  - В это время слуга, завидев стол пустым, снова выбирает двух курильщиков и кладёт их компоненты на стол.
  - Процесс повторяется бесконечно.

Уточнения к задаче о курильщиках

- Курильщики, по условию задачи, честные: они не прячут компоненты, выданные слугой, – они лишь скручивают сигарету тогда, когда докурят предыдущую. Если слуга кладёт, например, табак и бумагу на стол, пока поставщик спичек курит, то табак и бумага останутся нетронутыми на столе, пока поставщик спичек не докурит сигарету и только затем не возьмёт табак и бумагу.
- Решение задачи не должно быть тривиальным, когда слуга знает какому курильщику какого компонента не хватает и на основании этой информации управляет порядком следования курильщиков.

Иллюстрация взаимодействия курильщика и слуги



«Решение» задачи о курильщиках

- «Слуга» создает четыре семафора и запускает выполнение в цикле трех параллельных потоков обслуживания.

Семафоры:

agentSem = 1
tobacco = 0
paper = 0
match = 0

«Решение» задачи о курильщиках

Курильщик с табаком:

P (paper)
P (match)
V (agentSem)

Курильщик со спичками:

P (tobacco)
P (paper)
V (agentSem)

Курильщик с бумагой:

P (tobacco)
P (match)
V (agentSem)

- Представим ситуацию, что слуга положил на стол табак и бумагу. Курильщик со спичками, ожидающий табак, может приступить к курению. Но курильщик с табаком, ожидающий бумагу, также готов приступить к курению. В итоге первый из них блокируется по бумаге, а второй – по спичкам. Получается тупиковая ситуация !!!

Самостоятельная работа

- Предложить решение задачи о курильщиках.

Задача о Санта-Клаусе

- *“Санта периодически спит, пока не будет разбужен либо всеми своими девятью северными оленями, вернувшимися со свободной выпаски, либо группой из трех эльфов, которых у него всего девять. Если его разбудят олени, он запрягает каждого из них в сани, доставляет вместе с ними игрушки, и в заключение распрягает их (отпуская их на свободную выпаску). Если его разбудят эльфы, он ведет каждую группу в свой кабинет, совещается с ними по поводу разработки новых игрушек, а в заключение выводит каждого из них из кабинета (давая возможность вернуться к работе). Если Санта-Клауса будут одновременно ждать и группа эльфов, и группа оленей, он отдаст приоритет оленям”.*

Идеальный код//Под ред. Э. Орама, Г. Уилсона, Питер, 2011

Правила предотвращения взаимных блокировок

- Прежде чем процесс начнет свою работу, ему должны быть предоставлены все требуемые ресурсы.
- В том случае, если во время работы ему понадобился дополнительный ресурс, ему необходимо возвратить все ранее выделенные ресурсы ОС и затем запросить все требуемые ресурсы с этим дополнительным ресурсом.
- Порядок возврата ресурсов должен быть обратным порядку запроса.