Операционные системы
Автор В.А.Серков

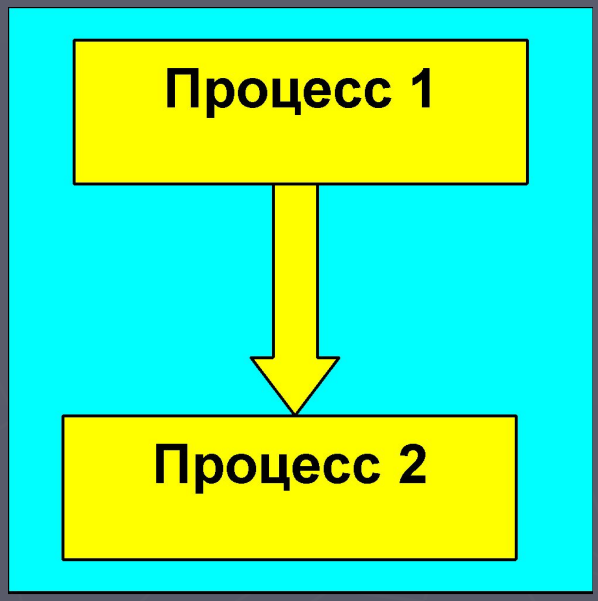
Средства синхронизации и взаимодействия процессов

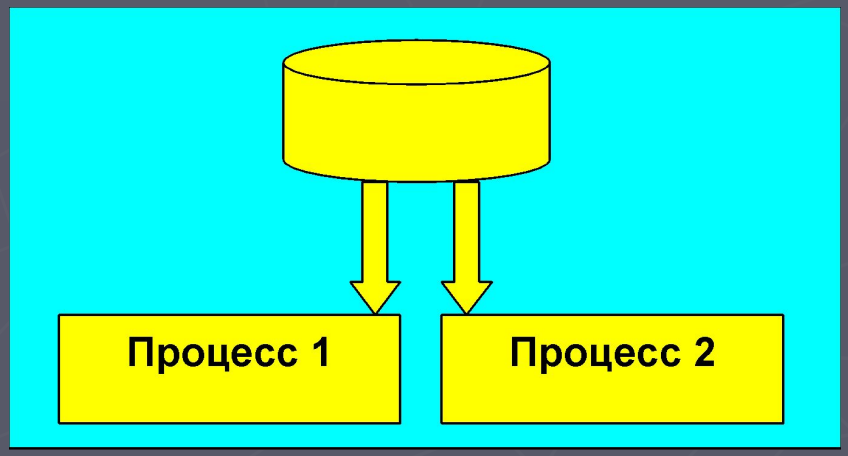
Проблема синхронизации процессов

Процессам часто нужно взаимодействовать друг с другом.

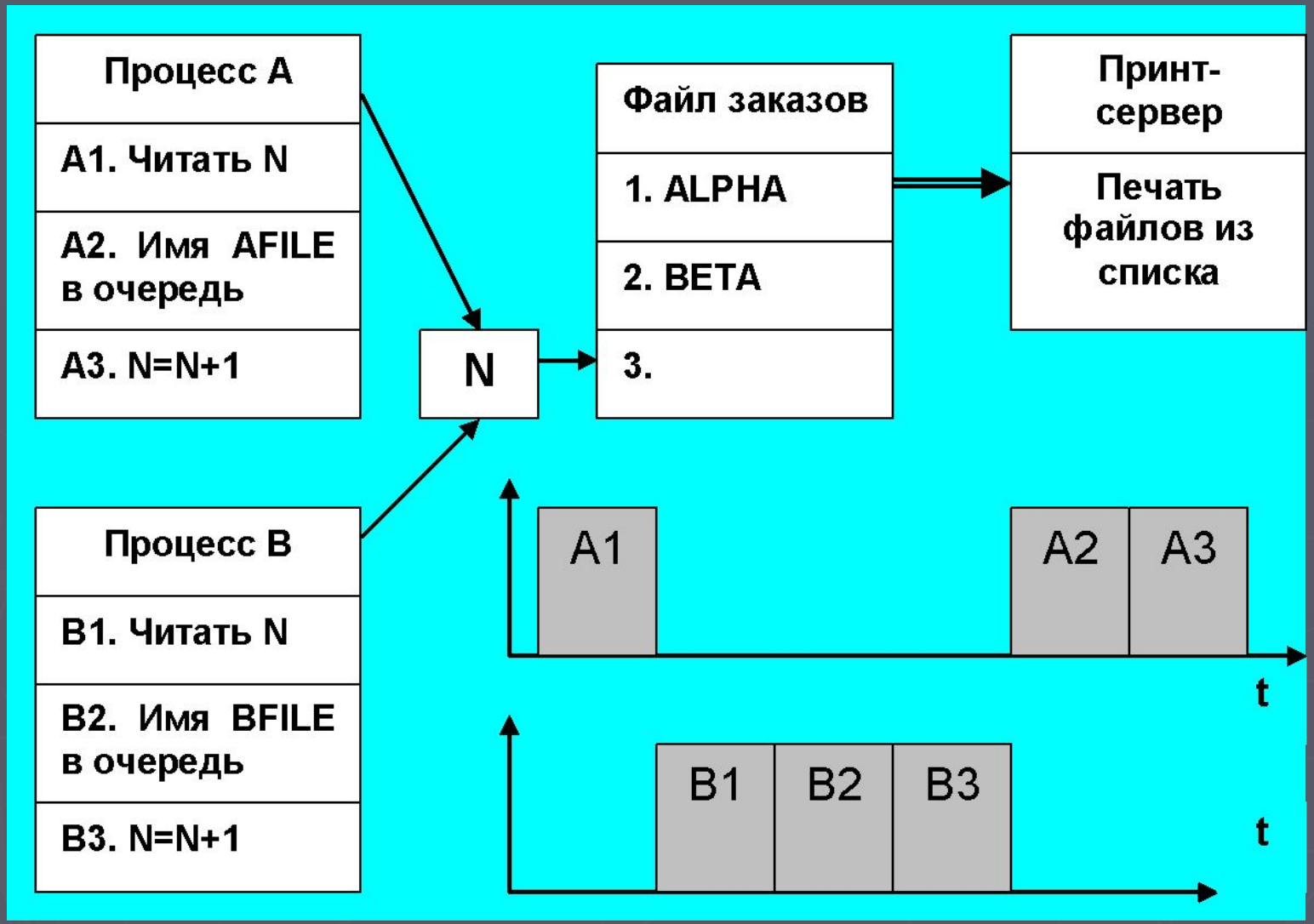
Например, один процесс может передавать данные другому процессу, или несколько процессов могут обрабатывать данные из общего файла.

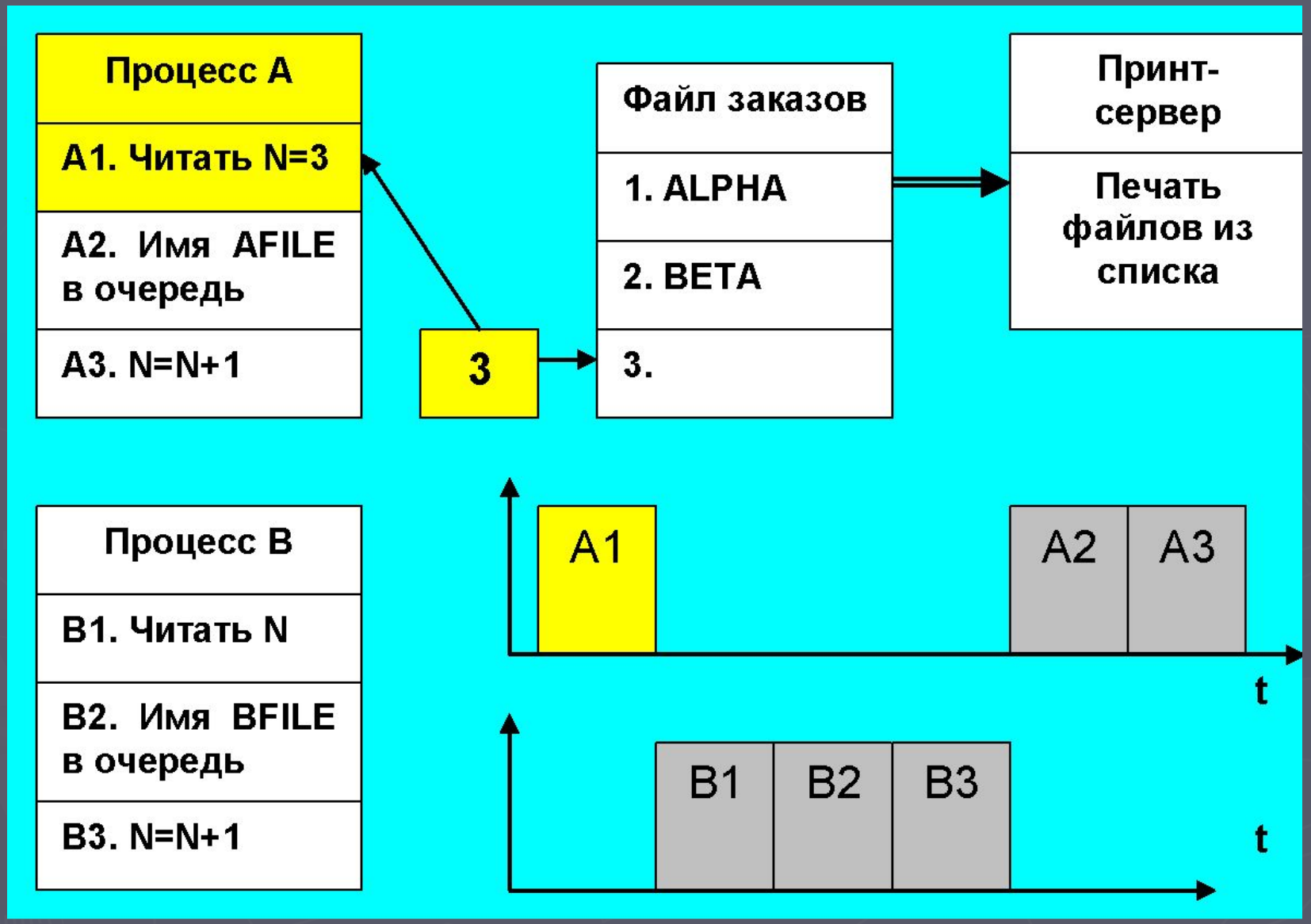
Во всех этих случаях возникает проблема синхронизации процессов, которая может решаться приостановкой и активизацией процессов, организацией очередей, блокированием и освобождением ресурсов.

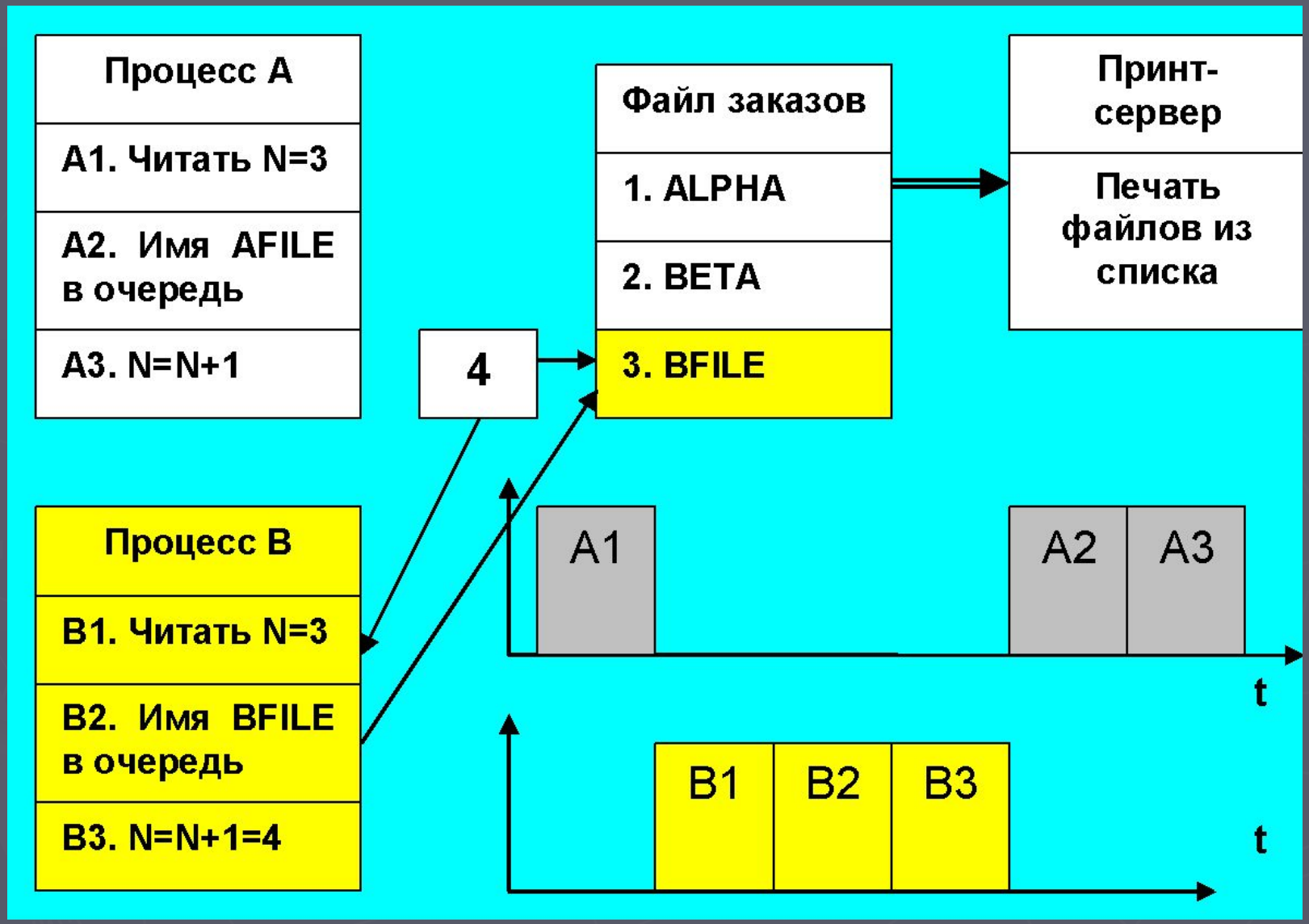


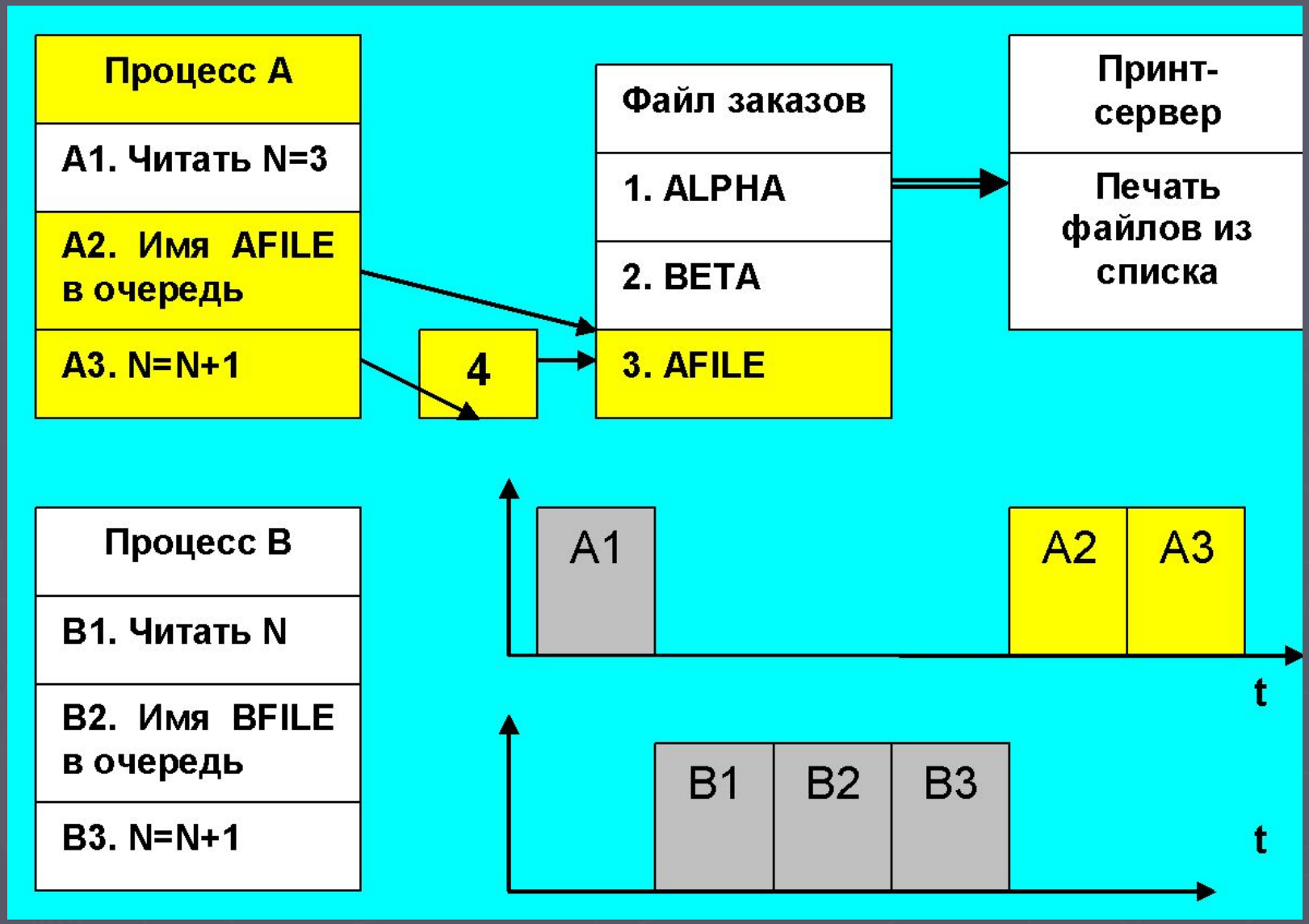


Взаимодействие 2-х процессов




Процесс A
A1. Читать N=3
A2. Имя AFILE в очередь
A3. N=N+1
3
Файл заказов
1. ALPHA
2. BETA
3.
Принт-сервер
Печать файлов из списка
Процесс B
B1. Читать N
B2. Имя BFILE в очередь
B3. N=N+1
A1
A2
A3
t
B1
B2
B3
t


Процесс A
A1. Читать N=3
A2. Имя AFILE в очередь
A3. N=N+1
4
Файл заказов
1. ALPHA
2. BETA
3. BFILE
Принт-сервер
Печать файлов из списка
Процесс B
B1. Читать N=3
B2. Имя BFILE в очередь
B3. N=N+1=4
A1
A2
A3
t
B1
B2
B3
t


Процесс A
A1. Читать N=3
A2. Имя AFILE в очередь
A3. N=N+1
4
Файл заказов
1. ALPHA
2. BETA
3. AFILE
Принт-сервер
Печать файлов из списка
Процесс B
B1. Читать N
B2. Имя BFILE в очередь
B3. N=N+1
A1
A2
A3
t
B1
B2
B3
t

Ситуации, когда два или более процессов обрабатывают разделяемые данные, и конечный результат зависит от соотношения скоростей процессов, называются **гонками**.

Критическая секция

Критическая секция - это часть программы, в которой осуществляется доступ к разделяемым данным.

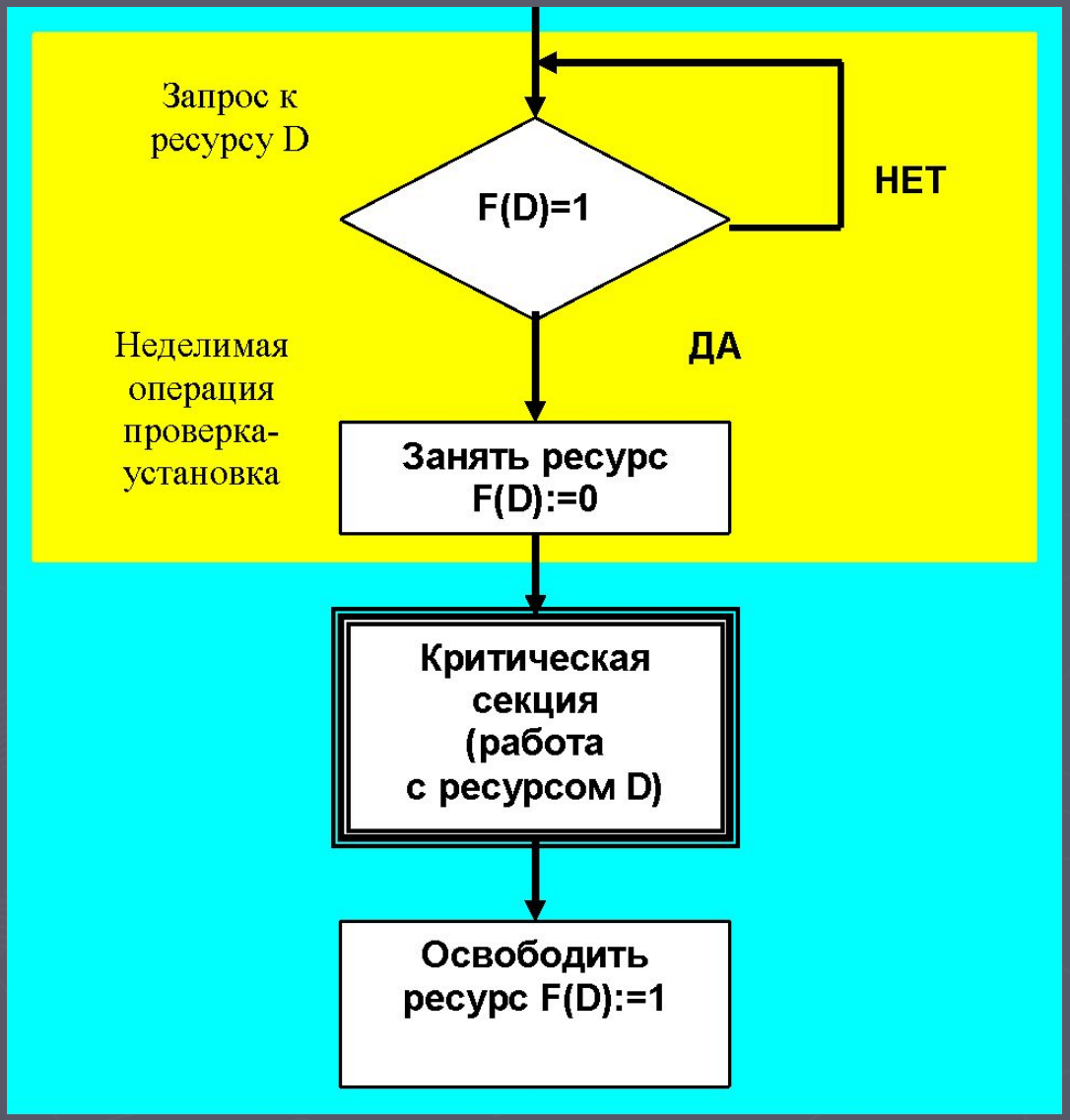
Пример

```
Assign(fl,'D:\Work\Filedan.txt');
Reset(fl);
For i:=1 To 20 Do Writeln(fl,X[i]);
Close(fl);
```

Чтобы исключить эффект гонок по отношению к некоторому ресурсу, необходимо обеспечить, чтобы в каждый момент в критической секции, связанной с этим ресурсом, находился максимум один процесс. Этот прием называют взаимным исключением.

Блокирующая переменная

С каждым разделяемым ресурсом связывается двоичная переменная, которая принимает значение 1, если ресурс свободен (то есть ни один процесс не находится в данный момент в критической секции, связанной с данным ресурсом), и значение 0, если ресурс занят.


Запрос к ресурсу D
F(D)=1
НЕТ
ДА
Неделимая операция проверка-установка
Занять ресурс F(D):=0
Критическая секция (работа с ресурсом D)
Освободить ресурс F(D):=1

Недостаток

В течение времени, когда один процесс находится в критической секции, другой процесс, которому требуется тот же ресурс, будет выполнять рутинные действия по опросу блокирующей переменной, бесполезно тратя процессорное время.

Аппарат событий

Используются системные функции, которые условно назовем **WAIT(x)** и **POST(x)**, где x - идентификатор некоторого события.

Если ресурс занят, то процесс не выполняет циклический опрос, а вызывает системную функцию **WAIT(D)**, здесь **D** обозначает событие, заключающееся в освобождении ресурса **D**.

Функция **WAIT(D)** переводит активный процесс в состояние **ожидание** и делает отметку в его дескрипторе о том, что процесс ожидает события **D**.

Процесс, который в это время использует ресурс **D**, после выхода из критической секции выполняет системную функцию **POST(D)**, в результате чего операционная система просматривает очередь ожидающих процессов и переводит процесс, ожидающий события **D**, в состояние **готовность**.


Запрос к ресурсу D
F(D)=1
НЕТ
WAIT(D)
ДА
Неделимая операция проверка-установка
Занять ресурс F(D):=0
Блокировка процесса до освобождения ресурса D
Критическая секция (работа с ресурсом D)
Освободить ресурс F(D):=1
POST(D)
Снятие блокировки со всех процессов, ожидающих ресурс D

Семафоры

Обобщающее средство синхронизации процессов предложил Эдсгер Дейкстра (1930-2002), который ввел два новых примитива. В абстрактной форме эти примитивы, обозначаемые P и V, оперируют над целыми неотрицательными переменными, называемыми **семафорами**. Пусть S такой семафор. Операции определяются следующим образом:

V(S) : переменная S увеличивается на 1 одним неделимым действием; выборка, инкремент и запоминание не могут быть прерваны, и к S нет доступа другим процессам во время выполнения этой операции.

P(S) : уменьшение S на 1, если это возможно. Если S=0, то невозможно уменьшить S и остаться в области целых неотрицательных значений, в этом случае процесс, вызывающий P-операцию, ждет, пока это уменьшение станет возможным. Успешная проверка и уменьшение также является неделимой операцией.

```
int e = 255, f = 0, b = 1;
void Writer ()
{
        while(1)
        {
                PrepareNextRecord(); /* подготовка новой записи */
                P(e);    /*Уменьшить число свободных буферов, если они есть */
                         /*в противном случае - ждать, пока они освободятся */
                P(b);          /* Вход в критическую секцию */
                AddToBuffer(); /* Добавить новую запись в буфер */
                V(b);          /* Выход из критической секции */
                V(f);          /* Увеличить число занятых буферов */
        }
}
void Reader ()
{
        while(1)
                {
                        P(f);     /* Уменьшить число занятых буферов, если они есть */
                                  /* в противном случае ждать, пока они появятся */
                        P(b); /* Вход в критическую секцию */
                        GetFromBuffer(); /*Взять запись из буфера */
                        V(b);     /* Выход из критической секции*/
                        V(e); /* Увеличить число свободных буферов*/
                        ProcessRecord(); /*Обработать запись*/
                }
}
```

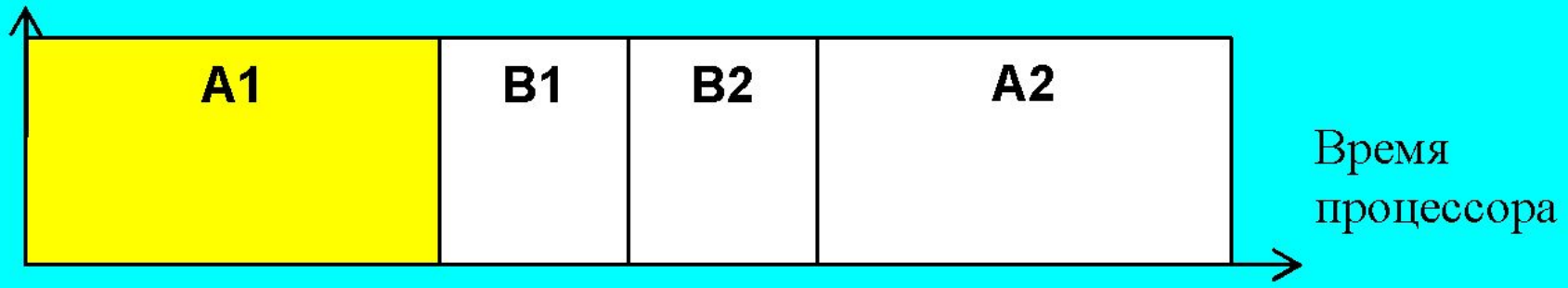
Тупики

Приведенный выше пример поможет нам проиллюстрировать еще одну проблему синхронизации - **взаимные блокировки**, называемые также **дедлоками, клинчами** или **тупиками.** Если переставить местами операции P(e) и P(b) в программе «писателе», то при некотором стечении обстоятельств эти два процесса могут взаимно заблокировать друг друга. Действительно, пусть «писатель» первым войдет в критическую секцию и обнаружит отсутствие свободных буферов.

Он начнет ждать, когда «читатель» возьмет очередную запись из буфера, но «читатель» не сможет этого сделать, так как для этого необходимо войти в критическую секцию, вход в которую заблокирован процессом «писателем».

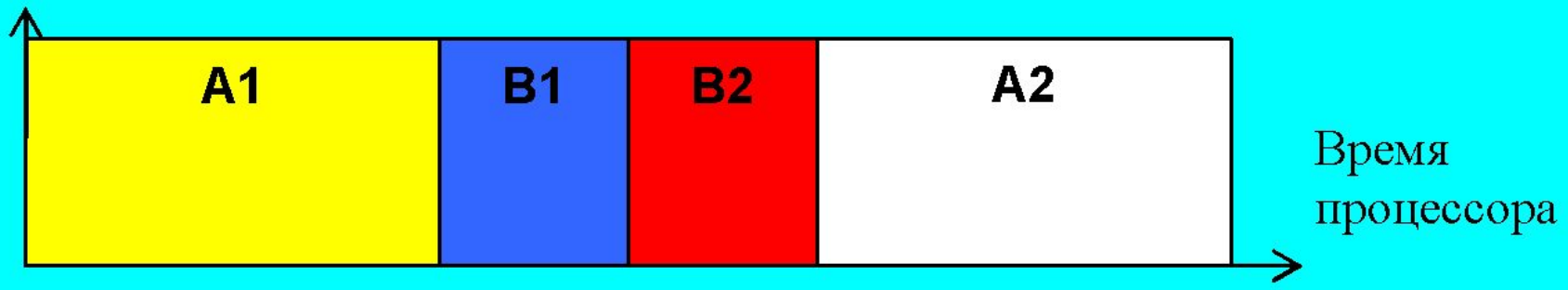
A1. Занять принтер
A2. Занять диск
A3. Освоб. принтер
A4. Освоб. диск

B1. Занять диск
B2. Занять принтер
B3. Освоб. диск
B4. Освоб. принтер



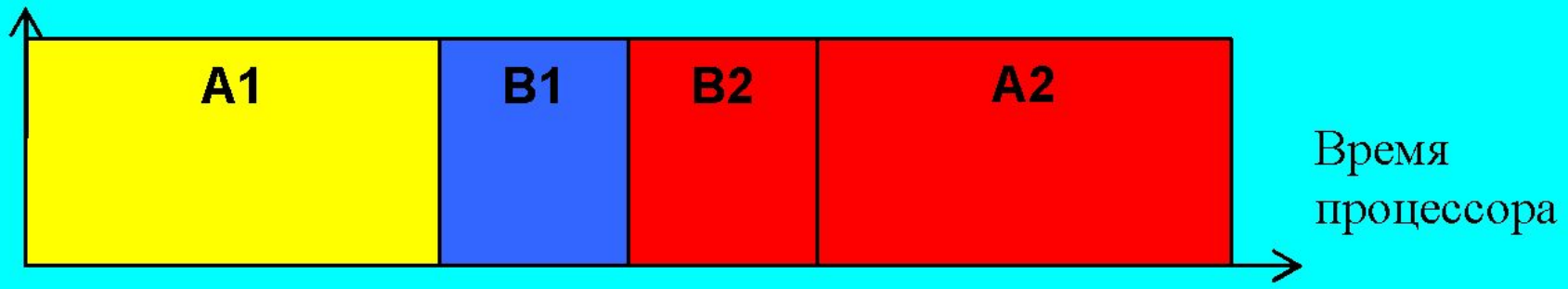
A1. Занять принтер
A2. Занять диск
A3. Освоб. принтер
A4. Освоб. диск

B1. Занять диск
B2. Занять принтер
B3. Освоб. диск
B4. Освоб. принтер



A1. Занять принтер
A2. Занять диск
A3. Освоб. принтер
A4. Освоб. диск

B1. Занять диск
B2. Занять принтер
B3. Освоб. диск
B4. Освоб. принтер



В рассмотренных примерах тупик был образован двумя процессами, но взаимно блокировать друг друга могут и большее число процессов.

Проблема тупиков включает в себя следующие задачи:

- предотвращение тупиков;
- распознавание тупиков;
- восстановление системы после тупиков.

Тупики могут быть предотвращены на стадии написания программ, то есть программы должны быть написаны таким образом, чтобы тупик не мог возникнуть ни при каком соотношении взаимных скоростей процессов. Так, если бы в предыдущем примере процесс А и процесс В запрашивали ресурсы в одинаковой последовательности, то тупик был бы в принципе невозможен.

Второй подход к предотвращению тупиков называется динамическим и заключается в использовании определенных правил при назначении ресурсов процессам, например, ресурсы могут выделяться в определенной последовательности, общей для всех процессов.

Восстановление системы. Если же тупиковая ситуация возникла, то не обязательно снимать с выполнения все заблокированные процессы. Можно снять только часть из них, при этом освобождаются ресурсы, ожидаемые остальными процессами, можно вернуть некоторые процессы в область свопинга, можно совершить «откат» некоторых процессов до так называемой контрольной точки, в которой запоминается вся информация, необходимая для восстановления выполнения программы с данного места. Контрольные точки расставляются в программе в местах, после которых возможно возникновение тупика.