Каналы (трубопроводы)

IPC: PIPE

безымянные и именованные

Это произведение доступно по лицензии
Creative Commons “Attribution-ShareAlike” (“Атрибуция — На тех же условиях”) 3.0 Непортированная.
http://creativecommons.org/licenses/by-sa/3.0/deed.ru

Известные на данный момент средства IPC

- Сигналы — не предназначены для передачи больших объёмов данных. Это средство асинхронного уведомления.
- Файловые операции — простые (read, write), но требуют участия файловой системы. Это не всегда возможно и очень расточительно — записывать на внешний носитель только для того, что бы передать в другой процесс.

Требования к средству ICP

Требуемое средство передачи данных должно быть:

- Простым, в идеале не отличаться от доступа к файлам.
- Файловая система не должна участвовать в передаче данных.
- Должен быть способ, предотвращающий переполнение и потерю данных.
- Взаимодействовать должны как родственные процессы, так и произвольные. Должен быть контроль, какие процессы допущены к обмену.

Пример из интерпретатора bash

В интерпретаторе bash мы уже использовали (на верхнем уровне) такое средство IPC, как трубопроводы:

```
ls -l | sort -u | head -15
```

В DOS+ трубопроводы тоже есть, то реализованы с использованием временных файлов (трижды+ плохо).

В POSIX-совместимых операционных системах для реализации этого механизма используются **безымянные каналы**.

Системный вызов pipe

Создать безымянный канал можно системным вызовом **pipe** :

```
#include <unistd.h>

int pipe(int pipefd[2]);
```

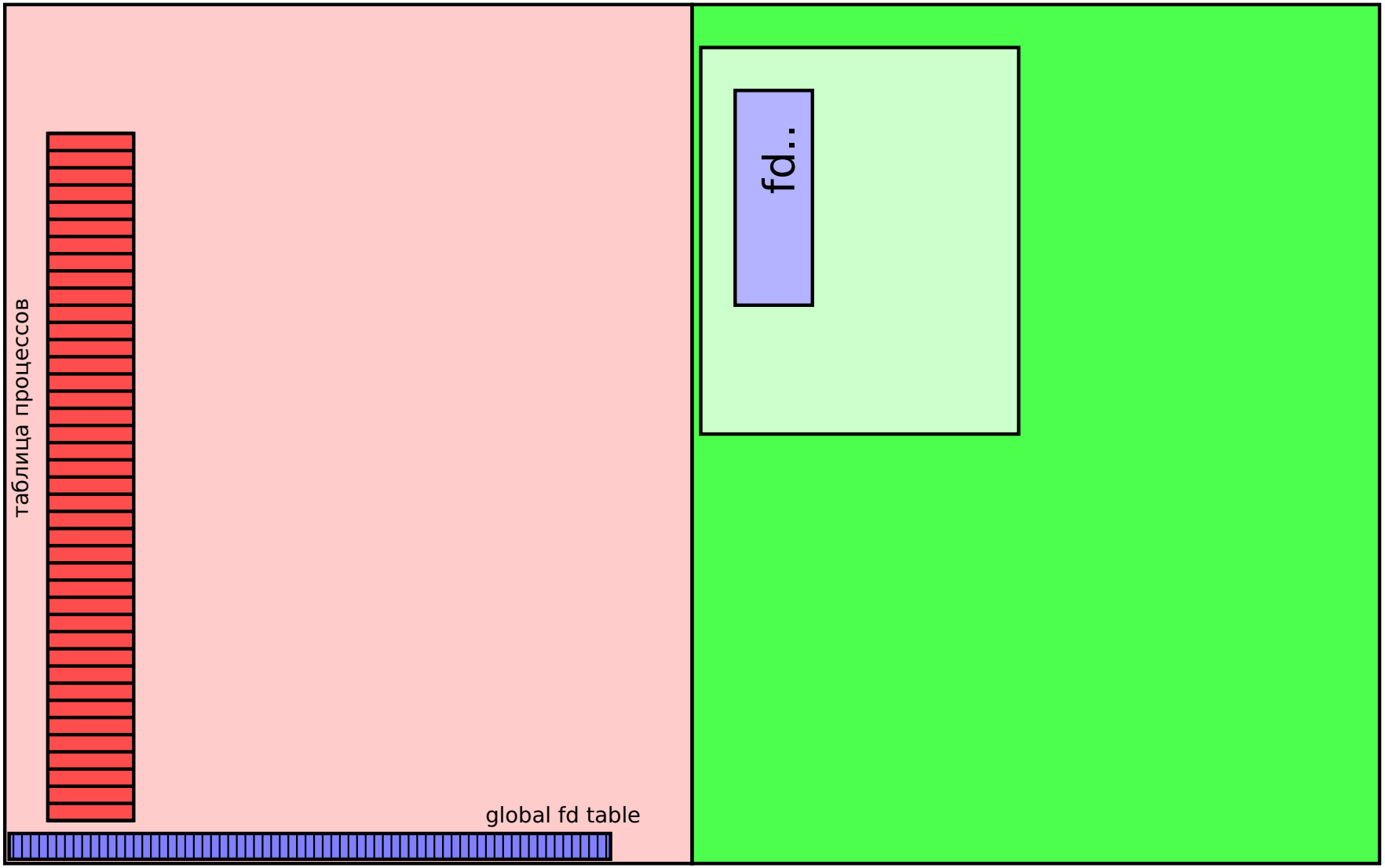
Аргумент pipefd — указатель на массив из двух целых (int), в который в случае успеха будут записаны **файловые дескрипторы** , соответствующие двум концам созданного безымянного канала.

pipefd[0] — дескриптор для чтения,
pipefd[1] — дескриптор для записи.

В памяти ядра будет создан буфер, в котором данные хранятся до того, как их прочитают.
При ошибке возвращает -1.

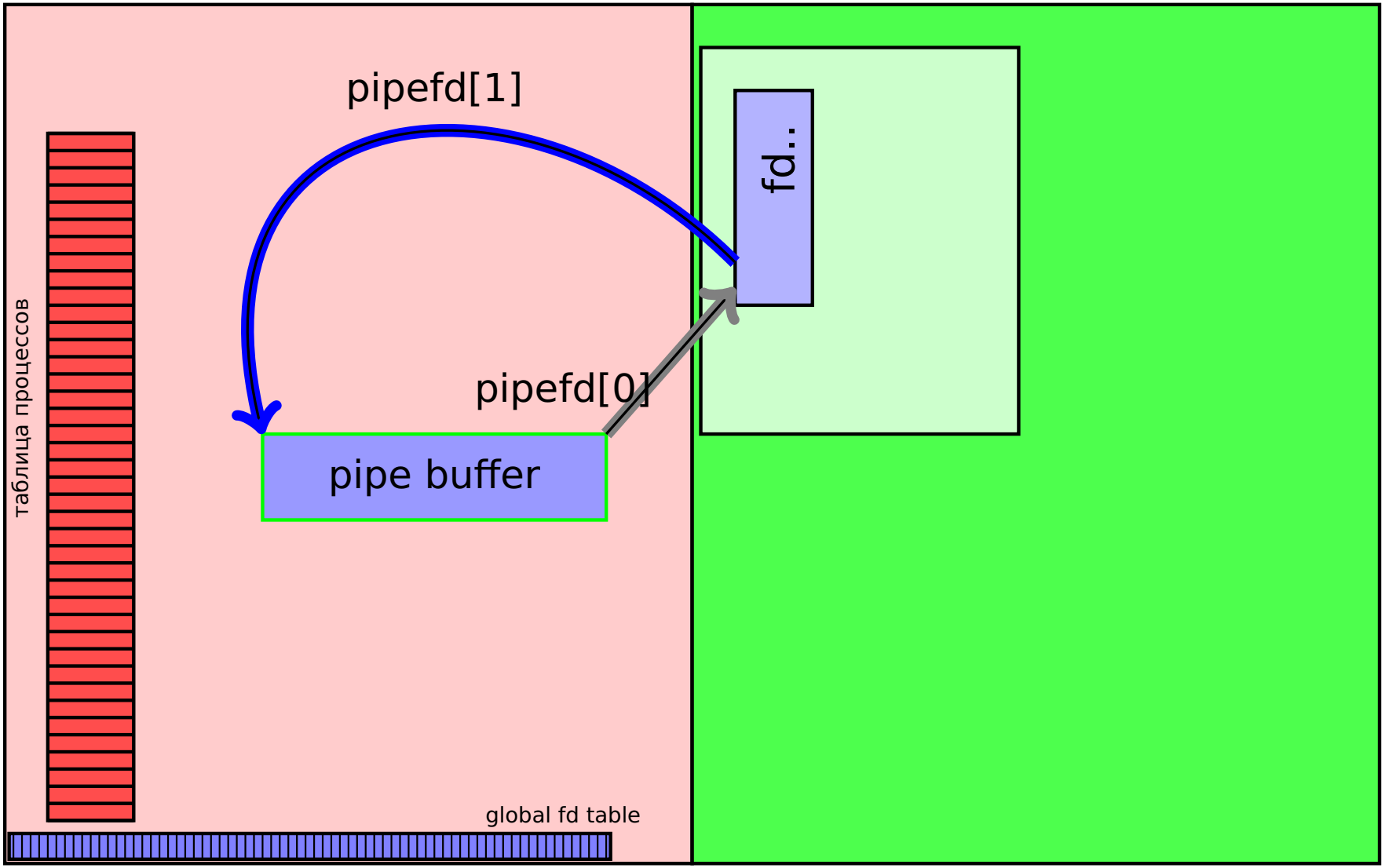
Системный вызов pipe

Процесс до вызова pipe



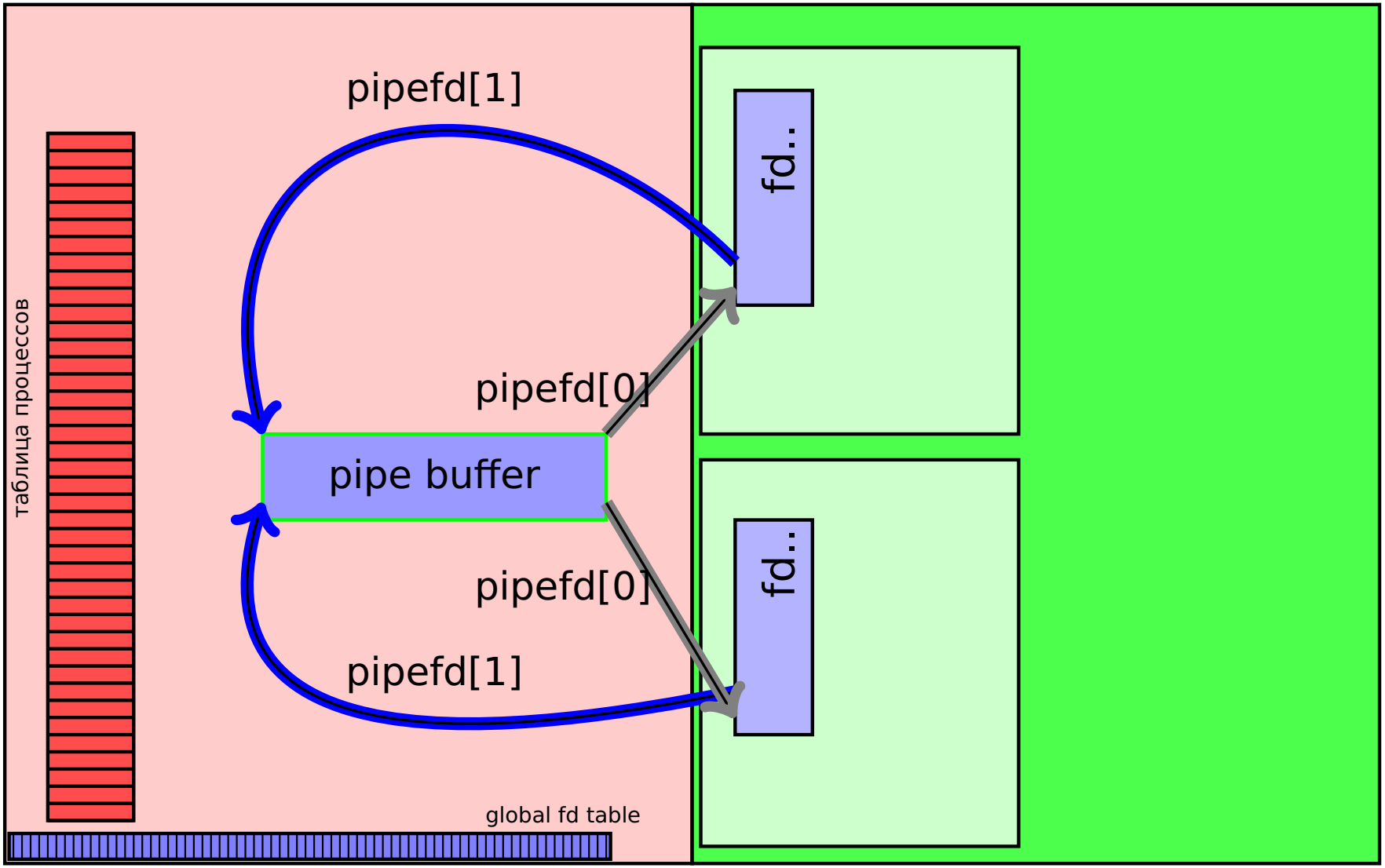
Системный вызов pipe

Процесс после вызова pipe



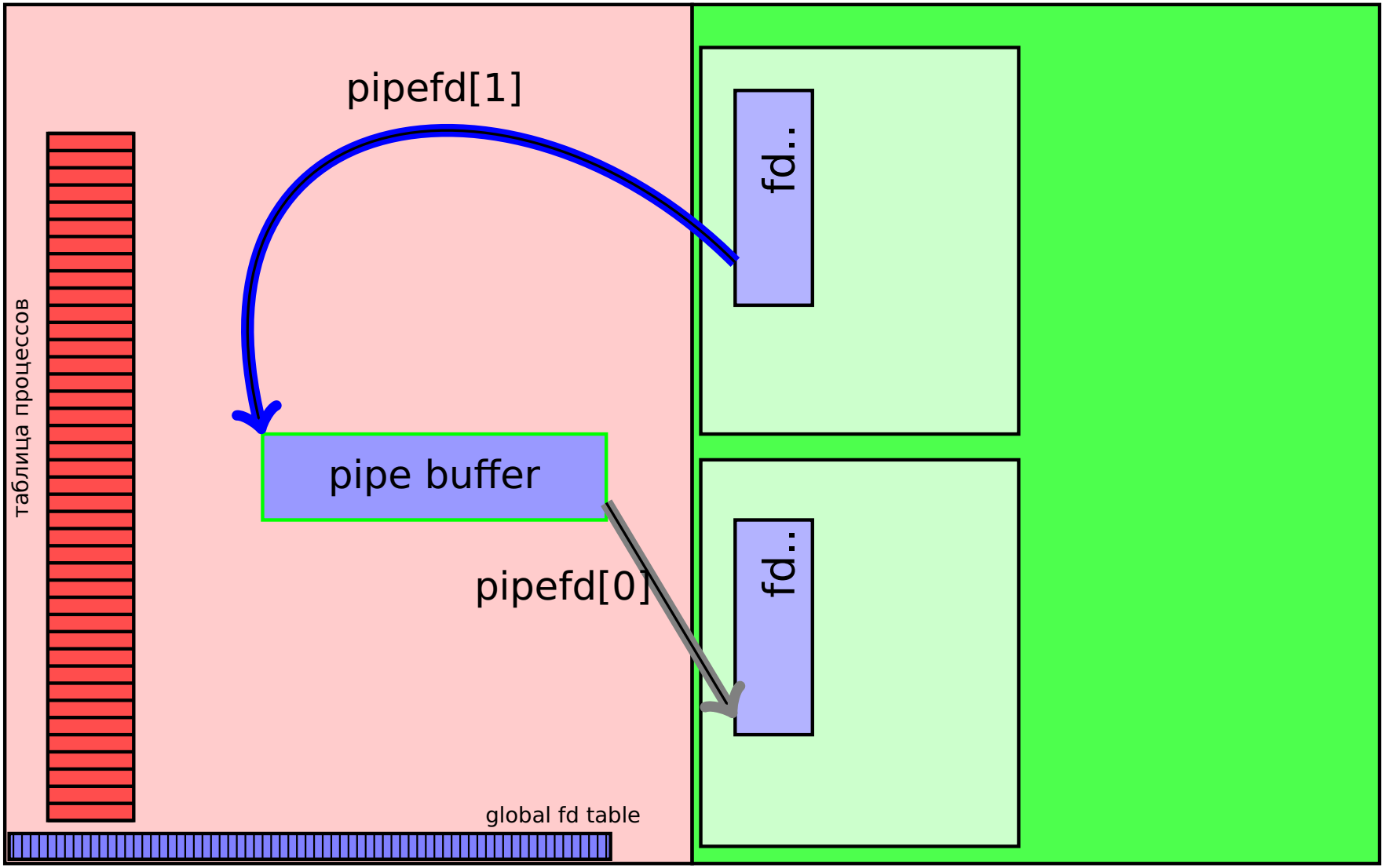
Системный вызов pipe

Пара процессов после вызова pipe и fork



Системный вызов pipe

Закрываем ненужные дескрипторы



Синхронизация процессов использующих pipe

Состояние 1 — блокировка “писателя”

Если процесс, пишущий в канал, поставляет данные быстрее, чем из успевает прочесть другой процесс, то буфер заполняется полностью.

В этом случае ядро блокирует пишущий процесс до тех пор, пока читающий не прочитает достаточное количество байт.

Таким образом, “писатель” будет блокироваться чаще, и скорость его будет уменьшена. Освободившиеся кванты времени достанутся другим процессам, в том числе “читателю”.

Синхронизация процессов использующих pipe

Состояние 2 — блокировка “читателя”

Если процесс успевает читать быстрее, чем поступают данные, то буфер опустошается.

В этом случае блокируется “читатель”, и синхронизация скоростей происходит аналогично.

P.S. Существуют неблокирующие чтение и запись.
`(fcntl(fd[0], F_SETFL, O_NONBLOCK))`

Разрыв трубопровода со стороны “писателя”

Если “писатель” закрывает свой конец трубопровода, то потерь данных нет.

В этом случае при чтении “читатель” просто получит состояние “конец файла (EOF)”.

Важно: у “читателя” не должно быть открытого файлового дескриптора для конца трубопровода, предназначенного для записи.

Разрыв трубопровода со стороны “читателя”

Если читатель закрывает свой конец трубопровода, то данные будут потеряны.

Об этой ситуации операционная система извещает “писателя” сигналом SIGPIPE — разрыв трубопровода. Реакция по умолчанию на этот сигнал — завершение.

Копии файловых дескрипторов

“Трюк с подменой”

Системные вызовы dup, dup2 позволяют получить копию ф.д. — другой файловый дескриптор, привязанный к тому же файловому объекту.

```
#include <unistd.h>
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

Оба при успехе возвращают новый файловый дескриптор, привязанный к тому же файлу, что и oldfd. dup — использует первый свободный, dup2 — заданный (newfd). При необходимости newfd предварительно закрывается. Часто используется при подмене файловых дескрипторов.

```
pipe( pipefd );
// ....
dup2( pipefd[0], 0 );
close( pipefd[0] );
```

Именованные каналы

Для связи процессов, не состоящих в родственных отношениях, используются именованные каналы. Для этого в операционной системе создаётся специальный файла (fifo), предназначенный для установления соединения

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *pathname, mode_t mode);
```

pathname — путь к создаваемому файлу, mode — права доступа (аналогично open).
При успехе возвращает 0 и создаёт специальный файл с указанным именем. При ошибке — -1.
Как только такой специальный файл откроют и для чтения, и для записи, ОС организует буфер в памяти.
Дальше — полностью аналогично pipe.

Выводы

Трубопроводы позволяют:

- Обмениваться данными родственным процессам (pipe) или произвольным процессам (mkfifo).
- Интерфейс практически полностью идентичен интерфейсу работы с файлами (open, read, write, close). Некоторые операции недоступны при работе с каналами (например, lseek).
- Файловая система в передаче данных не используется. Специальный файла типа fifo используется именованным каналом только для установления канала.
- Обеспечивается синхронизация скорости “читателя” и “писателя”.
- Правильная реакция на разрыв трубопровода с обеих сторон.