Виртуальная память

АКОС, МФТИ

17 октября, 2024

Как разные процессы используют одни и те же адреса?

В былые времена…

16-битные компьютеры на DOS использовали **сегменты**:

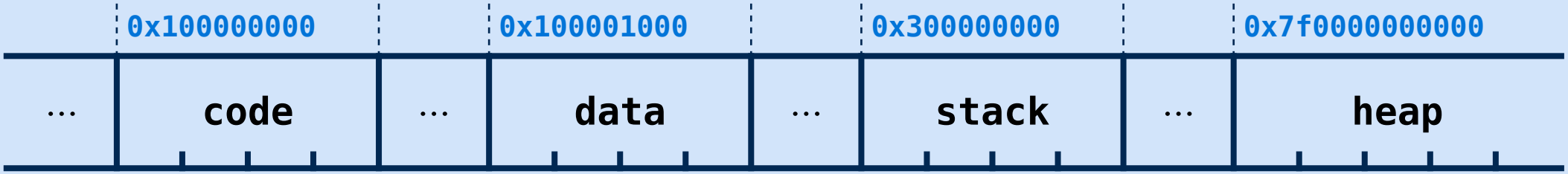
- **`CS`** - сегмент кода;
- **`DS`** - сегмент данных;
- **`SS`** - сегмент стека;
- **`ES`** - пользовательский сегмент;

Сегмент в x86 - постоянное слагаемое к адресу.

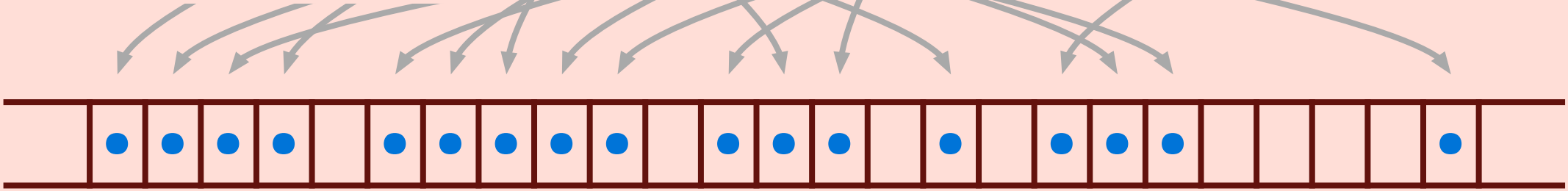
Разным процессам выдавались разные сегменты.

Современные компьютеры используют виртуальную адресацию.

Виртуальные адреса



Физические адреса



Адреса, которые используются процессами, называются **вирутуальными**.

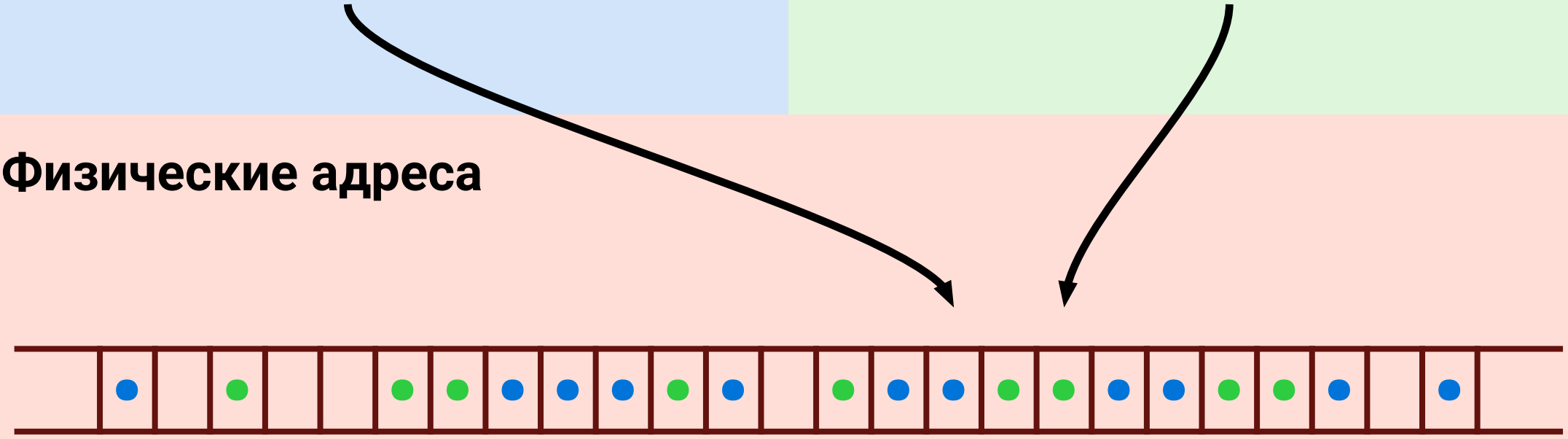
Физический адрес – уже реальный индекс байта в оперативной памяти.

Процесс 1

`0x100001020`

Процесс 2

`0x100001020`



Одинаковые виртуальные адреса могут указывать на разные физические.

Как связать виртуальный адрес с физическим?

```
mmap(void *start, size_t len, int prot, int flags, int fd, off_t off)
```

`void* start` – Виртуальный адрес или `NULL` (кратно `PAGE_SIZE`);

`size_t len` – Сколько байт связать (кратно `PAGE_SIZE`);

`int prot` – Права доступа к памяти;

`int flags` – Нужно установить в `MAP_ANONYMOUS` ;

`int fd` – Нужно установить в `-1` ;

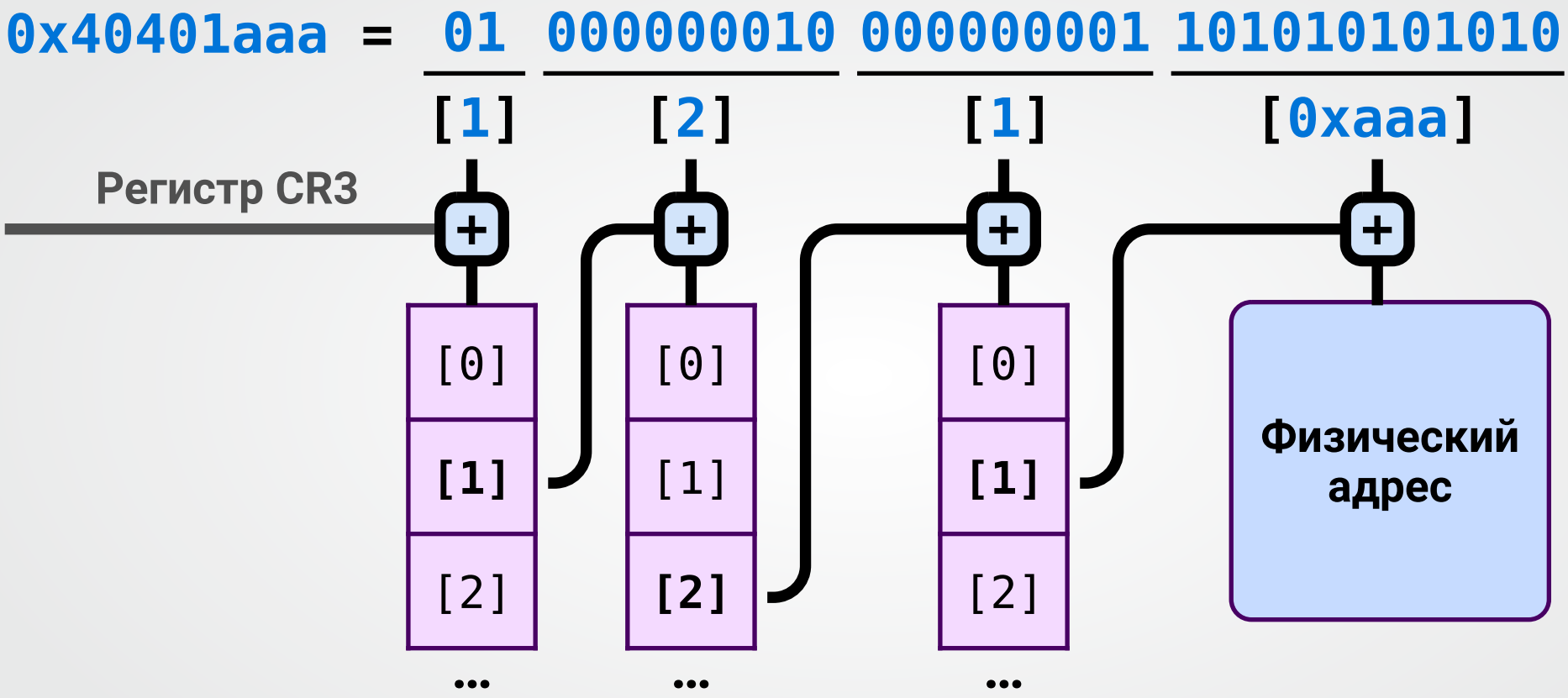
`off_t off` – Может быть любым.

- Ядро выберет свободный физический адрес и свяжет его с виртуальным. Самому выбрать физический адрес нельзя.
- `malloc()` и `calloc()` делают то же самое, когда им нужны новые страницы памяти.
- Когда страницы больше не нужны, используйте `munmap()`.

Таблицы страниц

- **Хранят соответствие** виртуальных адресов физическим и **права доступа** к ним.
- Модифицируются **только ядром**. Конвертацией адресов занимается **процессор**.
- `mmap()` просит ядро создать новую запись в таблице страниц.
- Связаны **иерархически**, как директории и файлы.
- Минимально адресует **страницу памяти** (обычно 4 КБ).
- Таблицу страниц любого процесса можно увидеть в `/proc/PID/maps`.

Конвертация виртуального адреса в x86 (PAE)



- Каждый уровень хранит кучу флажков. Например о том, что адрес действителен;
- Если нужно замаппить много памяти, можно опустить третий уровень;
- Частоиспользуемые адреса кешируются в TLB.

Что произойдет, если записи в таблице не существует?

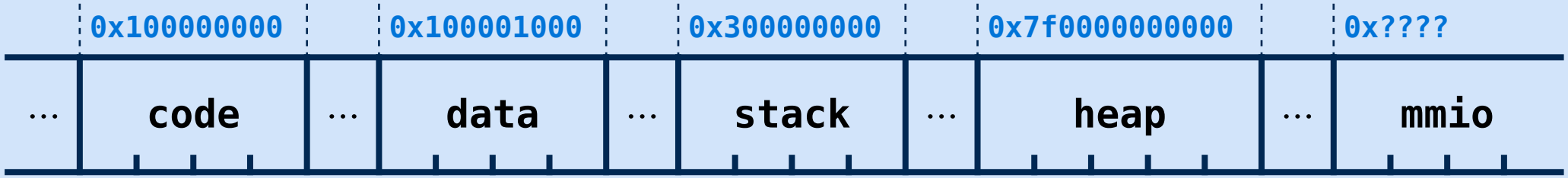
Page Fault

Исключение процессора, возникающее при ошибке доступа к виртуальному адресу.

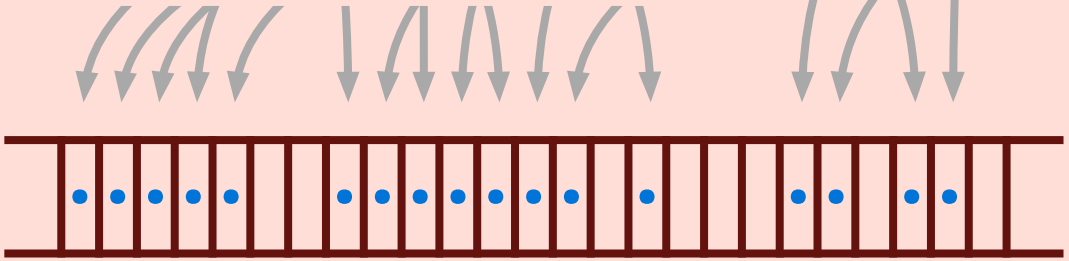
Возникает, если:

- Ваша программа сделала `*((int*)0) = 0` . Тогда ядро отправит вам `SIGSEGV`
- Ядро **выгрузило** эту страницу из памяти **в своп**. Тогда ядро загрузит страницу обратно и вы ничего не заметите;
- Ядро **лениво скопировало** эту страницу, а вы собрались её изменять. Тогда ядро создаст новую страницу и скопирует туда данные. (Copy-on-write).
- Вы **переполнили стек**, замапленный с `MAP_GROWSDOWN`. Тогда ядро незаметно замаппит вам ещё памяти.
- …

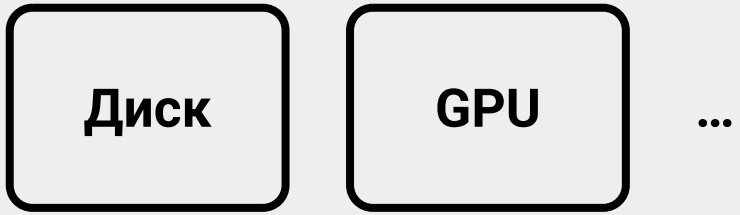
Виртуальные адреса



Физические адреса



Устройства



Виртуальные адреса можно связать не только с памятью, но и с устройствами.

Как связать виртуальный адрес с файлом на диске?

```
mmap(void *start, size_t len, int prot, int flags, int fd, off_t off)
```

`void* start` – Виртуальный адрес или `NULL` (кратно `PAGE_SIZE`);

`size_t len` – Сколько байт связать (кратно `PAGE_SIZE`);

`int prot` – Права доступа к памяти;

`int flags` – Флаги;

`int fd` – Дескриптор открытого файла;

`off_t off` – Смещение в файле.

В зависимости от флагов, запись в эти адреса будет влиять на файл на диске. При этом оперативная память может вообще не использоваться.

Флаги `mmap()`:

`int flags =`

`|=` `MAP_SHARED` – изменения будут видны другим процессам;

`|=` `MAP_PRIVATE` – изменения будут видны только текущему процессу;

`|=` `MAP_ANONYMOUS` – не связывать с файлом;

`|=` `MAP_FIXED` – использовать указанный адрес;

`|=` `MAP_GROWSDOWN` – выделить стек, растущий вниз;

`int prot`

`=` `PROT_READ` – разрешить чтение выделенных адресов;

`=` `PROT_WRITE` – разрешить запись выделенных адресов;

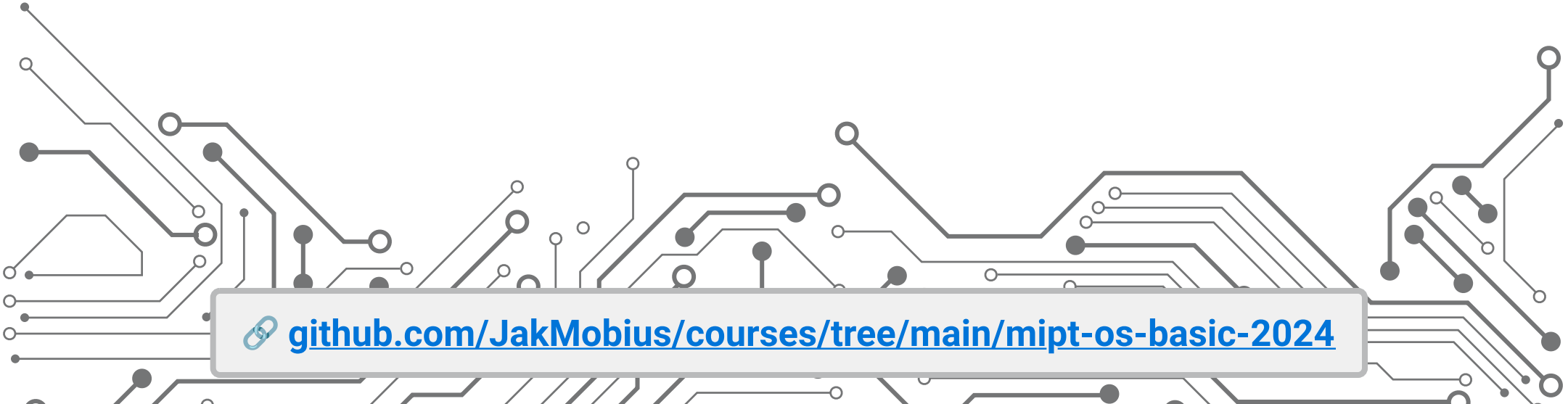
`=` `PROT_EXEC` – разрешить исполнение выделенных адресов;

`=` `PROT_NONE` – запретить всё.

Бонус

Используем видеопамять как оперативную память

Спасибо за внимание!



🔗 github.com/JakMobius/courses/tree/main/mipt-os-basic-2024