Управление памятью

Оперативная память и проц. время — важнейшие ресурсы системы;
Вычисл. системы совершенствуются с т.з. быстродействия, объёмов памяти, внеш. устр-в и возможности писать всё более совершенные приложения, решающие больше кол-во задач и сопровождаемые GUI;
Это приводит к изменениям требований к ПО;
⇒ необходимо изучать оперативную память (как ресурс и понимать, как и почему именно так система работает с этим ресурсом;

Фрагментация

Большие машины практически не выключаются

Вопрос восст. из тупика забыли рассмотреть ранее. Там также самое простое решение — перезагрузка системы;

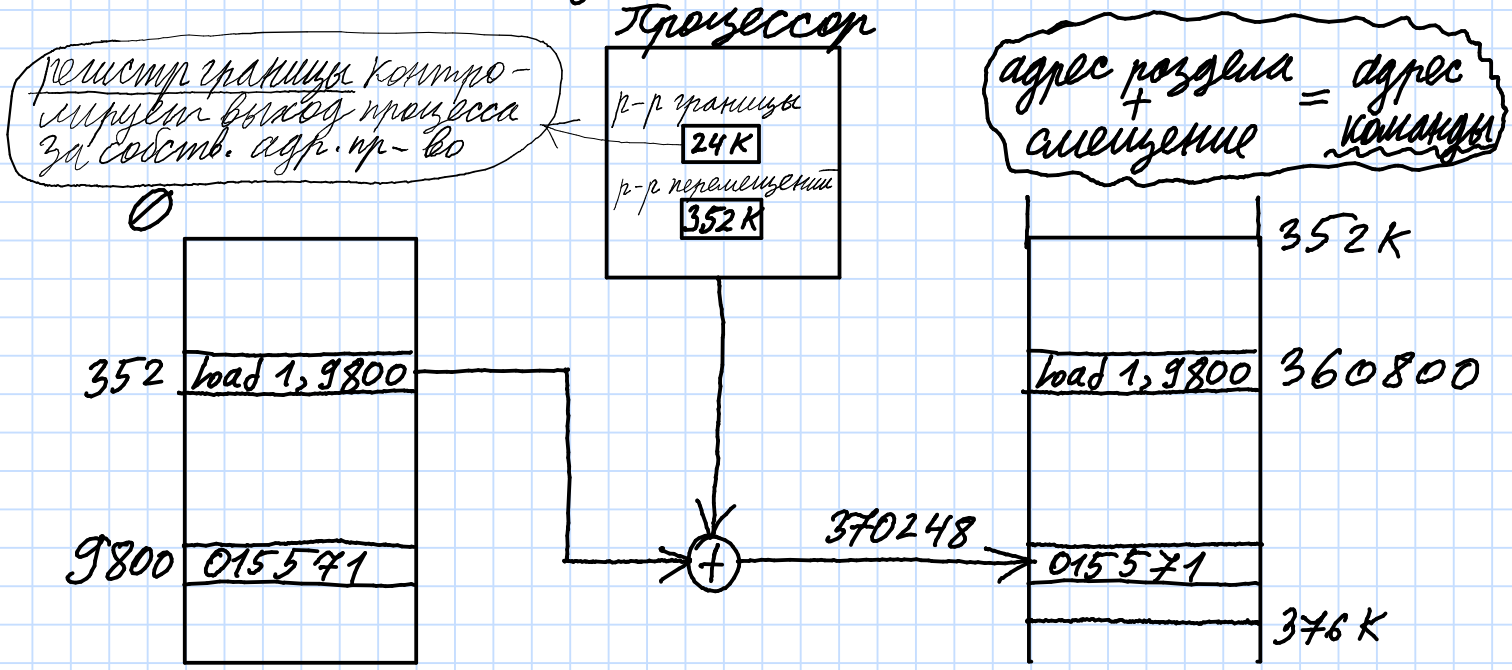
До 30% памяти не м.б. использовано из-за фрагментации — утечка памяти в явном виде;

Перемещение разделов потребовало преобразования адресов и включения в процессор регистра перемещений;

Такое преобразование выполняется на каждой команде, а то и несколько раз (особенно если в команде есть косвенная адресация)

Начальный адрес никак не связан расположением программы ⇒ логическое адр. пр-во ⟹
⟹ в программах только смещения;

Загрузчик (ПО) загружает программу в память, т.е. она имеет адрес в памяти:



Преобразование осталось: к нач. адресу прибавляется смещение;
Но это поменяло всё сист. программирование;

Системное программирование

Операционные системы
Утилиты ОС

Системы программирования
Трансляторы, редакторы связи, загрузчики

Начало любой программы с нулевого адреса изменило всё: появились адресно-независимые программы (старое название);
Программы теперь можно перемещать в памяти (всё стало транслироваться в относительных адресах);

Когда перемещать программу?
2 подхода:
1) программы перемещались, если для загрузки очередной программы в памяти не удавалось найти раздел необходимого размера;
2) --- пропущен ---;

Мы рассмотрели непр. распр-е, при котором программа загружается в свободный раздел целиком;

Рост мощности процессоров и объёмов оперативной памяти привёл к росту объёмов кода;

Размеры программ стали настолько большими, что эффективное управление памятью разделами стало невозможным;
Возникла мысль:
Если программа считает, что она начинается с нулевого адреса (и это поддерживается соотв. сист. таблицами), что мешает разделить адр. пр-во программы на участки равного размера и загружать в память именно эти участки программ?

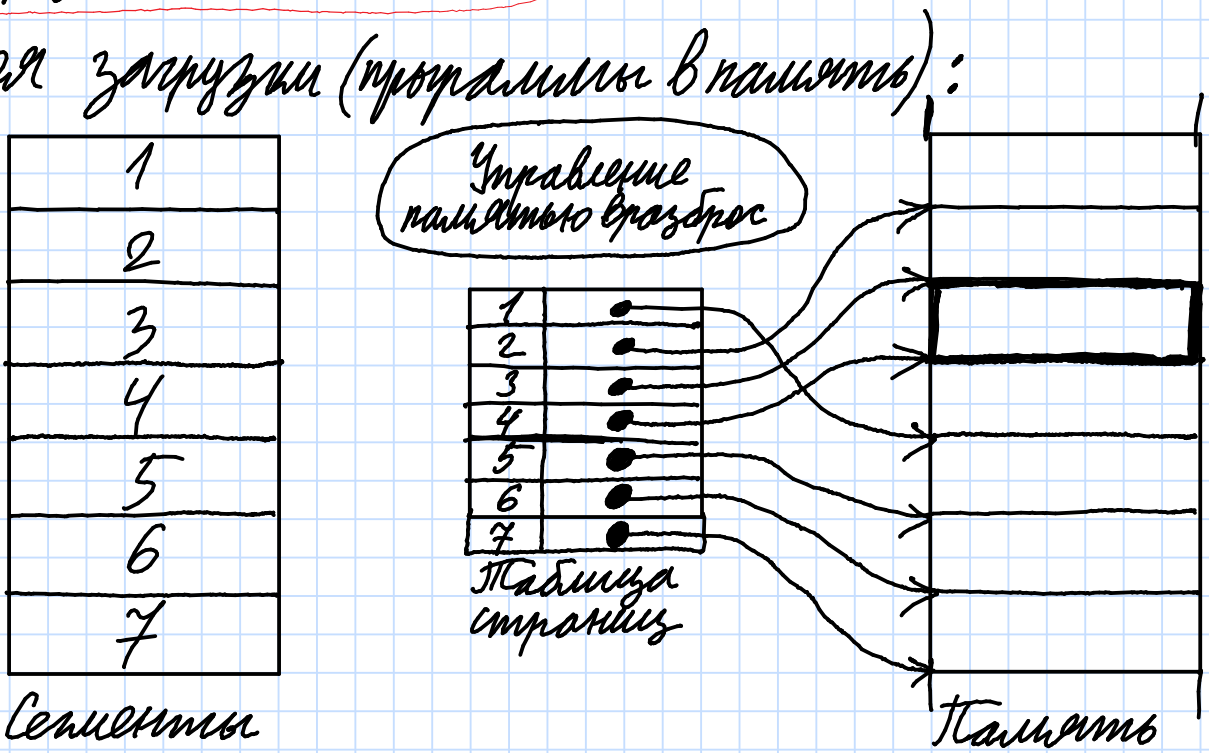
Возникла идея выделения памяти вразброс

Эти участки наз. страницами;

Адресное пр-во процесса делится на страницы ⟹ потребовалось иметь таблицы страниц, в которых указывались соотв. смещения;

Все страницы процесса загружались в оперативную память и в этой таблице указывалось, в какой кадр физ. памяти какая страница загружалась;

Идея загрузки (программы в память):



Страницами стали называть те страницы, на которые делится адр. пр-во процесса;

А для физ. памяти было принято понятие фреймы;

Кроме страниц, большие программы стали делиться на сегменты кода, данных и стека, причём сегментов кода могло быть несколько (модульная структура);

⇒ 2 способ: выделение памяти сегментами;
Отличие: для сегментов надо указывать размер;

{Регистр границы на схеме}

Первоначальная идея: программа делится на страницы и все страницы программы должны быть загружены в оперативную память;

При этом в памяти должно быть соотв. кол-во свободных кадров: если в памяти нет необходимого кол-ва свободных кадров, то такую программу нельзя было загрузить (и процесс откладывался);

Это дало толчок к след. идее:

> По арх. фон Неймана процессор может исполнять только программу, которая находится в оперативной памяти

А можем ли выполняться программа, загруженная в оперативную память не полностью (только некот. её страницы)?

Да, и это <u>основная идея виртуальной памяти</u>;

Разработчики пришли к ней от идеи выделения памяти вразброс;

Виртуальная - кажущаяся, на самом деле несуществующая;

<u>Но</u> в каждый момент времени в памяти должны находиться части кода программы, к которым обращается процессор;

Это опять помогло всё;

Виртуальная память

3 схемы управления виртуальной памятью:

1) Выделение памяти страницами по запросу;
2) Выделение памяти сегментами по запросу;
3) Выделение памяти сегментами, поделёнными на страницы по запросу;

Память выделяется процессу по запросу;

Такой запрос возникает тогда, когда процессор обращается к командам/данным, отсутствующим в физ. памяти;

Эти запросы реализованы в виде прерываний:
в страничной памяти это страничное прерывание;

Когда процесс создаётся, он сначала идентифицируется (ему выделяется дескриптор), а затем ему выделяется мин. необх. кол-во страниц (например, 3: для сегментов кода (1.) входа), данных и стека)

Далее по канонам арх. фон Неймана программа начинает выполняться команда за командой;

Но в какой-то момент процессор обращается к команде/данным, отсутствующим в памяти;
Это означает, что данная страница отсутствует в памяти;
Возникает страничное прерывание;
В рез-те прерывания, чтобы процесс мог продолжить

своё выполнение, страница должна быть загружена в память;
При этом выполняется преобразование;

У каждой страницы есть нач. адрес, к которому прибавляется смещение, которое берётся из программы, которая которая логически начинается с нулевого адреса

Когда процесс создаётся, для него с помощью таблиц страниц формируется вирт. адр. пр-во;

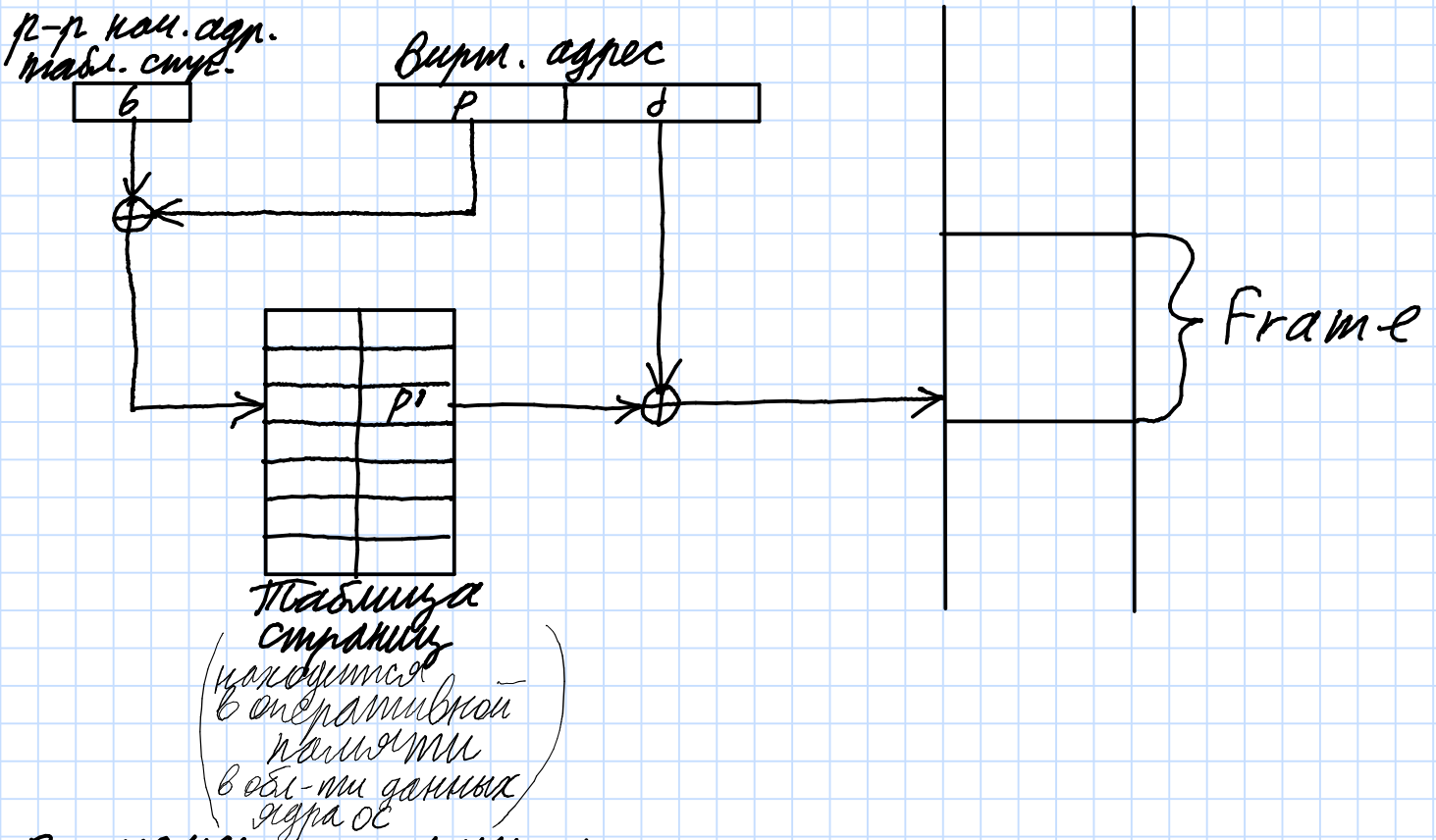
Пример есть в методичке по опт. Fork;

Чтобы загрузить вирт. страницу в физ. память и работать с этими адресами (нас интересуют физ. адреса), необходимо выполнять преобразование;

При управлении памятью страницами по запросу были предложены 3 схемы преобразования:

1) Прямое отображение:

В процессоре должен быть регистр начального адреса таблиц страниц;



р – номер страницы;
д – смещение в странице

Таблица страниц содержит дескрипторы страниц;
Адресация этих дескрипторов выполняется след. образом: к базовому адресу таблицы страниц прибавляется номер страницы (фактически смещение) и происходит обращение к дескриптору страницы. Если страница загружена в физ память, то в дескрипторе находится физ. адрес данной страницы.
↳ Это начальный адрес страницы;

Адресация данных в странице выполняется в рез-те преобразования: адрес страницы + смещение;

В Intel дескриптор страницы содержит флаги, чтобы указать, какая это страница:

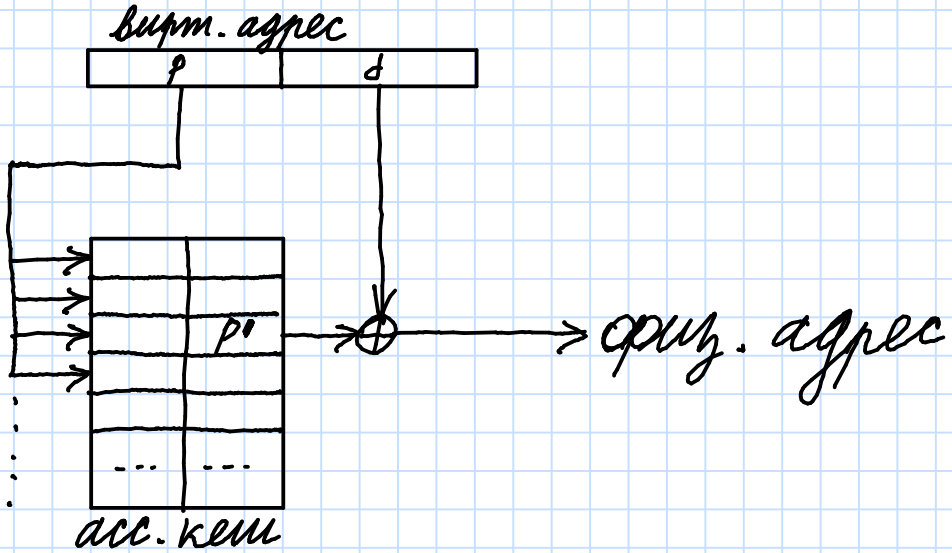
чистая / грязная,
загруженная / не загруженная
и т.д.

Эти флаги нужны системе для эффективного выполнения преобразования;

Ещё раз кратко:
дескриптор страницы → данные в странице;

Эти преобразования усложняют процесс и требуют накладных (времени) расходов, но все затраты со временем окупились увеличением быстродействия процессора;

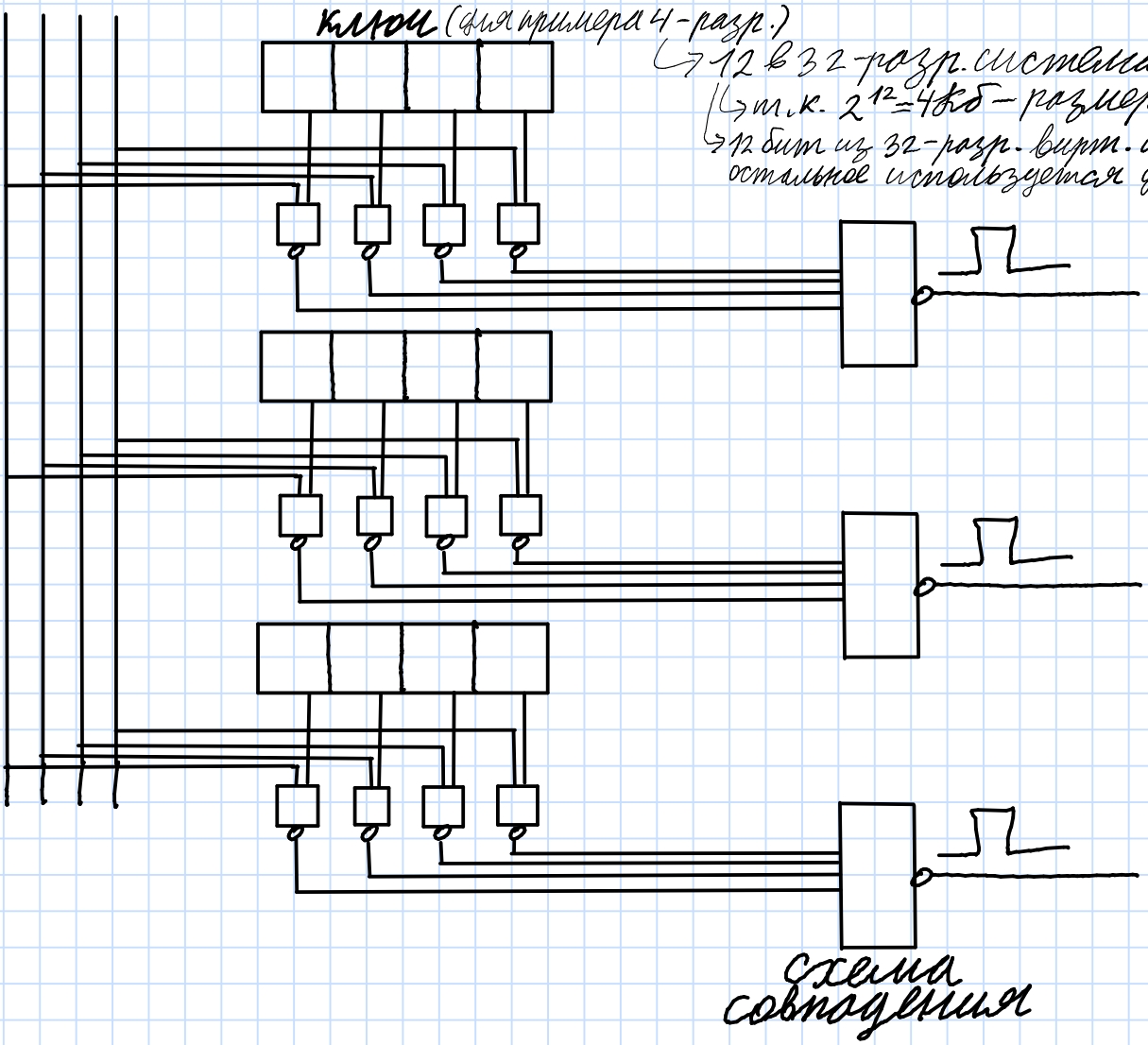
2) Для сокращения времени преобразования была предложена схема „ассоциативное отображение":
Предполагает наличие в системе спец. блока ассоциативной памяти:



Ассоциативная память всегда регистровая ⇒ ⇒ дорогая;
В асс. память должна быть загружена таблица страниц;

За один такт происходит сравнение со всеми ключами и выборка необходимой страницы;

Ключ – номер страницы;



При совпадении происходит считывание значения;

Каждый разряд должен иметь схему совпадения ⟹ кол-во логики удваивается
+ большое кол-во соединений;

Зато затраты времени уменьшились по сравнению с прежними отображениями;

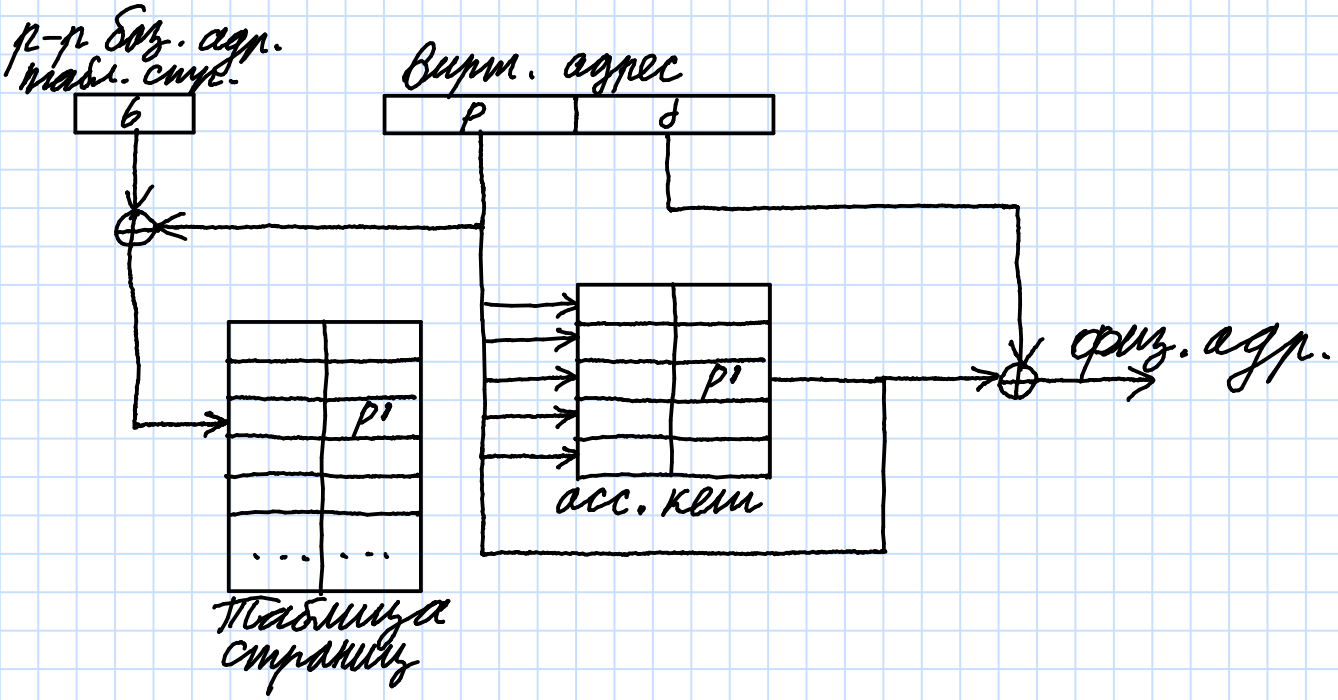
За 1 такт считывается нужное значение, но потом всё равно выполняется преобразование;

Объёмы кода растут ⟹

$\Rightarrow$ иметь большую асс. память очень дорого;

Идея плодотворная, но практически в полном объёме не реализуемая;

3) Ассоциативно-прямое отображение



В процессоре имеется базовый адрес таблицы страниц;

Сначала страница ищется в кеше. Если её там нет, то производится обращение к таблице страниц;

При этом в кеше должны находиться адреса физ. страниц, к которым были недавние обращения;

Также должно выполняться <u>замещение:</u> если страница не найдена в кеше и произошло обращение к таблице страниц, её необходимо полученный дескриптор загрузить в кеш;

При таком подходе даже небольшой кеш обеспечивает до 98% эффективных обращений, как если бы мы обращались к полностью ассоциативному кешу.

В Intel так и реализовано:
кеш TLB (Translation Lookaside Buffer)
явл. частично ассоциативным;

Такое преобразование должно выполняться на каждой команде, а то и несколько раз
(Flashback)

Страничные прерывания предполагают достаточно большую работу;

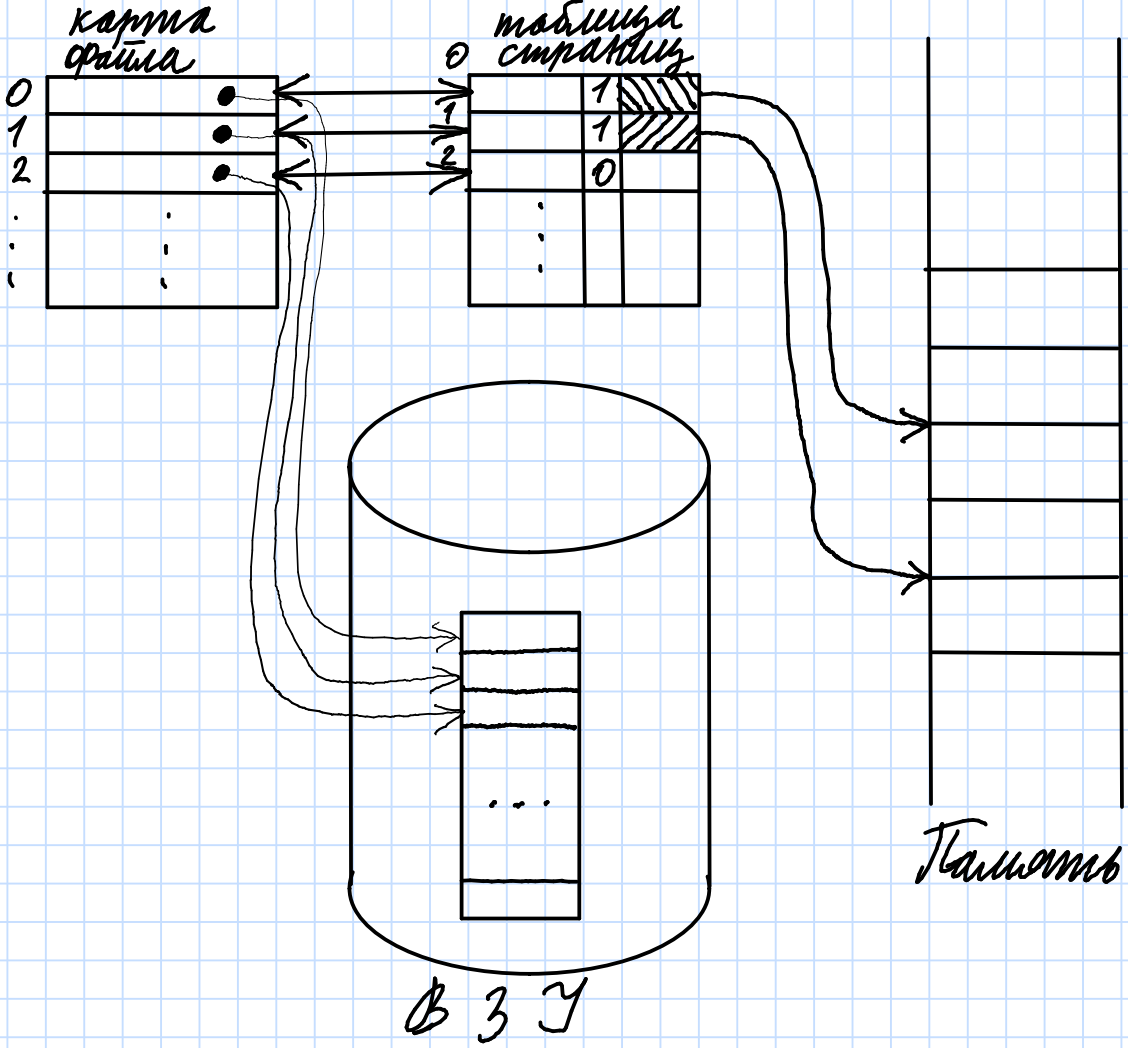
Вирт. адрес - абстракция, формируемая системой с помощью таблиц страниц;

Адреса на схемах ранее - виртуальные, а не логические, а смещение берётся из программы („из команды");

Программа до запуска была файлом и лежала на диске;

В старых системах дисковое пр-во делилось на 2 неравные части: большая - для хранения обычных файлов, меньшая - пр-во свопинга для выгрузки / подгрузки нужных страниц;

Иллюстрация:



В системе должна быть таблица – карта файла;

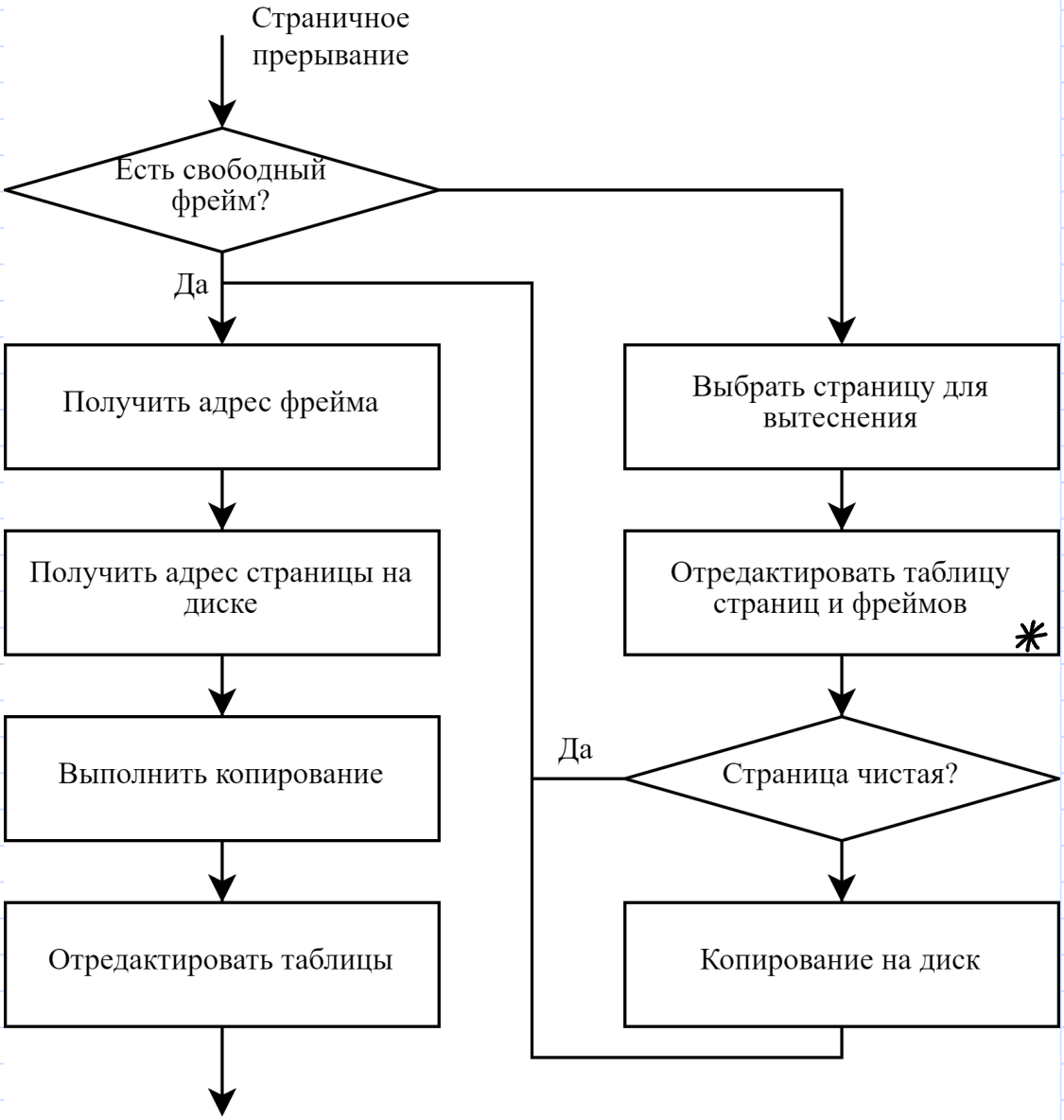
Устанавливается однозначное соответствие между страницами файла на диске (в данном случае это не файл, а процесс, но его „образ" лежит на диске) и дескрипторами в таблице страниц;

Если страница отсутствует в физ. памяти, возникает страничное прерывание, в рез-те которого необх. страница будет загружена в физ. память;

⟹ Модули ОС, предназначенные для управления памятью, вызываются обработчиком страничных прерываний;

→ в Windows это ММИ (Memory Management Unit);

При этом страница м. б. загружена, если есть свободные кадры;



выгоднее вытеснять чистую страницу (в которой не было изменений), т.к. её точная копия уже лежит на диске и ничего переносить не надо (исключается копирование);

В дескрипторе страницы всегда есть флаг обращения и флаг dirty, указывающий на то, что содержимое страницы редактировалось;

Данная страница помечается в таблице страниц как выгруженная (*);

Алгоритмы вытеснения страниц.
(page replacement)

1) Выталкивание случайной страницы
- Малые накладные расходы;
- Не исп. дискриминационным, т.к. вер-ть вытеснения страниц одинаковая
- Используется редко из-за недостатков:
  - м.б. вытеснена часто используемая страница;
  - м.б. вытеснена только что загруженная страница;

2) FIFO

Каждой странице либо приписывается временная метка (время загрузки стр. в память), либо организуется связный список страниц;
Выталкивается страница, которая дольше всего находится в памяти;