Операционные системы

Управление памятью [1]

Соловьев А. В.

ПетрГУ – КИИСиФЭ

(Rev. 2023 10 25)

[1] По материалам «Таненбаум Э., Бос Х. Современные операционные системы. СПб.: Питер, 2015.»

Абстракции памяти. Виртуальная память

Абстракции памяти

Менеджер, или *диспетчер памяти* – часть ОС, которая управляет иерархией памяти (или её частью) (следит за тем, какие части памяти используются, выделяет память процессам, которые в ней нуждаются, освобождает память, когда процессы завершат свою работу).

Без использования абстракций: модель памяти – физическая память. В памяти находится только одна программа.

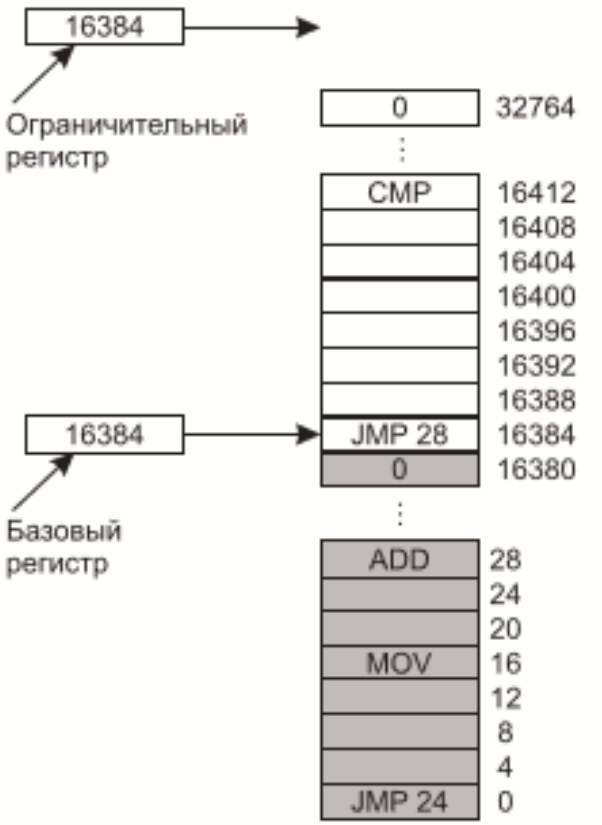
Если несколько программ –

- проблема абсолютных адресов (список перемещаемых адресов, статическое перемещение);
- проблема защиты.

Адресное пространство – это набор адресов, который может быть использован процессом для обращения к памяти. У каждого процесса имеется собственное адресное пространство, независимое от того адресного пространства, которое принадлежит другим процессам.

Базовый и ограничительный регистры

Как каждой программе можно выделить собственное адресное пространство? (Адрес 28 в одной программе означает иное физическое место, чем адрес 28 в другой программе.)



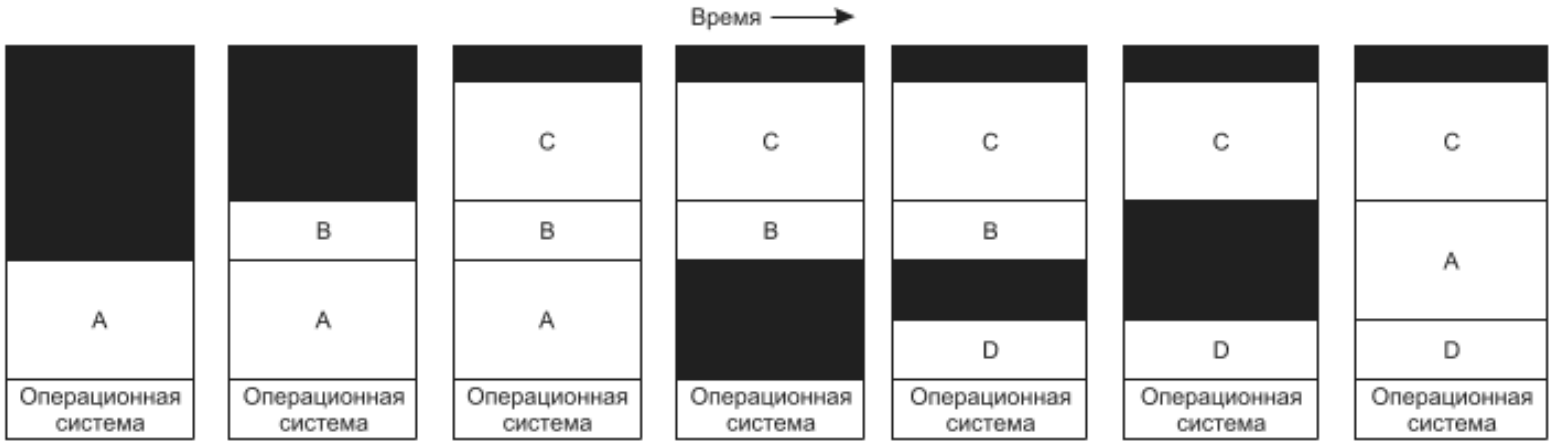
При каждой ссылке процесса на память ЦП перед выставлением адреса на шине памяти добавляет к адресу, сгенерированному процессом, значение базового регистра. Также проверяется значение ограничительного регистра.

Необходима защита этих регистров.
Повышаются затраты времени на расчёт адресов.

Свопинг

В многозадачной ОС суммарный объём оперативной памяти, необходимый для размещения всех процессов, может превышать имеющийся объём ОЗУ.

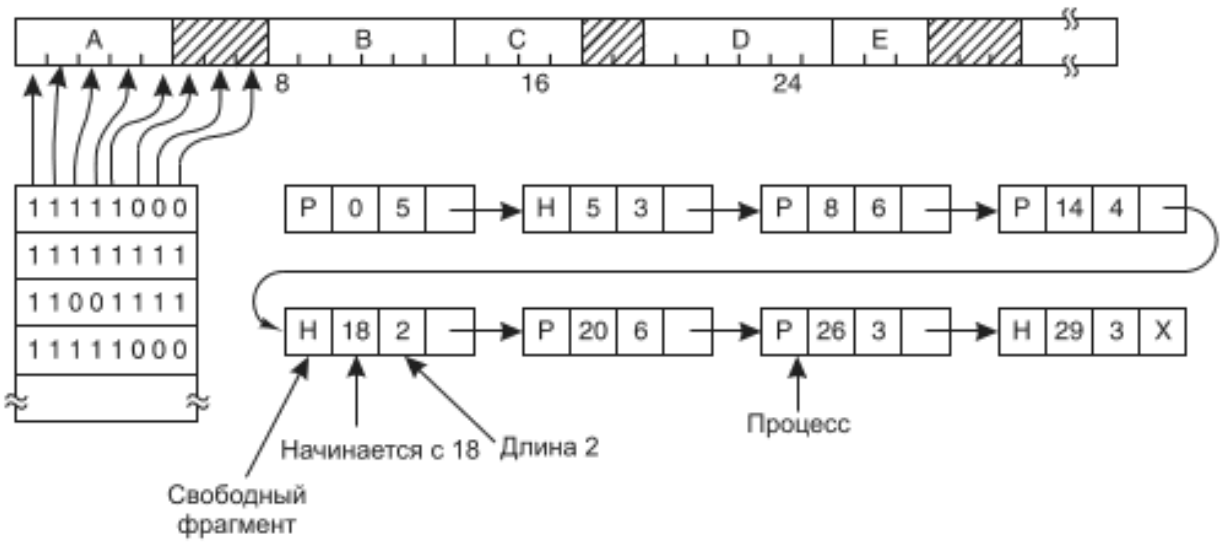
- *Свопинг (swaping)*: размещение в памяти всего процесса целиком, его запуск на некоторое время, а затем сброс на диск. Бездействующие процессы большую часть времени хранятся на диске и в нерабочем состоянии не занимают пространство оперативной памяти.
- *Виртуальная память* позволяет программам запускаться даже в том случае, если они находятся в оперативной памяти лишь частично.



Уплотнение памяти – объединение свободных областей.

Управление свободной памятью

- Битовые карты свободных участков
- Связанные списки



Алгоритмы поиска: «первое подходящее», «следующее подходящее», «наиболее подходящее», «наименее подходящее», «быстроискомое подходящее».

Страничная организация памяти (paging)

Виртуальная память (Fotheringham, 1961): у каждой программы имеется собственное адресное пространство, которое разбивается на участки, называемые *страницами*. Каждая страница представляет собой непрерывный диапазон адресов. Страницы отображаются на физическую память, но для запуска программы одновременное присутствие в памяти всех страниц необязательно. Когда программа ссылается на страницу, которая не находится в физической памяти, ОС предупреждается о том, что необходимо получить недостающую часть и повторно выполнить потерпевшую неудачу команду (в это время ЦП можно отдать другому процессу).

Страницы – блоки фиксированного размера (4Кб, 2Мб, 1Гб). Архитектура может поддерживать несколько размеров.

Сгенерированные программой адреса (*виртуальные адреса*) не выставляются напрямую на шине памяти. Вместо этого они поступают в диспетчер памяти (*Memory Management Unit (MMU)*), который отображает их на адреса физической памяти.

Страничная организация памяти: пример отображения

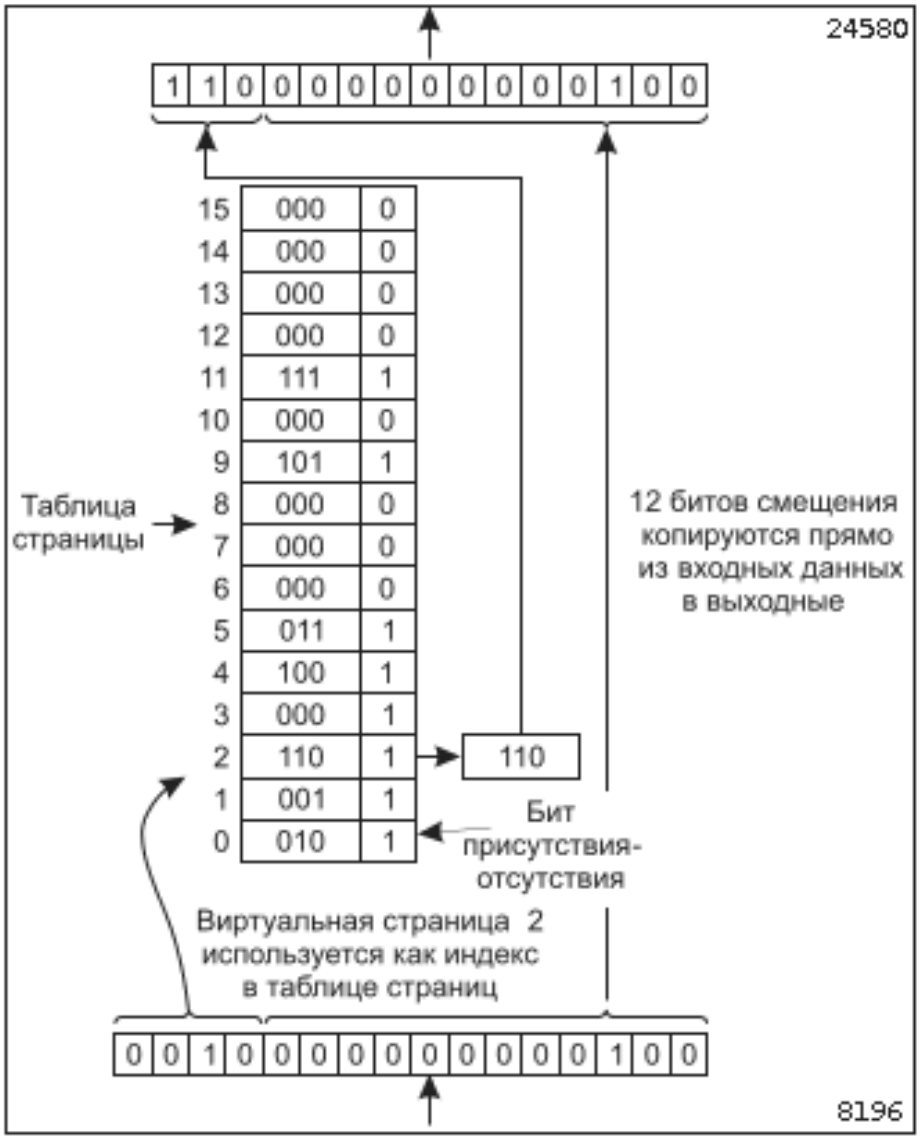


Таблица страниц

Структура записи (размер зависит от архитектуры):

- Номер страничного блока (физический адрес страницы).
- Бит присутствия/отсутствия (P).
- Биты защиты (R/W/X).
- Бит модификации (сбрасывать на диск?) (M).
- Бит ссылки/обращения (для алгоритма замещения) (A).
- Бит блокировки кэширования (для I/O).

Адрес на диске, который используется для хранения страницы, в таблице страниц не фигурирует. Он хранится в структурах ОС.

Страничная организации памяти: быстродействие

Отображение виртуального адреса на физический должно быть быстрым.

- Таблица страниц – в регистрах процессора.
- Таблица страниц всегда в памяти (в регистре только её базовый адрес).
  - *Буфер быстрого преобразования адреса (Translation Lookaside Buffer (TLB))* содержит небольшое количество недавно использовавшихся записей из таблицы страниц (x86: 64+64, IA64: 64+96, VAX: 64–256, MIPS: 96–128).

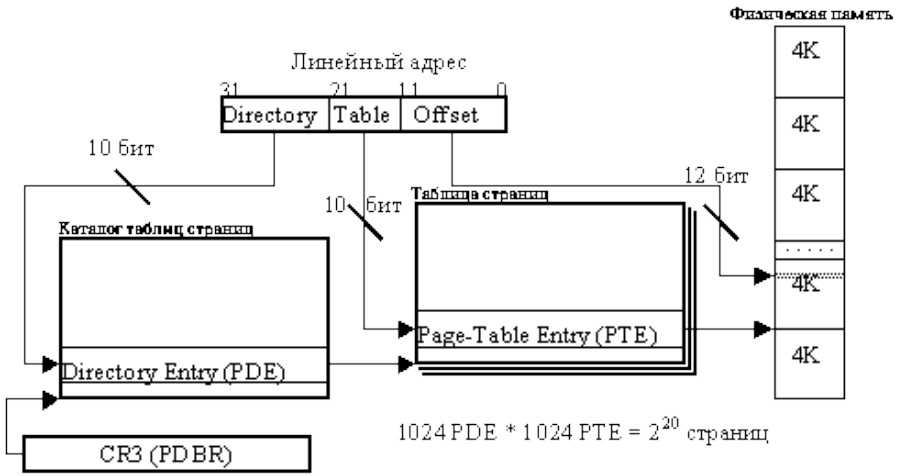
Если номер виртуальной страницы не найден в TLB, диспетчер памяти осуществляет обычный поиск в таблице страниц. Затем он выселяет одну из записей TLB, заменяя её только что найденной записью из таблицы страниц.

В некоторых архитектурах (MIPS, HP-PA, SPARC) TLB управляется программно.

Страничная организации памяти: объём памяти

Если пространство виртуальных адресов слишком обширное, таблица страниц будет иметь большой размер.

- Многоуровневые таблицы страниц (x86_32, x86_64).



- Инвертированные таблицы страниц (PPC, UltraSPARC, IA-64).

Если записи нет в TLB, чтобы найти страницу, необходимо провести поиск по всей таблице страниц. Для ускорения используется хэш-таблица виртуальных адресов.

Алгоритмы замещения страниц

Оптимальный алгоритм замещения страниц

Задача:

При возникновении ошибки отсутствия страницы ОС должна выбрать удаляемую из памяти страницу, чтобы освободить место для загружаемой страницы.

Оптимальный алгоритм:

Выбрать для выселения наименее востребованную страницу (которая позже всех понадобится).

Необходимо знать, когда понадобится каждая из присутствующих страниц, и выбрать ту, у которой время востребования в будущем больше.

Такой алгоритм не реализуем на практике.

Возможно использование в целях моделирования и тестирования.

Алгоритм NRU (Not Recently Used)

Бит A (*accessed*): было обращение к странице (ОС периодически по таймеру сбрасывает этот бит).

Бит M (*modified*): была модификация страницы (ОС сбрасывает этот бит, когда сохраняет копию страницы на диске).

Классы страниц:

1. A=0, M=0
2. A=0, M=1
3. A=1, M=0
4. A=1, M=1

Алгоритм NRU – исключения давно использовавшейся страницы:

Удаляется произвольная страница с наименьшим классом.

- Простая реализация.
- Приемлемая производительность.

Алгоритмы замещения страниц на основе FIFO

Алгоритм «первой пришла, первой и ушла» (FIFO):

ОС ведёт список всех страниц, находящихся на данный момент в памяти, в виде очереди FIFO. Недавно поступившие находятся в хвосте, поступившие раньше всех – в голове списка. При возникновении ошибки отсутствия страницы удаляется страница, находящаяся в голове списка, а к его хвосту добавляется новая страница.
(В чистом виде используется редко.)

Алгоритм «второй шанс»:

FIFO + проверка бита A самой старой страницы. Если A=0, страница старая и невостребованная, поэтому она тут же удаляется. Если бит A=1, он сбрасывается, а страница помещается в конец списка страниц и время ее загрузки обновляется, как будто она только что поступила в память. Затем поиск продолжается.
Если обращения были ко всем страницам, то алгоритм «второй шанс» превращается в простой алгоритм FIFO.
Модификация на основе циклической структуры – алгоритм «часы».

Алгоритм замещения LRU

Временная локальность: данные, интенсивно используемые несколькими последними командами, будут, скорее всего, снова востребованы следующими несколькими командами.

Алгоритм замещения наименее востребованной страницы

LRU (Least Recently Used):
Каждая запись в таблице страниц должна иметь время (счётчик) последнего обращения, обновляющийся при каждом обращении к странице. При возникновении ошибки отсутствия страницы ОС проверяет все значения счетчика в таблице страниц, чтобы найти наименьшее из них. Та страница и будет наименее востребованной (удалена).

(Принципиально возможен, но много накладных расходов.)

Алгоритм замещения NFU

Алгоритм замещения нечасто востребованной страницы
NFU (Not Frequently Used):
Каждая запись в таблице страниц должна иметь счётчик, обновляемый по таймеру: к текущему значению добавляется бит A. Так можно приблизительно отследить частоту обращений к каждой странице. Для замещения выбирается та страница, чей счетчик имеет наименьшее значение.
(Проблема нестарения счётчиков.)

Алгоритм «старения» (модификация NFU):
Бит A не прибавляется, а вдвигается слева.
Горизонт прошлого ограничен разрядностью счётчика.

Алгоритм «рабочий набор» (1)

Простейшая форма стратегии замещения страниц – *замещение страниц по требованию* (*demand paging*): при старте процесса в его памяти нет страниц, первая инструкция вызывает page fault и т. д.
Пространственная локальность: в течение любой фазы выполнения процесс обращается только к относительно небольшой части своих данных.
Набор страниц, который процесс использует в данный момент, известен как *рабочий набор*. Следует заметить, что с течением времени рабочий набор изменяется, но слабо. Приближение рабочего набора – это набор страниц, используемых в k самых последних обращений. Другое приближение – это набор страниц, используемых в течение последних m мс времени выполнения процесса (виртуального времени).
Многие системы замещения страниц пытаются отслеживать рабочий набор каждого процесса и обеспечивать его присутствие в памяти, перед тем как позволить процессу возобновить работу (*модель рабочего набора*). Это существенного сокращает количество PF. Загрузка страниц до того, как процессу будет позволено возобновить работу, называется также *опережающей подкачкой страниц* (prepaging).
Алгоритм замещения страниц: при возникновении ошибки отсутствия страницы нужно выселить ту страницу, которая не относится к рабочему набору.

Алгоритм «рабочий набор» (2)

Алгоритм WSClock: модифицированный алгоритм «рабочий набор» с кольцевой очередью.

Краткая сравнительная характеристика алгоритмов

Оптимальный	Не может быть реализован, но полезен в качестве оценочного критерия
NRU (Not Recently Used)	Простой, но грубый алгоритм (с элементами LRU)
FIFO (First In, First Out)	Может выгрузить важные страницы
Алгоритм «второй шанс»	Является существенным усовершенствованием алгоритма FIFO
Алгоритм «часы»	Вполне реализуемый алгоритм
LRU (Least Recently Used)	Очень хороший, но труднореализуемый во всех тонкостях алгоритм
NFU (Not Frequently Used)	Является довольно грубым приближением к алгоритму LRU
Алгоритм «старения»	Вполне эффективный алгоритм, являющийся неплохим приближением к LRU
Алгоритм рабочего набора	Затратный для реализации, но эффективный алгоритм
WSClock	Вполне эффективный алгоритм

Вопросы реализации систем страничной организации памяти

Локальная или глобальная политика замещения?

Как должна быть распределена память между конкурирующими работоспособными процессами?

- *Локальная политика замещения* – алгоритм замещения выбирает для вытеснения страницу того же самого процесса, что вызвал PF (хорошо подходит для выделения каждому процессу фиксированной доли памяти).
- *Глобальная политика замещения* – алгоритм замещения выбирает для вытеснения среди всех страниц (любых процессов), находящихся в памяти.

Один из способов управления распределением памяти между процессами (определения доли памяти) – *алгоритм частоты возникновения ошибки отсутствия страницы (Page Fault Frequency (PFF))*. Он подсказывает, когда нужно увеличивать или уменьшать количество выделенных процессу страниц.
Для некоторых алгоритмов замещения подходит любая политика.
Для WS – локальная.

Управление загрузкой

Когда сумма рабочих наборов всех процессов превышает объём памяти, наблюдается *пробуксовка* (частые PF).
Также показания алгоритма PFF, свидетельствующее о том, что некоторые процессы нуждаются в дополнительной памяти, но ни одному из процессов не нужен её меньший объём, говорят о пробуксовке.
Один из вариантов решения – свопинг. Какой-то процесс целиком выгружается на диск, а освободившаяся от него память распределяется между остальными процессами.
Двухуровневая диспетчеризация:

1. Диспетчер свопинга, выгружающий процессы целиком.
2. Краткосрочный диспетчер, распределяющий страницы между процессами, остающимися в памяти.

Диспетчер свопинга выбирает для выгрузки/загрузки процесс с учётом характера процесса (ограничен скоростью вычислений или скоростью работы у-в вв/выв).

Размер страницы

Аргументы за небольшой размер страницы:

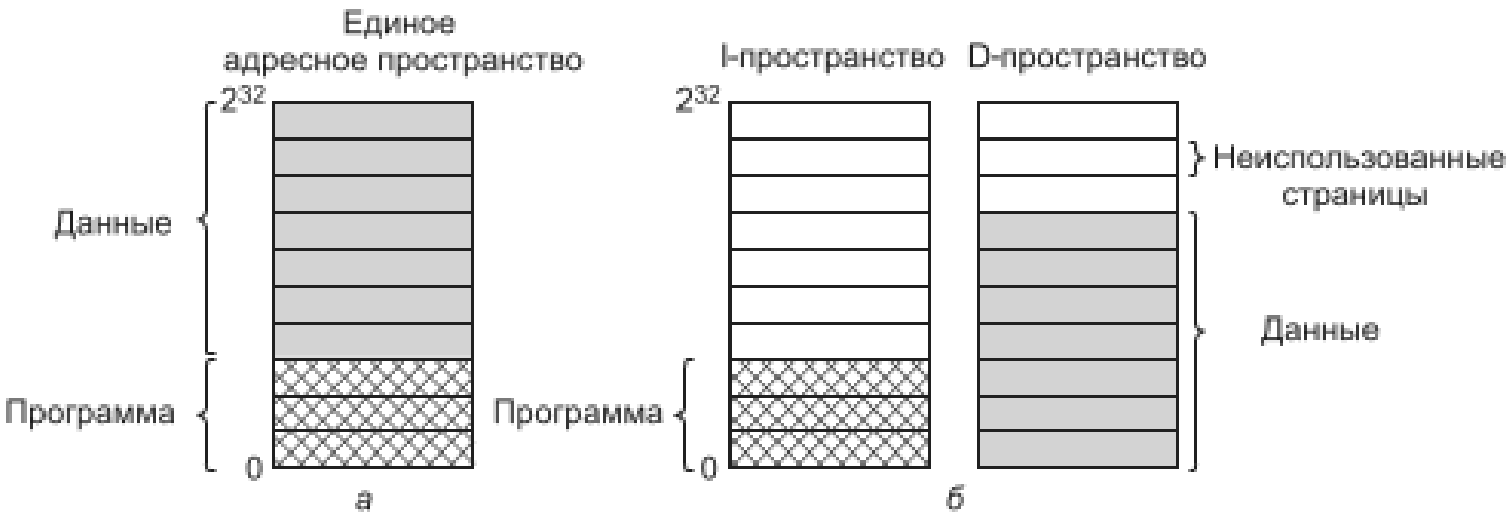
- внутренняя фрагментация («последняя» страница всегда незаполненная) меньше (при большом размере страницы в памяти будет больше неиспользуемого пространства, чем при малом).

Аргументы за большой размер страницы:

- таблица страниц меньше, эффективнее используется TLB,
- вытеснение страниц более эффективно (перенос небольшой страницы занимает практически столько же времени, что и перенос большой страницы).

Издержки (на 1 процесс) $\approx \frac{se}{p} + \frac{p}{2}$.
(s – средний размер процесса, p – размер страницы, e – размер записи таблицы страниц.
Чтобы сбалансировать все эти аргументы, ОС может использовать разные размеры страниц для разных частей системы (причём несовпадающие с аппаратными размерами).

Разделение пространства команд и данных



При разделении пространства памяти на I и D, каждое из них может иметь независимую страничную организацию (свои таблицы страниц, свои политики и алгоритмы замещения).

По крайней мере, кэш L1 в большинстве архитектур разделён.

Совместно используемые страницы

Чтобы избежать одновременного присутствия в памяти двух копий одной и той же страницы (например страницы кода программы, запущенной в нескольких экземплярах), применяются совместно используемые страницы (СИС). В таблицах страниц разных процессов фигурируют одни и те же страницы физической памяти.

Проблемы, что делать с СИС:

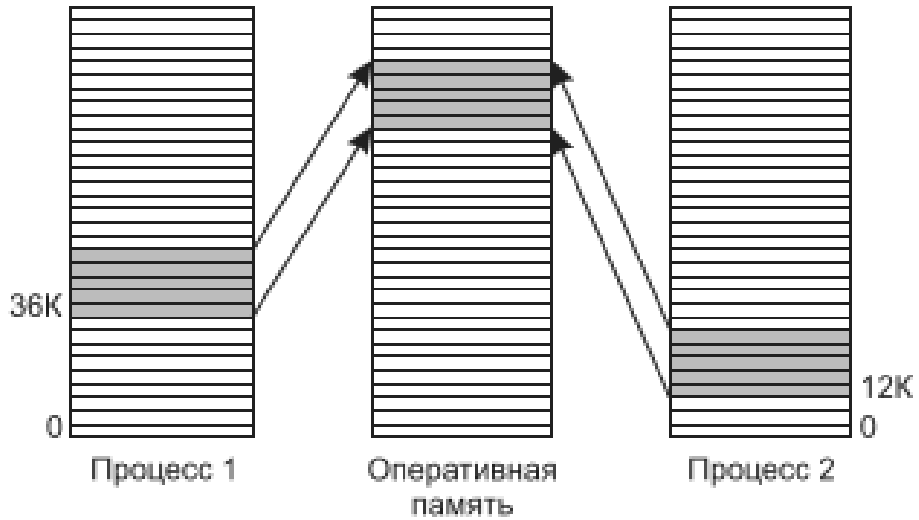
- когда процесс удаляется при свопинге,
- при завершении процесса.

Необходим специальный учёт СИС.

СИС используется при реализации copy-on-write (например, в fork).

Совместно используемые библиотеки

Концепция совместно используемых библиотек (Windows – DLL, Linux – SO). Вместо статической компоновки используется динамическая (в процессе запуска).



Совместно используемые библиотеки могут быть размещены на СИС, однако в разных процессах им не удаётся обеспечить одинаковые виртуальные адреса (проблема перемещения).

Проблема решается исключением абсолютной адресации – позиционно независимый код (специальные опции компилятора).

Отображаемые файлы

Совместно используемые библиотеки являются частным случаем более общих объектов, называемых *отображаемыми на память файлами*. Процесс может выдать системный вызов для отображения файла на какую-то часть его виртуального адресного пространства. Когда процесс выходит из рабочего состояния или явным образом демонтирует отображение файла, все измененные страницы записываются обратно в файл на диске.

Альтернативная модель для ввода-вывода: вместо осуществления операций чтения и записи к файлу можно обращаться как к большому символьному массиву, находящемуся в памяти.

ОС может реализовать возможность отображения одного и того же файла в виртуальную память нескольких процессов. Тогда они могут связываться посредством совместно используемой памяти. Запись, произведенная одним процессом в общую память, становится тут же видна, если другой процесс считывает данные из части своего виртуального адресного пространства, отображенного на файл.

Политика очистки страниц

Если возникает PF, то процесс вытеснения страницы может быть осложнён необходимостью сброса на диск вытесняемой страницы. Производительность повысится, если время от времени контент изменённых страниц будет сбрасываться на диск.
Для этого используется *страничный демон*, который обеспечивает поставку свободных страничных блоков для системы замещения страниц. Большую часть времени он находится в состоянии спячки, но периодически пробуждается для проверки состояния памяти. Если свободно слишком мало страничных блоков, страничный демон начинает подбирать страницы для выгрузки, используя какой-нибудь алгоритм замещения страниц. Если эти страницы со времени своей загрузки подверглись изменению, они записываются на диск. Как минимум, страничный демон обеспечивает чистоту всех свободных блоков, чтобы не приходилось в спешке записывать их на диск, когда в них возникнет потребность.

Участие ОС в процессе подкачки страниц (1)

При создании процесса:

1. Определить, каким будет (первоначально) объём программы и данных.
2. Создать ТС (выделить пространство в памяти и инициализировать её). (ТС не обязана быть резидентной, но она должна находиться в памяти при запуске процесса, следует предусмотреть место в области подкачки.)
3. Инициализировать область подкачки (поместить туда код программы и данные, чтобы при PF оттуда извлекать недостающие страницы). (Некоторые ОС подкачивают код программы прямо из исполняемого файла, экономя дисковое пространство и время на инициализацию.)
4. Записать информацию о ТС и области подкачки в таблицу процесса.

При планировании процесса на выполнение:

1. MMU перезапускается под новый процесс (очищается TLB, устанавливается таблица страниц нового процесса).
2. Чтобы уменьшить количество PF, в память могут быть загружены некоторые страницы процесса или все его страницы.

Участие ОС в процессе подкачки страниц (2)

При возникновении PF:

1. Определить виртуальный адрес, вызвавший PF.
2. Определить требуемую страницу, найти её на диске.
3. Найти под загружаемую страницу подходящий страничный буфер, при необходимости очистив его.
4. Загрузить требуемую страницу.
5. Вернуть назад счётчик команд, выполнить рестарт команды, вызвавшей PF.

При завершении процесса:

1. Освободить таблицу страниц процесса, страницы процесса в памяти, дисковое пространство страниц процесса.
   (Если некоторые из этих страниц совместно используются другими процессами, то страницы в памяти и на диске могут быть освобождены только тогда, когда будет прекращена работа последнего использующего их процесса.)

Блокировка страниц в памяти

Процесс сделал системный запрос на чтение из какого-то файла или устройства в буфер, находящийся в его адресном пространстве. В ожидании завершения операции ввода-вывода процесс приостанавливается, и разрешается работа другого процесса. В этом другом процессе возникает PF. Страница, содержащая буфер ввода-вывода, может быть выбрана на удаление из памяти. При использовании DMA это приведёт к повреждению данных.

Решение 1: Блокировка, или *прикрепление (pinning)* в памяти страниц, занятых в операциях ввода-вывода, чтобы они не были удалены.

Решение 2: Все операции ввода-вывода происходят с использованием буфера ядра с последующим копированием данных в пользовательские страницы.

Организация области подкачки (1)

Дисковая область подкачки может быть организована:

- В виде отдельного раздела, на котором отсутствует ФС как таковая (используются номера блоков от начала раздела), например UNIX.
- В виде заранее выделенного одного или нескольких файлов внутри обычной ФС (требуется перевод смещения в файле в адреса блоков), например Windows.

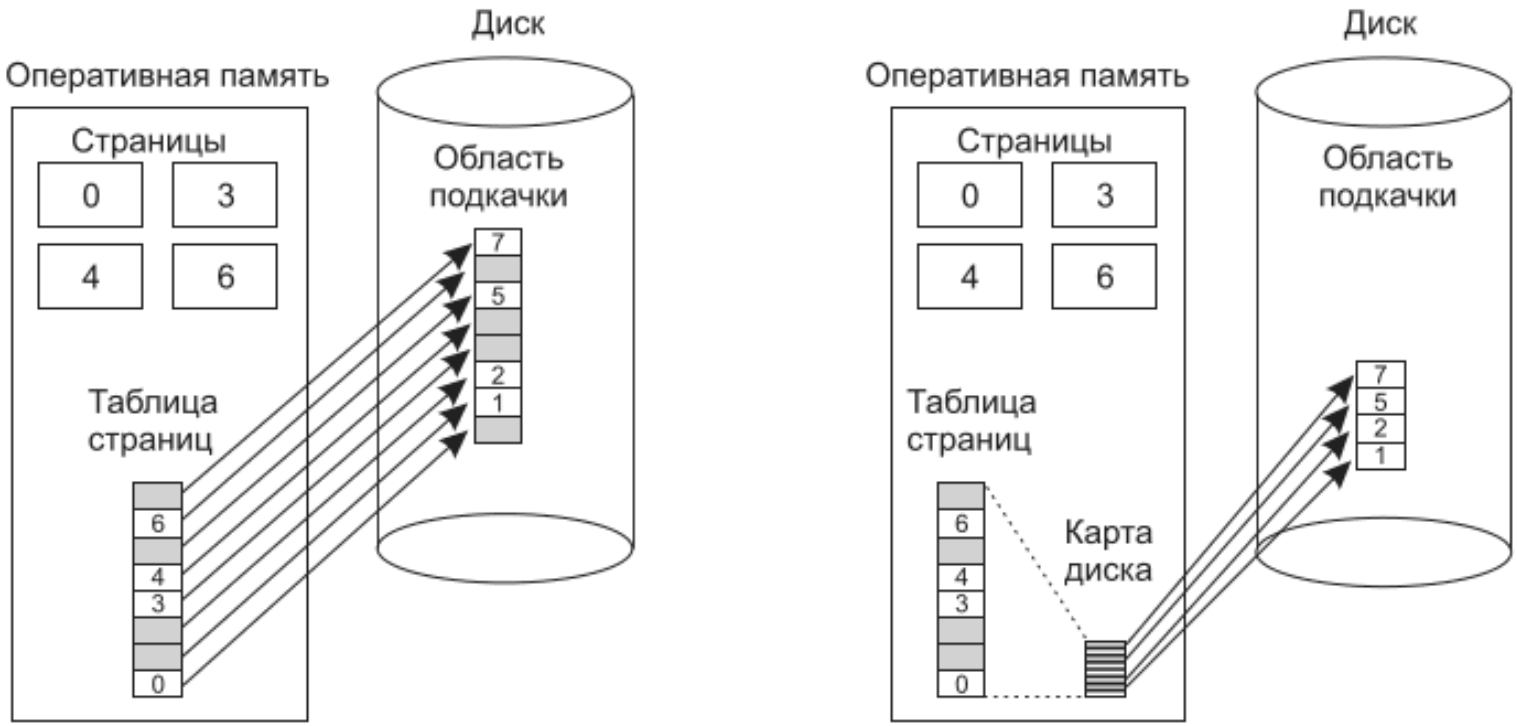
Раздел подкачки управляется как список свободных участков.

С каждым процессом связывается дисковый адрес его области подкачки. Способы инициализации области подкачки (статической):

- Весь образ процесса копируется в область свопинга, его станицы можно *получать* по мере необходимости.
- Весь процесс загружается в память, по мере необходимости его страницы можно *выгружать* на диск.

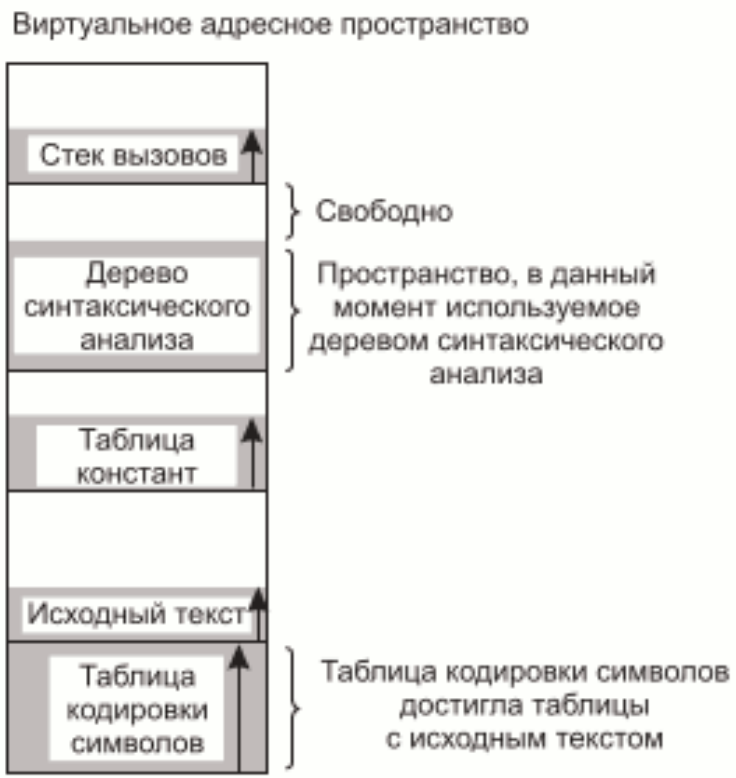
Организация области подкачки (2)

Статическая или динамическая область подкачки:

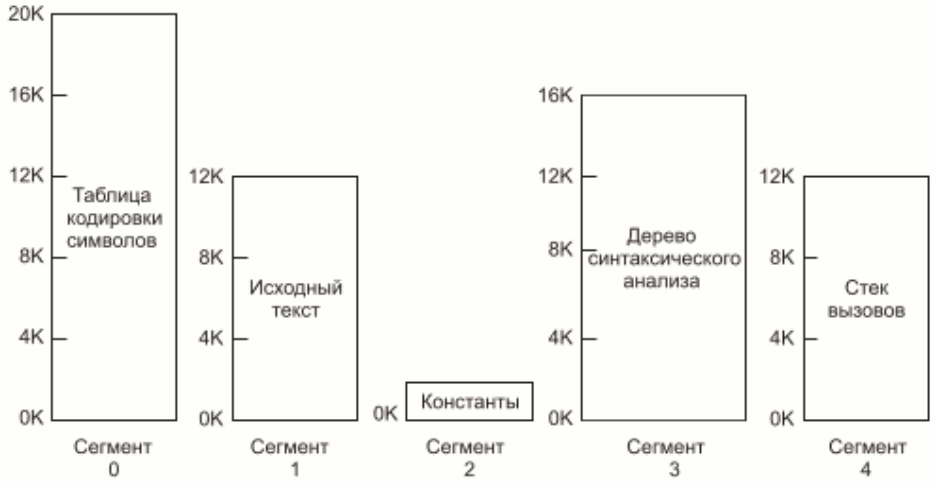


Сегментация

Одномерное АП vs. сегменты



Несколько независимых адресных пространств – *сегментов*.



Сегмент – логический объект.

Адрес = (номер сегмента, адрес внутри сегмента).

Упрощается работа с динамически меняющимися структурами.

Упрощается компоновка кода.

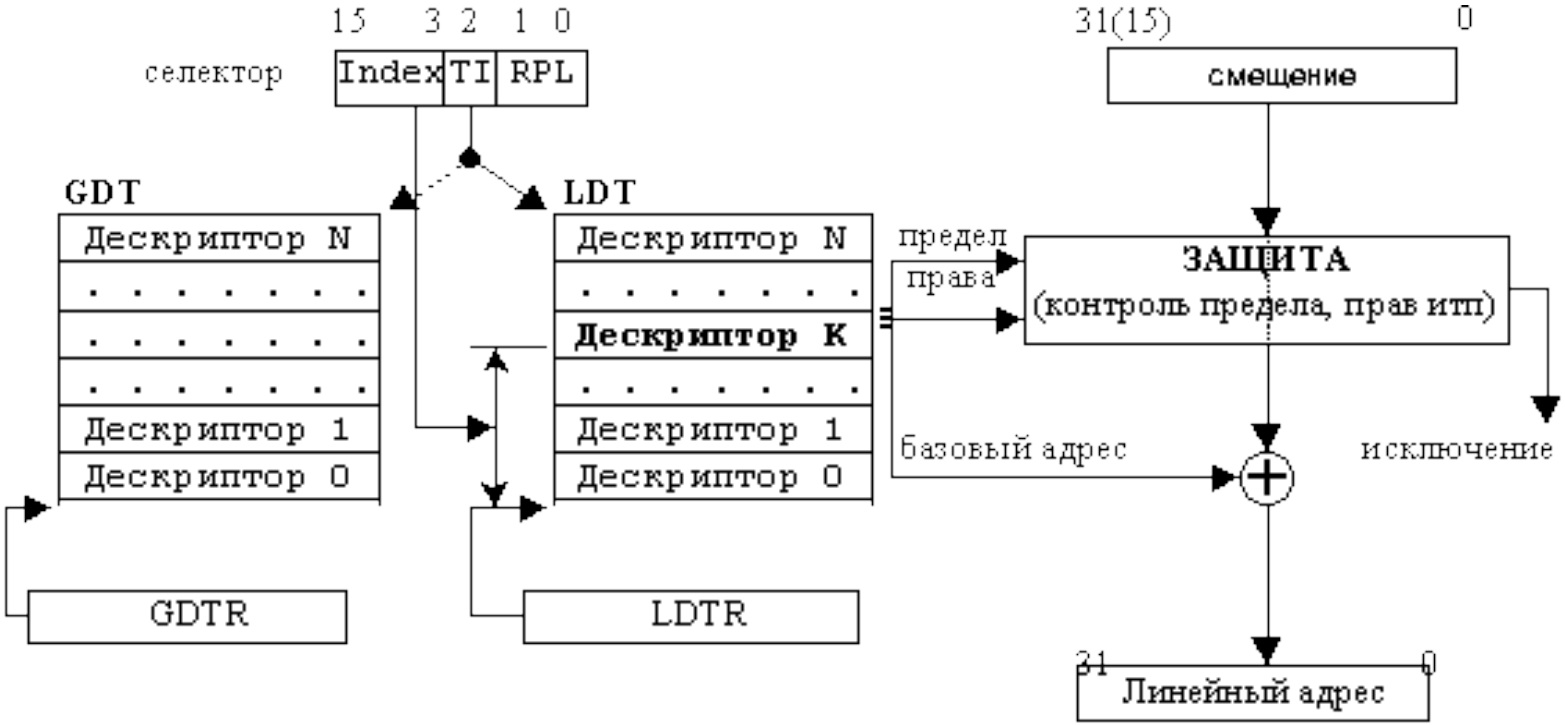
Сравнение страничной организации и сегментации

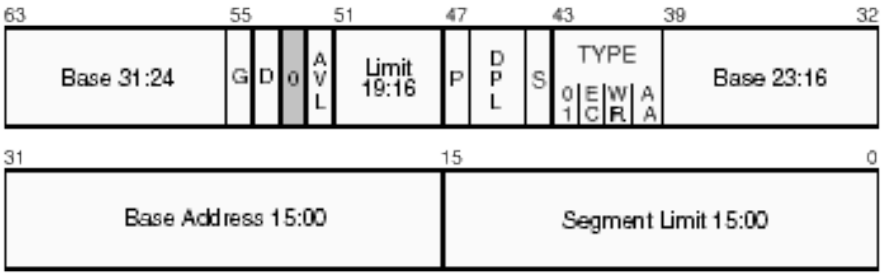
Нужно ли программисту знать, что используется именно эта технология?	Нет	Да
Сколько имеется линейных адресных пространств?	1	Много
Может ли все адресное пространство превысить размер физической памяти?	Да	Да
Могут ли различаться и быть отдельно защищены процедуры и данные?	Нет	Да
Можно ли без особого труда предоставить пространство таблицам, изменяющим свой размер?	Нет	Да
Облегчается ли для пользователей совместный доступ к процедурам?	Нет	Да

Paging: Для получения большого линейного адресного пространства без приобретения дополнительной физической памяти.

Segmentation: Чтобы дать возможность разбить программы и данные на логически независимые адресные пространства и облегчить их совместное использование и защиту.

Сегментация со страничной организацией памяти: IA-32





Использовалась в OS/2 (в Linux/Windows – нет).

Из архитектуры x86-64 сегментация исключена.