Лекция 10: Управление Памятью

Алексей Линёв
Александр Мощук
Кирилл Погорельский

somes slides are adapted from the OS course at the University of Washington

Объявления

- Домашнее задание №2 – сдаем сейчас
- Проект №2 – сдача в среду вечером
- Понедельник, среда – лекции (управление памятью)
- Пятница – промежуточный тест (midterm)
  - Темы: все, включая сегодняшний материал
    - Лекции 1-10
    - Синхронизация, планирование, процессы и потоки, структура ОС
  - Подробности в среду
- Пятница + выходные – конференция, другие мероприятия

Задачи подсистемы управления памятью

- Распределение памяти между конкурирующими процессами
  - достижение максимальной эффективности использования памяти и пропускной способности всей системы
- Предоставление удобной абстракции адресного пространства прикладной программы (для прикладных программистов, компиляторов и т.д.)
- Обеспечение изоляции процессов друг от друга
  - К настоящему моменту для нас понятия "адресуемость" и "защищенность" представляются тесно связанными, хотя на самом деле – это отдельные и независимые механизмы

Механизмы, используемые при управлении памятью

- Регистры начала и размера (Base and limit registers)
- Своппинг (swapping)
- Страничное преобразование (paging)
  - Таблицы страниц и TLB
- Сегментная адресация
  - таблицы дескрипторов сегментов
- Обработка страничных сбоев (page faults)
- Стратегии использования всех перечисленных механизмов

Современные настольные и серверные системы

- Основная абстракция, предоставляемая операционной системой прикладным программам – виртуальная память (virtual memory) или виртуальное адресное пространство
  - ВП позволяет программам выполняться, не находясь в оперативной памяти целиком
    - программы могут выполняться даже на системах с меньшим количеством памяти, чем им "необходимо"
  - большинство программ не использует постоянно весь свой код и данные
    - например, могут быть никогда не выполняющиеся ответвления кода или неиспользуемые данные, соответственно, для их размещения не обязательно выделять оперативную память
    - таким образом, ОС должна динамически изменять объем памяти, выделенной процессу, в зависимости от поведения программы во время выполнения
  - виртуальная память изолирует процессы друг от друга
    - каждый процесс использует адреса, недоступные всем остальным процессам – он имеет свое собственное изолированное адресное пространство

Механизмы, используемые для поддержки виртуальных адресных пространств

- Организация виртуальных адресных пространств требует поддержки со стороны аппаратного обеспечения и операционной системы
  - MMU's (memory management units, блоки управления памятью)
  - TLB's (translation look-aside buffers, буферы быстрого преобразования адреса)
  - page tables (таблицы страниц)
  - page fault handling (обработка страничных сбоев)
  - …
- Как правило, сопровождается поддержкой механизмов подкачки (swapping) и сегментной адресации (полностью или частично)

Мы будем изучать не только поддержку ВАП

- Зачем?
  - Полезно знать даже неиспользуемые в настоящий момент подходы
  - Большинство встраиваемых процессоров (98% из общего числа процессоров) не поддерживают использование виртуальной памяти
- В ранних системах в каждый момент времени выполнялась только одна программа
  - программы использовали непосредственно физические адреса
  - ОС загружала задачу на выполнение (возможно, используя загрузчик, настраивающий адреса программы в соответствии с адресом ее загрузки), запускала задачу, по завершении – выгружала ее
  - если программа не помещалась в памяти – прикладные программисты должны были использовать специальные техники разработки программ
    - например, оверлеи (overlays)
- Встроенные системы могут содержать и выполнять только одну программу!

- Подкачка (swapping)
  - полностью сохраняем состояние процесса (включая образ его адресного пространства) на жесткий диск
  - выполняем другой процесс
  - первый процесс "подкачивается" назад в память и перезапускается с момента остановки
- Первая система разделения времени “Compatible Time Sharing System” (CTSS), разработанная в MIT, была однозадачной системой, поддерживающей подкачку
  - только один пользовательский процесс размещается в памяти
  - после завершения обработки запроса или по истечении кванта происходило вытеснение одного процесса на жесткий диск и подкачка в память другого процесса
  - Круто! … И это еще и работало!

- Затем появилась многозадачность
  - в памяти одновременно могут размещаться несколько процессов/задач
    - для организации перекрытия вычислительной активности и операций ввода-вывода
  - возникли новые требования, связанные с управлением памятью
    - защита: нужно ограничить пространство адресов, доступное каждому процессу, чтобы они не могли "вредить" друг другу
    - быстрое преобразование адресов: скорость обращения к памяти не должна замедляться из-за используемых изменений в механизме адресации
    - быстрое переключение контекста: при переключении процесса изменение аппаратного контекста (связанное со сменой адресного пространства и изменениями в атрибутах защиты) должно выполняться быстро

Поддержка виртуальных адресных пространств и многозадачность

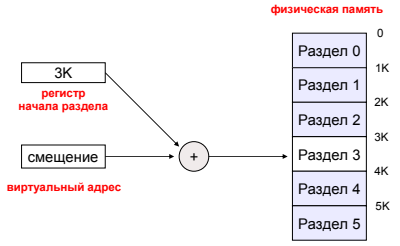
- Для облегчения управления памятью в многозадачных системах организуется использование в процессах виртуальных адресов
  - виртуальные адреса не зависят от физических адресов размещения данных в оперативной памяти
    - размещением этих данных в физической памяти управляет операционная система
  - во всех инструкциях ЦП используются виртуальные адреса
    - например, указатели, аргументы инструкций чтения/записи,…
  - виртуальные адреса транслируются в физические аппаратно, но для организации трансляции требуется поддержка со стороны ОС

- Множество виртуальных адресов, которые может использовать процесс, называется его адресным пространством
  - существует несколько различных механизмов трансляции виртуальных адресов в физические
    - мы рассмотрим их в порядке возникновения, завершив рассмотрение современными подходами
- Замечание: Мы пока еще не говорим о страничном преобразовании (paging) или виртуальной памяти – только о том, что адреса, используемые в программах – это виртуальные адреса, из которых впоследствии "получаются" физические адреса

Подход №1: Разделы фиксированного размера

- Физическая память разбивается на разделы фиксированного размера
  - размер всех разделов одинаков, разбиение никогда не изменяется
  - требуемая поддержка со стороны аппаратного обеспечения: наличие регистра начала раздела (base register)
    - физический адрес = виртуальный адрес + регистр начала раздела
    - значение регистра начала раздела загружается операционной системой при переключении процесса
- Преимущества
  - Простота
- Недостатки
  - внутренняя фрагментация: память, не используемая владельцем раздела, не может быть использована другими процессами
  - размер раздела: не существует размера, подходящего для всех процессов
    - таким образом, либо большие программы не помещаются в раздел, либо мы имеем внутреннюю фрагментацию

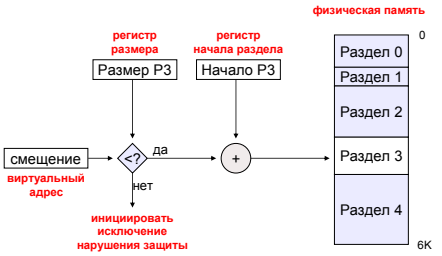
Разделы фиксированного размера
Механизм преобразования адреса



Подход №1а: Разделы фиксированного размера – вариант 2

- Физическая память разбивается на разделы фиксированного размера
  - размеры разделов могут быть различны, разбиение никогда не изменяется
  - требуемая поддержка со стороны аппаратного обеспечения: наличие регистра начала раздела (base register) и регистра размера раздела (limit register)
  - физический адрес = виртуальный адрес + регистр начала раздела
  - защищенность обеспечивается посредством выполнения проверок:
    - if (physical address > base + limit) then…
- Преимущества
  - Простота
- Недостатки
  - внутренняя фрагментация: память, не используемая владельцем раздела, не может быть использована другими процессами
  - внешняя фрагментация: предположим, имеются 2 небольших свободных раздела и поступает задача, которой требуется большой раздел
    - как выбирать размеры разделов?

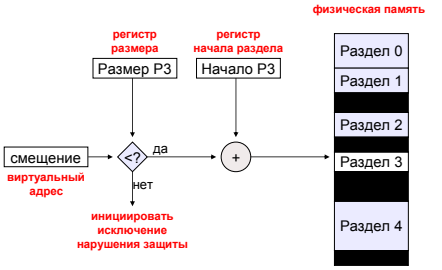
Разделы фиксированного размера, вариант 2
Механизм преобразования адреса



Подход №2: Разделы переменного размера

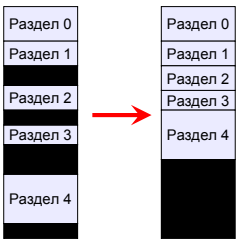
- Очевидное развитие предыдущего подхода: физическая память разбивается на разделы переменного размера
  - требуемая поддержка со стороны аппаратного обеспечения: наличие регистра начала раздела (base register) и регистра размера раздела (limit register)
  - физический адрес = виртуальный адрес + регистр начала раздела
  - защищенность обеспечивается посредством выполнения проверок:
    - if (physical address > base + limit) then…
- Преимущества
  - отсутствие внутренней фрагментации
    - процессу просто выделяется раздел требуемого размера (предполагается, что мы знаем этот размер!)
- Недостатки
  - внешняя фрагментация
    - мы создаем и уничтожаем процессы -> со временем в физической памяти появляется множество занятых разделов и неиспользуемых областей

Разделы переменного размера
Механизм преобразования адреса



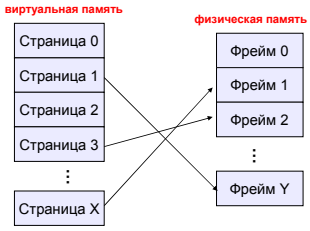
Борьба с фрагментацией

- Выгружаем процесс на диск
- Загружаем его непосредственно после (до) другого процесса
- Изменяем регистр начала раздела для процесса
- Переходим к следующему процессу



Современный подход: страничное преобразование

- Проблема внешней фрагментации решается разбиением физической и виртуальной памяти на блоки фиксированного размера



С точки зрения прикладных программ

- С точки зрения процесса, ему выделено непрерывное пространство адресов с 0 до N
  - виртуальное адресное пространство (ВАП)
- Фактически, страницы виртуальной памяти разбросаны по физической памяти
  - организовано отображение виртуальных страниц в физические
    - virtual-to-physical mapping
  - этот отображение прозрачно для процессов
- Защищенность обеспечивается невозможностью обращения программ к памяти за пределами своего ВАП
  - виртуальный адрес 0xDEADBEEF отображен на различные физические адреса в ВАП разных процессов

Страничное преобразование Механизм преобразования адреса

- Трансляция виртуальных адресов
  - виртуальный адрес состоит из двух частей:
    - virtual page number (номер страницы ВАП)
    - offset (смещение в странице)
  - номер страницы ВАП – это индекс в таблице страниц
  - запись таблицы страниц содержит номер фрейма страницы в физической памяти (page frame number, PFN)
  - физический адрес = PFN::смещение
- Таблицы страниц
  - заполняются и управляются операционной системой
  - отображают номера страниц ВАП в номера фреймов страниц в ФП
    - номер страницы ВАП – это просто индекс в таблице страниц
  - каждой странице ВАП соответствует своя запись в таблице страниц – дескриптор страницы (page table entry, PTE)

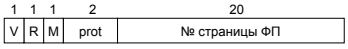
Страничное преобразование Механизм преобразования адреса

ВАП = виртуальное адресное пространство
ФП = физическая память

Пример преобразования адреса

- Сделаем следующие предположения
  - используются 32-битные адреса
  - размер страницы равен 4 Кб (4096 байт = 2^{12} байт)
  - длина номера страницы ВАП равна 20 бит (то есть ВАП содержит 2^{20} страниц), длина смещения – 12 бит
- Транслируем виртуальный адрес 0x13325328
  - № страницы ВАП = 0x13325, смещение = 0x328
  - предположим, что запись таблицы страниц с индексом 0x13325 содержит значение 0x03004
    - номер фрейма страницы в ФП равен 0x03004
    - страница ВАП № 0x13325 отображена на страницу ФП № 0x03004
  - физический адрес = № страницы ФП :: смещение = 0x03004328

Дескрипторы страниц - Page Table Entries (PTEs)



- Дескрипторы страниц управляют отображением
  - признак достоверности (valid bit) определяет, можно ли использовать данный дескриптор
    - указывает, является ли использованный виртуальный адрес корректным
    - проверяется каждый раз при использовании виртуального адреса
  - признак использования (referenced bit) указывает, выполнялся ли доступ к данной странице ВАП
    - устанавливается процессором при выполнении любой операции над содержимым страницы
  - признак изменения (modified bit) указывает, изменялось ли содержимое страницы
    - устанавливается при выполнении записи в страницу
  - биты прав доступа (protection bits) определяют, какие операции разрешены для содержимого страницы
    - чтение, запись, исполнение (read, write, execute)
  - № страницы ФП задает номер фрейма страницы ФП, на которую отображена данная страница ВАП
    - адрес начала страницы ФП = № страницы ФП :: 0…0

Преимущества страничного преобразования

- Упрощается выделение физической памяти
  - физическая память выделяется просто из списка свободных страниц
    - для выделения страницы ФП достаточно удалить ее из списка свободных
  - отсутствует внешняя фрагментация!
    - очень удобный механизм для управления разделами переменного размера и разделами, изменяющими свой размер
- Позволяет организовать действительно "виртуальную" память
  - не обязательно вся программа должна располагаться в ФП
  - страничные сбои могут быть отслежены посредством использования valid bit'а ("признака достоверности")
  - тем не менее, изначально страничное преобразование предназначалось для борьбы с внешней фрагментацией, не предполагая возможность частичного размещения программ в оперативной памяти

Недостатки страничного преобразования

- Может присутствовать внутренняя фрагментация
  - размер используемой процессом памяти может быть не кратен размеру страницы
- Накладные расходы на выполнение преобразования адреса
  - 2 обращения к памяти на одну операцию (сначала к таблице страниц, затем – собственно к данным)
  - решение: использование аппаратного кэша для уменьшения числа обращений к записям таблицы страниц
    - TLB's (translation look-aside buffers, буферы быстрого преобразования адреса) - на следующей лекции

Недостатки страничного преобразования

- Для размещения таблиц страниц тоже требуется память
  - для каждой страницы ВАП требуется свой дескриптор страницы
  - при размере адреса 32 бита и размере страницы 4 Кб требуется 2^{20} = 1,048,576 дескрипторов страниц
  - если размер дескриптора страницы равен 4 байта, то размер таблицы страниц = 4 Мб
    - как правило, для каждого процесса используется отдельная таблица страниц
    - 25 процессов требуют 100 Мб для хранения таблиц страниц
  - решение – использование таблиц таблиц страниц!
    - рассмотрим в следующий раз