# Лекция 11:
## Страничное преобразование (paging)
## Буферы быстрого преобразования адреса (TLBs)

Алексей Линёв
Александр Мощук
Кирилл Погорельский

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

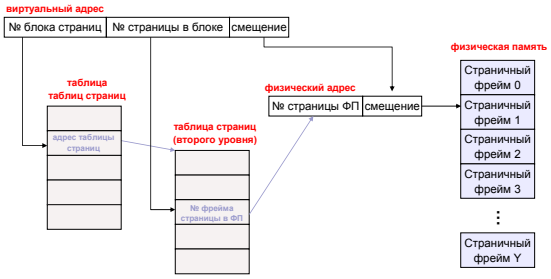
# Управление таблицами страниц

- На прошлой лекции
  - размер таблицы страниц при 32-битной адресации составляет 4 Мб!
    - слишком большие дополнительные затраты
  - как можно их уменьшить?
    - отметим, что программе требуется только часть ВАП процесса, в котором она выполняется (как правило, очень небольшая)
      - соответственно, нам необходимы дескрипторы только для используемых страниц
    - как это сделать?
      - сделать структуру таблицы страниц динамически изменяемой
  - подобные проблемы в вычислительной технике решаются введением дополнительных уровней косвенной адресации
    - двухуровневые таблицы страниц

# Двухуровневые таблицы страниц

- При использовании двухуровневых таблиц страниц виртуальные адреса состоят из 3 частей:
  - номер блока страниц ВАП, номер страницы в блоке, смещение
  - таблица таблиц страниц ставит в соответствие номерам блоков таблицы страниц второго уровня
  - таблица страниц второго уровня ставит в соответствие номеру страницы в блоке номер фрейма страницы ФП
  - физический адрес = номер фрейма страницы ФП + смещение
- Пример:
  - размер страницы 4 Кб, размер дескриптора страницы 4 б
    - сколько бит занимает смещение? для страниц размером 4 Кб – 12 бит
  - если мы желаем, чтобы таблица таблиц страниц занимала 1 страницу, в ней должно содержаться 4 Кб / 4б = 1024 записи
    - таким образом, мы имеем 1024 таблиц страниц второго уровня
  - итак: номера блока страниц ВАП – 10 бит, смещения – 12 бит
    - при использовании 32-битных адресов это оставляет нам 10 бит для номера страницы в блоке

# Двухуровневые таблицы страниц




# Как адресуются таблицы страниц?

- Где расположены сами таблицы страниц?
  - в каком/чьём адресном пространстве?
- Вариант 1: в физической памяти
  - просто адресовать, на требуется дополнительная трансляция адреса
  - однако, в этом случае таблицы страниц занимают память все время существования ВАП
- Вариант 2: в виртуальной памяти (ВАП операционной системы)
  - неиспользуемые таблицы страниц могут быть вытеснены на диск
  - но, обращения к таблицам страниц потребует дополнительных трансляций адресов
    - есть ли возможность разрушить данную рекурсию?
  - внешняя таблица страниц не должна вытесняться (быть закрепленной в физической памяти)
- Таким образом, мы применяем страничное преобразование для хранения таблиц страниц и можем применять его для всего адресного пространства самой ОС!
  - это требует закрепления в физической памяти некоторого множества кода и данных, например, обработчиков прерываний

# Как сделать страничную адресацию эффективной

- Базовая схема страничного преобразования удваивает стоимость обращений к памяти
  - сначала происходит чтение дескриптора страницы, затем – собственно обращение к данным
- Двухуровневая схема утраивает стоимость!
  - два обращения к таблицам страниц, третье – к данным
- Как повысить эффективность?
  - цель: сделать обращение по виртуальному адресу таким же эффективным, как обращение по физическому адресу
  - решение: использовать аппаратный кэш в центральном процессоре
    - на уровне аппаратного обеспечения кэшировать дескрипторы страниц
    - такой кэш называется "буфер быстрого преобразования адреса" (translation look-aside buffer,TLB)
    - TLB управляется блоком управления памятью (memory management unit, MMU)

## Буферы быстрого преобразования адреса (TLBs)

- Буферы быстрого преобразования адреса
  - позволяют получить по номеру виртуальной страницы ее дескриптор (НЕ физический адрес)
  - и сделать это за один цикл
- TLB реализуется аппаратно
  - в идеале – представляет собой полностью ассоциативный кэш (при поиске все записи проверяются параллельно)
  - ключ записи кэша – номер виртуальной страницы
  - значения записей – дескрипторы страниц
  - имея дескриптор страницы и смещение в ней, блок управления памятью может вычислить физический адрес
- TLB использует принцип локальности
  - процесс в течение ограниченного времени использует ограниченное количество своих страниц
    - типичный размер TLB – 16-48 записей (64-192 Кб)
    - достаточно, чтобы обслужить "рабочее множество" процесса
  - процент попадания в TLB очень важен!

## Управление TLB…

- Большинство трансляций виртуальных адресов выполняются на основании записей из TLB
  - >99%, но случаются и промахи (TLB misses)
  - кто помещает дескриптор в TLB в случае промаха?
- Аппаратное обеспечение (блок управления памятью)
  - знает, где в памяти расположены таблицы страниц
    - ОС формирует и поддерживает их, аппаратное обеспечение – использует
  - формат таблиц страниц задается на аппаратном уровне
  - так работают x86 процессоры
- ПО загружает TLB (ОС)
  - промах TLB вызывает исключение, ОС ищет запрошенный дескриптор страницы и загружает его в TLB
  - должно выполняться быстро (но, как правило, занимает 20-200 тактов)
    - в ЦП ISA имеются инструкции манипулирования TLB
    - ОС может выбирать любой формат таблиц страниц

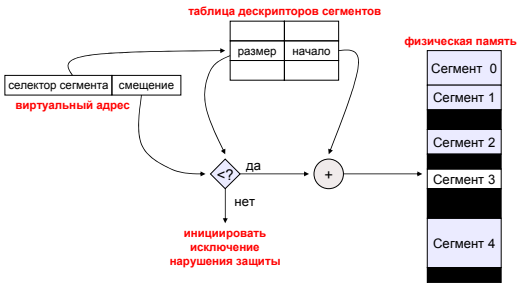
## Управление TLB

- ОС должна обеспечивать согласованность содержимого TLB и таблиц страниц
  - если ОС изменяет биты прав доступа в дескрипторе страницы, закэшированное значение данного дескриптора должно стать недействительным
- Что происходит при смене контекста процесса?
  - помним? – как правило, каждый процесс имеет собственное ВАП и собственную таблицу страниц
  - нужно признать недействительными все записи TLB! (очистить TLB)
    - фактически, это наибольший вклад в стоимость переключения контекста процесса
  - можете ли вы предложить аппаратный способ решения данной проблемы?
- При промахе TLB в него нужно загрузить новый дескриптор – значит какой-то из имеющихся должен быть замещен
  - алгоритм выбора замещаемого дескриптора называется "стратегия замещения TLB"
  - реализуется аппаратно, как правило, достаточно простая (например, LRU)

## Сегментная адресация

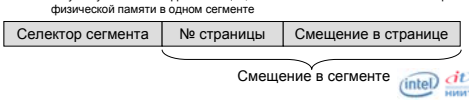
- Сегментная адресация – механизм, похожий на страничное преобразование
  - при сегментной адресации память разбивается на логические блоки
    - стек, код, данные, куча,…
  - на системе с сегментной адресацией виртуальный адрес представляет собой пару <№ сегмента, смещение>
  - с точки зрения процесса, сегменты – это единицы выделения памяти
- Естественное расширение концепции разделов переменного размера
  - разделы переменного размера = 1 сегмент на процесс
  - сегментная адресация = несколько сегментов на процесс
- Поддержка со стороны аппаратного обеспечения
  - множество пар <начало сегмента, размер сегмента>, по одной на каждый сегмент
    - хранятся в таблице дескрипторов сегментов
  - сегменты указываются посредством использования селектора сегмента – № записи в таблице

## Сегментная адресация
## Механизм преобразования адреса




## Будем использовать сегментную адресацию и страничное преобразование совместно!

- Это возможно
  - архитектура x86 поддерживает и сегментную адресацию и страничное преобразование
- Можно использовать сегменты для представления логических единиц (код, данные,…)
  - сегменты могут иметь различный, но как правило большой размер (много страниц)
- Можно использовать страницы для разбиения сегмента на блоки одинакового размера
  - каждый сегмент будет иметь собственную таблицу страниц
    - то есть нужно поддерживать таблицы страниц для сегментов, а не для ВАП процессов
  - выделение памяти еще более упрощается
    - отсутствует внешняя фрагментация, можно использовать несмежные страницы физической памяти в одном сегменте



- Linux:
  - 1 сегмент кода ядра, 1 сегмент данных ядра
  - 1 пользовательский сегмент кода, 1 пользовательский сегмент данных
  - N сегментов состояния задачи (в них сохраняются значения регистров при переключении контекста)
  - 1 локальная таблица дескрипторов сегментов (не используется)
  - для всех сегментов используется страничное преобразование
    - поддерживается 3 уровня таблиц страниц
- Замечание: это пример крайне ограниченного использования возможностей сегментной адресации!

## Тонкие приемы при использовании страничного преобразования…

- Использование наличия косвенной адресации при переходе от виртуальных адресов к физическим
  - разделяемая память
    - регионы ВАП различных процессов можно отобразить на одни и те же страничные фреймы физической памяти
      - при чтении/записи – получаем разделяемые данные
      - при выполнении – разделяемые библиотеки!
    - поскольку у каждого процесса свои дескрипторы страниц, разным процессам можно предоставить различные права доступа!
    - нужно ли разделяемый блок памяти отображать на одни и те же виртуальные адреса в различных процессах?
  - копирование при записи (copy-on-write, COW), например, при выполнении fork( )
    - вместо копирования для процесса-потомка всех страниц родительского процесса, для них организуется разделяемое адресное пространство
      - для обоих процессов к разделяемому адресному пространству разрешается доступ на чтение
      - при попытке записи возникает исключение защиты, ОС получает управление, создает для запрошенной страницы копию и выделяет ее запросившему процессу (оставляя оригинал другому), разрешает и родителю и потомку доступ на запись

## Тонкие приемы при использовании страничного преобразования

- Файлы, отображаемые в память
  - вместо использования вызовов open, read, write, close
    - "отображаем" файл в регион ВАП
      - например, в регион со стартовым адресом X
    - обращение по виртуальному адресу X+N фактически является обращением по смещению N в файле
    - сначала все страницы в используемом при отображении регионе помечаются недействительными
  - ОС читает страницу из файла при каждом обращении к недействительной странице
  - ОС записывает страницу в файл при замещении ее в оперативной памяти
    - необходимо только если содержимое страницы изменялось