Семинар 14: виртуальная память

17 марта, 2020

Фрагментация памяти

- Для выделения памяти в операционных системах обычно используется два подхода
- **Сегментация** – память делится на куски разной длины и затем раздаётся
- Однако это ведёт к **фрагментации** – в какой-то момент может быть ситуация, когда каждый второй байт занят и программа требует половину памяти; несмотря на то, что в реальности 50% памяти свободно, выделить её невозможно
- Второй подход – **paging** или **табличная адресация**

Виртуальная память и страничная адресация

- Вся физическая память разделена на *фреймы* — куски размером 4096 байт
- Вся виртуальная память аналогично разделена на *страницы*
- Трансляцией виртуальной памяти в физическую занимается *memory management unit* (MMU)

Page tables

- Специальные структуры, которые хранят отображение виртуальной памяти в физическую
- Всего существует 2^{52} страниц памяти
- Если каждая страница описывается 8 байтами, то такая структура занимает 2^{60} байт в памяти
- Нужен более экономный способ хранить это отображение

Multi-level page tables

- Идея: давайте сделаем таблицы многоуровневыми — сначала поделим всё пространство на части, каждую из этих частей ещё на части итд
- Не храня лишние «дыры» мы будем экономить место

Multi-level page tables

- Идея: давайте сделаем таблицы многоуровневыми — сначала поделим всё пространство на части, каждую из этих частей ещё на части итд
- Не храня лишние «дыры» мы будем экономить место
- Под x86-64 используются четырёхуровневые таблицы: P4, P3, P2, P1.
- Каждая таблица занимает ровно 4096 байт и содержит 512 записей (*PTE = page table entry*) по 8 байт
- Каждая запись ссылается на индекс в следующей таблице, последняя таблица ссылается на адрес фрейма

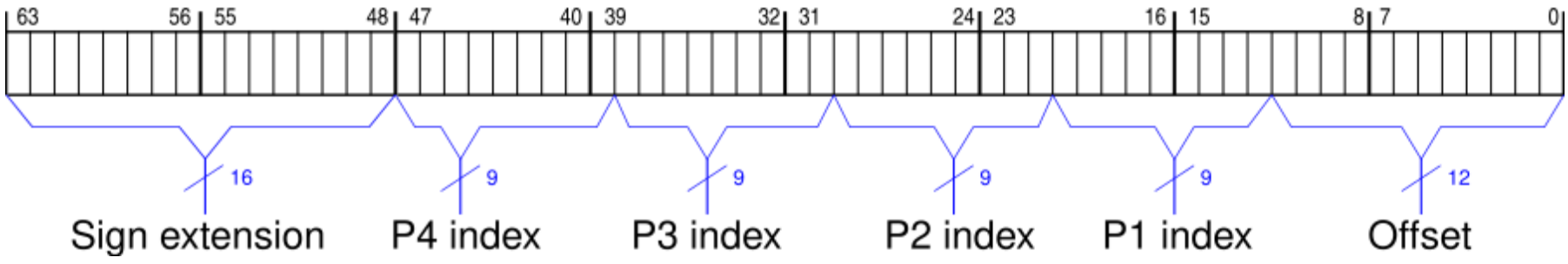
Что хранится в PTE?

- Индексация следующей таблицы или фрейма не занимает все 8 байт PTE
- Кроме неё в PTE есть ещё специальные *флаги страниц*
- Например, 1-ый бит отвечает за то, будет ли страница доступна на запись
- 63ий – за то, будет ли процессор исполнять код на этой странице
- Также в некоторые биты процессор сам пишет флаги, например, dirty-бит устанавливается всегда, когда происходит запись в страницу
- Флаги имеют иерархическую видимость: если в P2 writeable-бит равен 0, а в P4 – 1, то страница будет доступна на запись

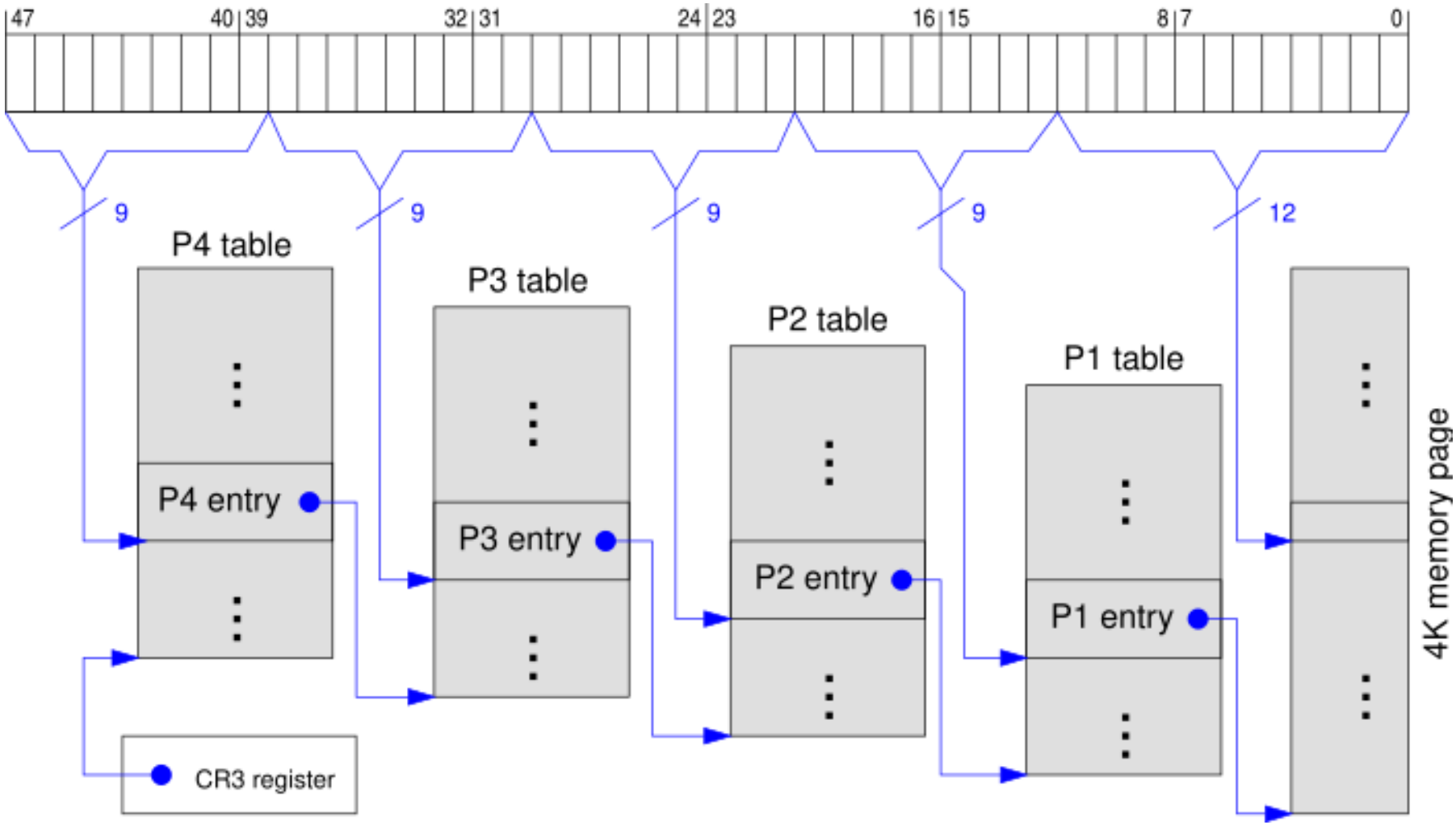
Устройство виртуального адреса

- На текущий момент x86-64 позволяет адресовать 48 бит физической памяти
- Старшие биты (с 48 по 63) должны быть sign extended копии 47ого бита
- Следующие биты (с 38 по 47) адресуют PTE в P4
- Биты с 29 по 37 адресуют PTE в P3
- Биты с 21 по 28 адресуют PTE в P2
- Биты с 12 по 20 адресуют PTE в P1, которая ссылает непосредственно на фрейм
- Биты с 0 по 11 адресуют смещение внутри фрейма

Устройство виртуального адреса



Устройство виртуального адреса



ОС и таблицы страниц

- Операционная система хранит таблицы страниц для каждого процесса
- Таблица страниц переключается каждый раз при context switch
- Физический адрес текущей P4 хранит специальный регистр CR3 (такие регистры называются *MSR = model specific registers*)
- В реальности каждое обращение к памяти не вызывает прыжки по таблицам, оно кэшируется в *TLB = translation lookaside buffer*
- При context switch TLB полностью сбрасывается

Выделение памяти: on-demand paging

- Современные ОС не выделяют всю запрошенную память сразу
- Вместо этого используется on-demand paging
- Если страницы нет в текущем memory mapping'е, то процессор сгенерирует специальное исключение, называемое *page fault*'ом
- Идея состоит в том, чтобы детектировать с помощью page fault'ов реальные обращения к памяти и только тогда её выделять

Выделение памяти: minor page

- Кроме самих таблиц страниц ОС обычно хранят свои отображения, запрошенные пользователем
- В Linux такие отображения называются *VMA = virtual memory areas*
- В ядре они хранятся в КЧ-дереве
- Когда пользователь выделяет новый участок виртуальной памяти, ядро создаёт VMA и добавляет его в дерево, но не выделяет ему страницу
- Когда происходит первое обращение к любому байту в этой области, MMU видит, что отображения на физическую память нет и выбрасывает page fault
- Ядро перехватывает его, видит, что тут должна быть физическая страничка, выделяет её, добавляет в таблицы страниц и возвращает управление в процесс
- Это и называется *minor page fault*

Выделение памяти: major page faults

- То, что было описано выше справедливо для т.н. *anonymous mappings* – маппингов, за которыми скрыта только RAM-память, отдельная для каждого процесса
- Однако, Linux позволяет производить *file memory mapping*, то есть отображение файла в память по фиксированному адресу — там будет начало файла
- Для таких страниц Linux тоже выделяет VMA, но они называются *shared*
- Для них тоже используется on-demand paging: при первом обращении генерируется page fault, ядро перехватывает исключение, читает с диска файл и копирует его в память
- Это называется *major page fault*

Интерфейс Linux для работы с VMA: mmap и munmap

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
int munmap(void *addr, size_t length);
int mprotect(void *addr, size_t len, int prot);
```

mmap

- mmap выделяет область виртуальной памяти, начиная с адреса `addr` длиной `lengh` байт
- `prot` определяют *режим страницы*
- `flags` определяют *как* будет страница замапплена
- `fd` определяет какой файл будет стоять за выделенной областью (`offset` – это оффсет файла, чтобы можно было mmap'ить куски)

mmap: prot

- **PROT_EXEC** – процессор сможет выполнять код на этой странице
- **PROT_READ** – страница будет доступна на чтение
- **PROT_WRITE** – страница будет доступна на запись
- **PROT_NONE** – к странице никак нельзя будет обратиться

mmap: flags

- **MAP_ANONYMOUS** – определяет, что область будет анонимной, `fd == -1`
- **MAP_SHARED** – определяет, что область будет доступна детям текущего процесса
- **MAP_FIXED** – говорит ядру использовать *в точности* адрес `addr` или вернуть ошибку
- **MAP_POPULATE** – говорит ядру сразу выделить физическую память для этой области

Псевдофайлы для контроля расхода памяти

- `/proc/<pid>/maps` хранит текущие VMA
- `/proc/<pid>/status` содержит статус процесса, есть куча информации о памяти
- `/proc/<pid>/mem` представляет собой память процесса (её можно читать и писать)
- `/proc/<pid>/map_files` – директория, хранит список файлов, которые замапленны в процесс

Вытеснение страниц и swap

- Если системе не хватает памяти для хранения страниц файлов, она начинает их сбрасывать на диск
- Обычно это никак не заметно на приложениях
- Однако в условиях hard memory pressure это приводит к странным последствиям: бинарник процесса может из-за этого постоянно вытесняться из памяти и перечитываться обратно
- Так можно поступать только с не-анонимными маппингами
- Для анонимных страниц обычно выделяется специальная swap-область на диске, куда ОС скидывает редко используемые страницы процессов

Gratias ago!