

Операционные Системы

Логическая память

April 28, 2017

Логическое адресное пространство

- ▶ Зачем вообще нужно разделение на логическое и физическое адресное пространства?

Логическое адресное пространство

- ▶ Зачем вообще нужно разделение на логическое и физическое адресное пространства?
 - ▶ абстракция - приложение не знает о структуре физической памяти;

Логическое адресное пространство

- ▶ Зачем вообще нужно разделение на логическое и физическое адресное пространства?
 - ▶ абстракция - приложение не знает о структуре физической памяти;
 - ▶ изоляция и защита - каждое приложение имеет свое логическое адресное пространство.

Понятие процесса

- ▶ Процесс - контейнер для ресурсов ОС

Понятие процесса

- ▶ Процесс - контейнер для ресурсов ОС
 - ▶ ОС может и не поддерживать (XX-DOS);

Понятие процесса

- ▶ Процесс - контейнер для ресурсов ОС
 - ▶ ОС может и не поддерживать (XX-DOS);
 - ▶ ОС выделяет ресурсы, например, память, процессам;

Понятие процесса

- ▶ Процесс - контейнер для ресурсов ОС
 - ▶ ОС может и не поддерживать (XX-DOS);
 - ▶ ОС выделяет ресурсы, например, память, процессам;
 - ▶ код, исполняющийся в рамках процесса, используют его ресурсы.

Понятие процесса

- ▶ Процессы, по-умолчанию, изолированы друг от друга:

Понятие процесса

- ▶ Процессы, по-умолчанию, изолированы друг от друга:
 - ▶ у каждого процесса свое логическое адресное пространство;

Понятие процесса

- ▶ Процессы, по-умолчанию, изолированы друг от друга:
 - ▶ у каждого процесса свое логическое адресное пространство;
 - ▶ т. е. код в рамках одного процесса не может залезть в память другого процесса.

Логическое адресное пространство

- ▶ Как логические адреса отображаются на физические?
- ▶ Как логические адресные пространства защищены?

Логическое адресное пространство

- ▶ Как логические адреса отображаются на физические?
- ▶ Как логические адресные пространства защищены?
 - ▶ сегментация (важно для x86);

Логическое адресное пространство

- ▶ Как логические адреса отображаются на физические?
- ▶ Как логические адресные пространства защищены?
 - ▶ сегментация (важно для x86);
 - ▶ paging (страничная организация памяти).

Сегментация

- ▶ Сегментация в Real Mode:

Сегментация

- ▶ Сегментация в Real Mode:
 - ▶ логический адрес - сегмент (*SEG*) и смещение (*OFF*);
 - ▶ *SEG* хранится в одном из сегментных регистров (*CS, SS, DS, ES, GS, FS*);

Сегментация

- ▶ Сегментация в Real Mode:
 - ▶ логический адрес - сегмент (SEG) и смещение (OFF);
 - ▶ SEG хранится в одном из сегментных регистров (CS, SS, DS, ES, GS, FS);
 - ▶ $A_{phys} = (SEG \times 16 + OFF) \bmod 2^{20}$.

Сегментация

- ▶ *SEG* - идентификатор сегмента физической памяти:

Сегментация

- ▶ *SEG* - идентификатор сегмента физической памяти:
 - ▶ сегмент *SEG* начинается по физическому адресу $SEG \times 16$;

Сегментация

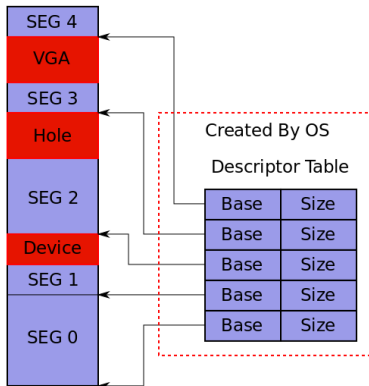
- ▶ *SEG* - идентификатор сегмента физической памяти:
 - ▶ сегмент *SEG* начинается по физическому адресу $SEG \times 16$;
 - ▶ сегмент *SEG* имеет размер 2^{16} байт.

Сегментация

- ▶ *SEG* - идентификатор сегмента физической памяти:
 - ▶ сегмент *SEG* начинается по физическому адресу $SEG \times 16$;
 - ▶ сегмент *SEG* имеет размер 2^{16} байт.
- ▶ А что если разрешить ОС изменять параметры сегмента?

Таблица дескрипторов сегментов

Physical Memory



Изоляция и защита с помощью сегментации

- ▶ Пусть ОС "выдает" каждому процессу свой дескриптор (*SEG*)

Изоляция и защита с помощью сегментации

- ▶ Пусть ОС "выдает" каждому процессу свой дескриптор (*SEG*)
 - ▶ каждый дескриптор описывает свой участок физической памяти;

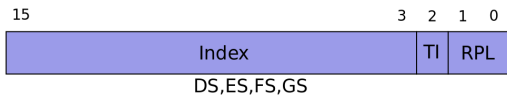
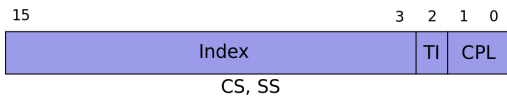
Изоляция и защита с помощью сегментации

- ▶ Пусть ОС "выдает" каждому процессу свой дескриптор (*SEG*)
 - ▶ каждый дескриптор описывает свой участок физической памяти;
 - ▶ разные процессы пользуются разными дескрипторами (т. е. разные значения *SEG*);

Изоляция и защита с помощью сегментации

- ▶ Пусть ОС "выдает" каждому процессу свой дескриптор (*SEG*)
 - ▶ каждый дескриптор описывает свой участок физической памяти;
 - ▶ разные процессы пользуются разными дескрипторами (т. е. разные значения *SEG*);
 - ▶ непривилегированному коду запрещено изменять сегментные регистры.

Селектор сегмента



Уровни привилегий в x86

- ▶ В x86 выделяют 4 уровня привилегий:
 - ▶ ring0 - ring3;

Уровни привилегий в x86

- ▶ В x86 выделяют 4 уровня привилегий:
 - ▶ ring0 - ring3;
 - ▶ ring0 - наивысший уровень привилегий (пользовательские приложения);
 - ▶ ring3 - низший уровень привилегий (код ядра ОС).

Global Descriptor Table

- ▶ В x86 таблицу дескрипторов называют GDT

Global Descriptor Table

- ▶ В x86 таблицу дескрипторов называют GDT
 - ▶ адрес и размер GDT хранятся в специальном регистре GDTR;

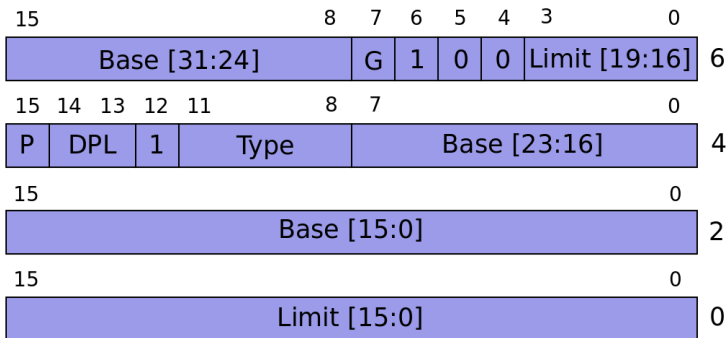
Global Descriptor Table

- ▶ В x86 таблицу дескрипторов называют GDT
 - ▶ адрес и размер GDT хранятся в специальном регистре GDTR;
 - ▶ писать и читать GDTR можно с помощью инструкций *LIDT* и *SIDT*;

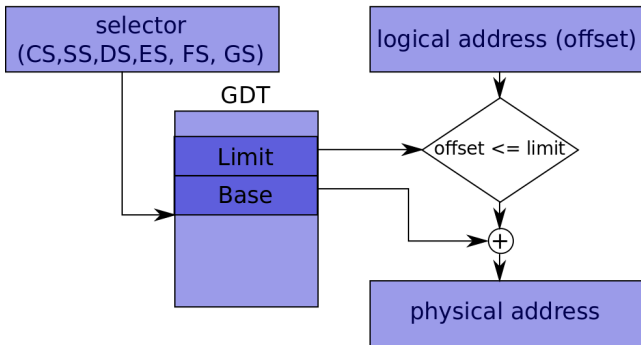
Global Descriptor Table

- ▶ В x86 таблицу дескрипторов называют GDT
 - ▶ адрес и размер GDT хранятся в специальном регистре GDTR;
 - ▶ писать и читать GDTR можно с помощью инструкций *LIDT* и *SIDT*;
 - ▶ писать в GDTR может только привилегированный код.

Дескриптор сегмента в Protected Mode



Преобразование в физический адрес



Сегментация в Long Mode

- ▶ Сегментация в Long Mode *практически* не используется

Сегментация в Long Mode

- ▶ Сегментация в Long Mode *практически* не используется
 - ▶ *ES, DS, FS, GS* обычно равны 0;

Сегментация в Long Mode

- ▶ Сегментация в Long Mode *практически* не используется
 - ▶ *ES, DS, FS, GS* обычно равны 0;
 - ▶ поля *Base* и *Limit* дескрипторов игнорируются;

Сегментация в Long Mode

- ▶ Сегментация в Long Mode *практически* не используется
 - ▶ *ES, DS, FS, GS* обычно равны 0;
 - ▶ поля *Base* и *Limit* дескрипторов игнорируются;
 - ▶ *CS* и *SS* все еще хранят *CPL*.

Сегментация в Long Mode

- ▶ Сегментация в Long Mode *практически* не используется
 - ▶ *ES, DS, FS, GS* обычно равны 0;
 - ▶ поля *Base* и *Limit* дескрипторов игнорируются;
 - ▶ *CS* и *SS* все еще хранят *CPL*.
- ▶ Вместо сегментации используется paging.

Paging

- ▶ Давайте просто использовать словарь
 - ▶ словарь хранит отображение логических адресов на физические;

Paging

- ▶ Давайте просто использовать словарь
 - ▶ словарь хранит отображение логических адресов на физические;
 - ▶ ядро ОС создает свой словарь для каждого процесса.

Paging

- ▶ Как должен выглядеть словарь?

Paging

- ▶ Как должен выглядеть словарь?
 - ▶ отображать каждый байт отдельно
непрактично;

Paging

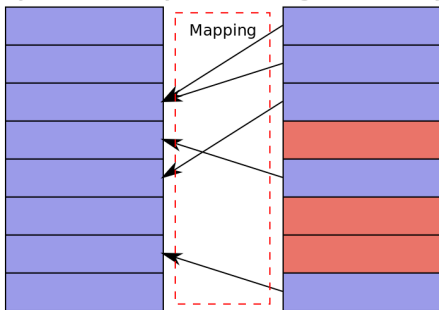
- ▶ Как должен выглядеть словарь?
 - ▶ отображать каждый байт отдельно
непрактично;
 - ▶ отображение происходит блоками
фиксированного размера (страницами);

Paging

- ▶ Как должен выглядеть словарь?
 - ▶ отображать каждый байт отдельно непрактично;
 - ▶ отображение происходит блоками фиксированного размера (страницами);
 - ▶ размер страницы определяется архитектурой (типичные размеры: 4Kb и 64Kb).

Paging

Physical Memory Logical Memory



Paging

- ▶ Как должен выглядеть словарь?

Paging

- ▶ Как должен выглядеть словарь?
 - ▶ не каждый процесс использует все логическое адресное пространство (даже в 32-битных системах и тем более в 64-битных);

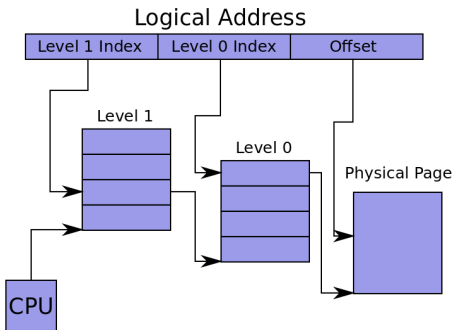
Paging

- ▶ Как должен выглядеть словарь?
 - ▶ не каждый процесс использует все логическое адресное пространство (даже в 32-битных системах и тем более в 64-битных);
 - ▶ не хочется хранить информацию для неиспользуемых страниц;

Paging

- ▶ Как должен выглядеть словарь?
 - ▶ не каждый процесс использует все логическое адресное пространство (даже в 32-битных системах и тем более в 64-битных);
 - ▶ не хочется хранить информацию для неиспользуемых страниц;
 - ▶ структура должна быть сравнительно простой.

Таблица страниц



Translation Lookaside Buffer

- ▶ А вы заметили проблему таблиц страниц?

Translation Lookaside Buffer

- ▶ А вы заметили проблему таблиц страниц?
 - ▶ мы хотим прочитать 1 байт по некоторому логическому адресу;

Translation Lookaside Buffer

- ▶ А вы заметили проблему таблиц страниц?
 - ▶ мы хотим прочитать 1 байт по некоторому логическому адресу;
 - ▶ процессор должен прочитать записи в нескольких таблицах.

Translation Lookaside Buffer

- ▶ Процессор кеширует результаты трансляции в TLB:

Translation Lookaside Buffer

- ▶ Процессор кеширует результаты трансляции в TLB:
 - ▶ TLB может значительно ускорить обращение к памяти;

Translation Lookaside Buffer

- ▶ Процессор кеширует результаты трансляции в TLB:
 - ▶ TLB может значительно ускорить обращение к памяти;
 - ▶ если код не обращается каждый раз к новой странице.

Translation Lookaside Buffer

- ▶ Процессор не может отследить изменения в таблицах страниц:

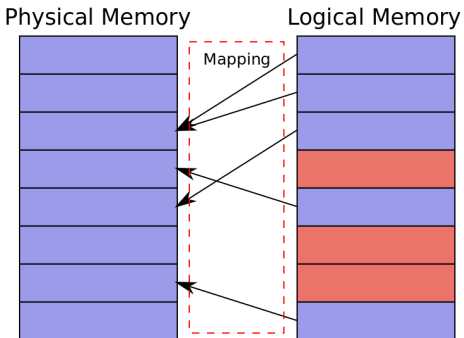
Translation Lookaside Buffer

- ▶ Процессор не может отследить изменения в таблицах страниц:
 - ▶ TLB не прозрачен, т. е. необходимо явно "сбрасывать" записи;

Translation Lookaside Buffer

- ▶ Процессор не может отследить изменения в таблицах страниц:
 - ▶ TLB не прозрачен, т. е. необходимо явно "сбрасывать" записи;
 - ▶ об этом тоже должно заботиться ядро ОС.

Page Fault



Page Fault

- ▶ Не все записи в таблицах страниц используются
 - ▶ что, если код обратится к логическому адресу, для которого нет отображения?

Page Fault

- ▶ Не все записи в таблицах страниц используются
 - ▶ что, если код обратится к логическому адресу, для которого нет отображения?
 - ▶ генерируется специальное исключение - Page Fault.

Защита памяти

- ▶ Paging также позволяет запретить некоторые действия с памятью:

Защита памяти

- ▶ Paging также позволяет запретить некоторые действия с памятью:
 - ▶ мы уже видели запрет на обращение к памяти;

Защита памяти

- ▶ Paging также позволяет запретить некоторые действия с памятью:
 - ▶ мы уже видели запрет на обращение к памяти;
 - ▶ запись в какой-то участок логической памяти;

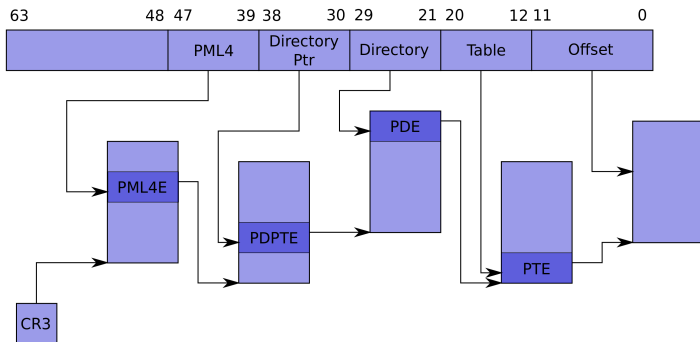
Защита памяти

- ▶ Paging также позволяет запретить некоторые действия с памятью:
 - ▶ мы уже видели запрет на обращение к памяти;
 - ▶ запись в какой-то участок логической памяти;
 - ▶ исполнение кода из какого-то участка памяти;

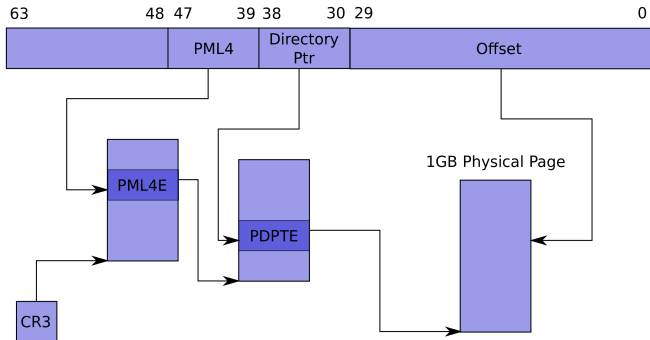
Защита памяти

- ▶ Paging также позволяет запретить некоторые действия с памятью:
 - ▶ мы уже видели запрет на обращение к памяти;
 - ▶ запись в какой-то участок логической памяти;
 - ▶ исполнение кода из какого-то участка памяти;
 - ▶ обращение непривилегированного кода.

Paging в x86 Long Mode



Paging в x86 Long Mode



Paging в x86 Long Mode



- ▶ *P* - если 0, запись не используется;
- ▶ *R/W* - если 0, то запись запрещена;
- ▶ *U/S* - если 0, то запрещен непривилегированный доступ;
- ▶ *PS* - если 1, это последний уровень;
- ▶ *XD* - если 1, то запрещено исполнение.

Резюме

- ▶ Логическое и физическое адресное пространства:

Резюме

- ▶ Логическое и физическое адресное пространства:
 - ▶ программы используют логические адреса (указатели);

Резюме

- ▶ Логическое и физическое адресное пространства:
 - ▶ программы используют логические адреса (указатели);
 - ▶ процессор использует физические адреса;

Резюме

- ▶ Логическое и физическое адресное пространства:
 - ▶ программы используют логические адреса (указатели);
 - ▶ процессор использует физические адреса;
 - ▶ ОС определяет как логические адреса отображаются на физические.

Резюме

- ▶ Понятие процесса:

Резюме

- ▶ Понятие процесса:
 - ▶ каждый процесс имеет свое логическое адресное пространство;

Резюме

- ▶ Понятие процесса:
 - ▶ каждый процесс имеет свое логическое адресное пространство;
 - ▶ процессы изолированы друг от друга.

Резюме

- ▶ Сегментация и страничная адресация памяти:

Резюме

- ▶ Сегментация и страничная адресация памяти:
 - ▶ ОС использует эти аппаратные механизмы для организации изоляции процессов;

Резюме

- ▶ Сегментация и страничная адресация памяти:
 - ▶ ОС использует эти аппаратные механизмы для организации изоляции процессов;
 - ▶ многие современные архитектуры с поддержкой защиты памяти используют paging (и очень немногие сегментацию).