САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
Математико-механический факультет
Кафедра системного программирования

Реализация механизмов виртуальной памяти для x86 архитектуры в ОСРВ Embox

Курсовая работа студента 345 группы
Ефимова Глеба Дмитриевича

Научный руководитель
аспирант кафедры
системного программирования . . . . . Бондарев А.В.

Санкт-Петербург
2012

Содержание

Введение

Виртуальная память - один из способов управления памятью ЭВМ, использующийся в многозадачных операционных системах. Данная технология позволяет программисту не думать о совместном владении памятью с другими приложениями и защите данных своей программы и даных уровня ядра, также становится возможным использовать все адресное пространство, не смотря на отсутствие соответствующего объема ОЗУ. Есть два основных способа организации виртуальной памяти : сегментная и страничная адресация пространства. Сегментная адресация заключается в делении памяти на некоторые части произвольного(необходимого на тот момент) размера, называемыми сегментами(segment). Виртуальный адрес складывается таким образом из номера сегмента и смещением внутри него. При страничной адресации все адресное пространство состоит из блоков одинаковой длины, которые называются страницами (page). Виртуальный адрес получается по похожему принципу, что и при сегментной адресации, но чуть более сложнее. Он состоит из индекса или мультииндекса в таблицах виртуальной памяти и смещения внутри выбранной страницы. Далее в статье будут подробно рассмотрены оба этих способа на примере Intel 80x86.

У процессоров архитектуры x86 имеется несколько режимов работы:

- Real Mode (Режим реальных адресов или реальный режим)
- Protected Mode (Защищенный режим)
- System Management Mode (Режим системного управления)

Реальный режим присутствует в 80286(1982г.) и более поздних процессорах этой архитектуры. Он позволяет использовать адреса из 20 битов, что дает возможность адресовать до мегабайта памяти. Режим реальных адресов не поддерживает защиту памяти, многозадачность или уровни привилегий кода. В частности при работе в нем, имеется прямой программный доступ ко всей памяти, включая адреса ввода/вывода и периферийных аппаратных устройств. При запуске, все x86-процессоры находятся в реальном режиме.

Впервые защищенный режим появился в процессоре 80286 , а с выходом 80386(1985г.) был значительно расширен и присутствует во всех последующих процессорах этого типа. Он позволяет системному программному обеспечению использовать различные механизмы работы с памятью, например виртуальная память, безопасная многозадачность и другие технологии, созданные для управления прикладным программным обеспечением операционной системой.

Режим системного управления был впервые реализован в процессоре 386SL(1990г.), и затем был доступен в поздних процессорах. При входе в этот режим процессор останавливает исполнение текущих задач и запускает специальный код, чаще всего встроенное программное обеспечение или аппаратный отладчик, который выполняется в высоко привилегированном режиме. Этот режим предназначается для обработки ошибок системы(памяти, чипсета), управление питанием и других важных моментов.

Из представленных вариантов работы наиболее содержательным для нас является защищенный. Это основной режим, в котором современные x86-процессоры проводят большую часть своего времени. Мы будем в дальнейшем рассматривать режим реальных адресов и защищенный режим.

Постановка задачи

Данная курсовая работа выполнялась в рамках операционной системы реального времени Embox. Главная цель заключалась в добавлении поддержки виртуальной памяти через страничную организацию. На основе этой задачи нужно также добиться еще двух результатов. Первый - необходимо обеспечить для каждого процесса свое линейное адресное пространство. Второй - виртуальная память каждого процесса должна быть защищена от действий других процессов. Если быть точнее, то нужно изолировать процессы друг от друга, но при необходимости разрешать совместное использование физической памяти.

Механизмы

Главная особенность Protected mode это поддержка механизма виртуальной памяти. Как было сказано выше, в настоящее время есть два способа реализации этой технологии: сегментная и страничная организация памяти.

Сегментная адресация памяти

Рассмотрим, как связаны между виртуальный и физический адреса. Пусть у нас уже имеется виртуальный адрес (см. Рис.1). При формировании физического адреса можно выделить три стадии:

1. Вычисление эффективного адреса(смещение);
2. Затем находится адрес в линейном виртуальном 4-х гигабайтном адресном пространстве; (базовый адрес складывается с эффективным)
3. Линейное виртуальное адресное пространство отражается на физические блоки в основной памяти

На первом этапе просто берутся младшие 16 бит, которые и являются эффективным адресом, их еще называют смещением внутри сегмента.

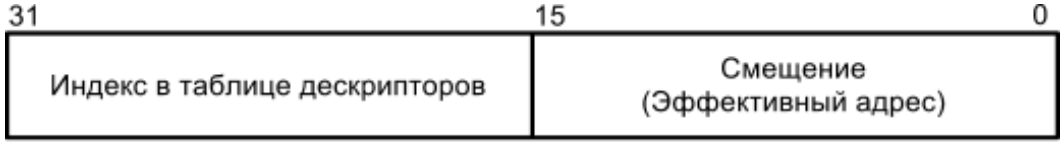

Рис. 1: Виртуальный адрес

На втором этапе нужно найти сам сегмент. В этом нам поможет вторая часть нашего виртуального адреса, которая служит индексом в таблице дескрипторов. Дескриптор это структура данных, которая описывает сегмент (см. Рис. 2). Она имеет размер 8 байт и содержит в себе такую информацию, как базовый адрес, права доступа и предел сегмента. Таким образом, зная индекс в таблице дескрипторов, мы знаем всю информацию о нужном нам сегменте. Затем уже

происходит вычисление физического адреса путем сложения смещения и базы сегмента.

Рис. 2: Структура дескриптора

Но процессору необходимо знать где находится таблица дескрипторов. Для этого есть специальные регистры. GDTR (Global Descriptor Table Register) - регистр из 6 байт: первые 32 бита определяют адрес глобальной дескрипторной таблицы (GDT - Global Descriptor Table), оставшиеся 16 бит содержат в себе предел таблицы или, другими словами, её размер. Помимо неё есть еще множество локальных дескрипторных таблиц (LDT Local Descriptor Table), из которых в один момент времени доступна только одна, какая - определяется регистром LDTR. LDTR (Local Desriptor Table Register) - регистр из 16 бит, который содержит в себе селектор необходимой таблицы. Селектор это структура (см. Рис.3), размером 2 байта, которая содержит в себе индекс дескриптора в GDT, который описывает сегмент, где расположена таблица, и прочую административную информацию. Обращение к глобальной таблице происходит только один раз, когда изменяется LDTR, затем найденый дескриптор помещается в недоступную часть регистра. Стоит отметить, что GDT присутствует в системе в единственном экземпляре, а количество LDT может доходить до 8192.

TT - тип таблицы (LDT или GDT)
ТПД - требуемые права доступа

Рис. 3: Структура селектора

Для включения сегментации необходимо подготовить глобальную и, если нужно, локальные таблицы дескрипторов, затем устанавливается 0-й бит регистра CR0. CR0 (Control Register 0) - управляющий регистр процессора, который контролирует режим работы и состояние процессора, состоит из 32 бит. В нем нас интересуют два бита: 0-й и 31-й. Нулевой бит, если установлен, включает режим защиты, где поначалу начинает использоваться сегментная адресация. Тридцать первый бит, при установке его, разрешает страничное преобразование, об этом сейчас и пойдет речь.

Страничная адресация памяти

У сегментной адресации есть три основных минуса. Первый и самый весомый заключается в том, что сегменты могут иметь различную длину, а это вызывает сильную фрагментацию и следовательно приводит к неэффективному использованию памяти. Второй минус это избыточность такой модели, на практике средний размер сегмента гораздо больше чем запрашиваемые данные, поэтому при загрузке или выгрузке сегмента возникают лишние перемещения. Третий минус это сложность получения физического адреса, обращение к памяти происходит весьма часто и постоянное использование операции сложения делает процесс более затратным по времени. Как уже было сказано во введении, в новом механизме стали использовать блоки памяти одинаковой длины, так называемые страницы (page). Чаще всего используются страницы объемом 4 килобайта, но есть и другие реализации.

Давайте разберемся, как происходит страничное преобразование в стандартной схеме(см. Рис. 4). Виртуальный адрес разбивается на три части: индекс в каталоге

таблиц, индекс в таблице страниц и смещение внутри страницы. Запись, соответствующая индексу в каталоге таблиц, содержит в себе физический адрес нужной таблицы страниц. Затем индекс в таблице страниц указывает нам на запись, в которой находится ссылка на конкретную страницу, а именное её адрес. И наконец, к адресу страницы прибавляется смещение и мы получаем нужную ячейку памяти. На рисунке ниже показана схема страничной адресации.

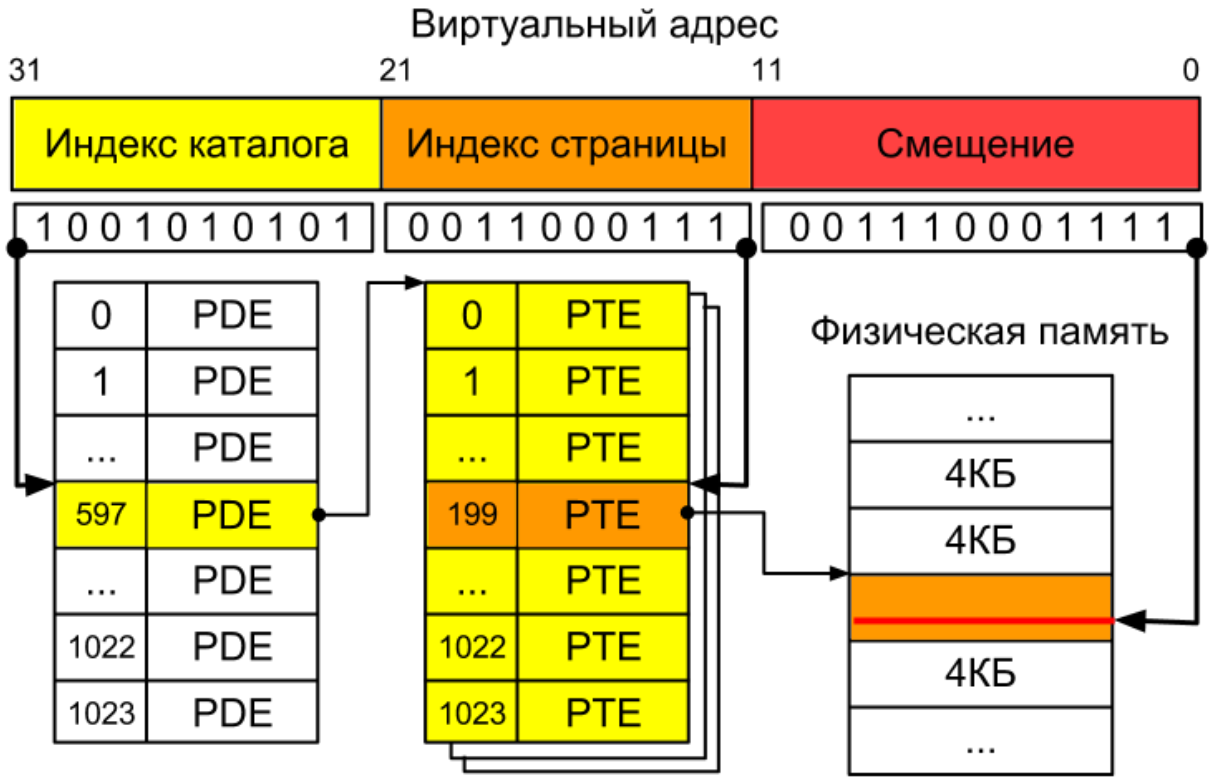

Рис. 4: Схема страничной трансляции

В механизме страничной адресации присутствуют такие типы данных как каталоги таблиц (Page Directory) и таблицы страниц (Page Table), обе эти структуры состоят из 1024 элементов(записей) размером 4 байта. Записи из каталога таблиц ссылаются на таблицы страниц, которые в свою очередь ссылаются на конкретные страницы. Все эти структуры выравнены по границе страницы.

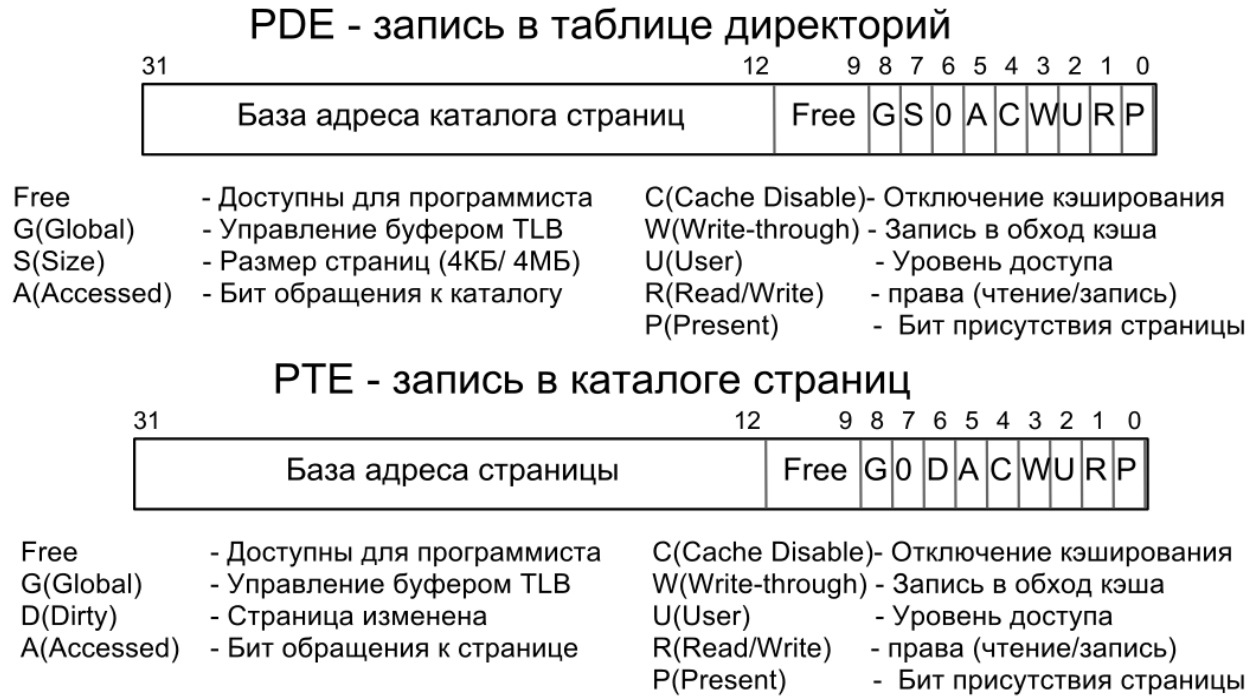

Рис. 5: Дескрипторы таблиц страниц и самих страниц

Записи в таблице директорий и каталогах страниц называются дескрипторами таблиц страниц и каталогов страниц соответственно. Помимо базы адреса в них содержатся управляющие флаги, из которых, в настоящий момент, для решения поставленных задач нам особенно важны два : уровень доступа к каталогу/странице (бит 2) и возможность записи в каталог/страницу (бит 1). Таким образом с помощью структуры этих дескрипторов решается задача изолирования адресного пространства процессов.

Реализация

Самая важная вещь для механизма страничной адресации это таблицы трансляции. Если они отсутствуют или процессор не знает об их существовании, то процессор сделает перезагрузку. выброшено исключение. Таким образом необходимо создать эти таблицы и оповестить об этом процессор. Особенность их создания заключаются в том, что все структуры должны быть выровнены по границе страницы. Когда разметка таблиц в памяти закончена, необходимо заполнить дескрипторы значениями по умолчанию. В нашем случае мы выставим следующие атрибуты: уровень доступа - только администратор, а права доступа - разрешение записи. На начальном этапе нам не нужно виртуальное пространство большого размера, поэтому вполне можно обойтись небольшим количеством страниц, например один к одному с физической памятью, в случае необходимости мы легко сможем добавить недостающий объем. Так же мы должны особенным образом заполнить дескрипторы, ссылки которых указывают на страницы, где размещены наши таблицы. На этом заканчивается инициализация таблиц, теперь мы должны уведомить об этом процессор. Для этого в архитектуре x86 существует управляющий регистр CR3 (Control Register 3). В нем содержится база адреса каталога таблиц и пара флагов управления кэшированием. Соответственно мы должны загрузить в этот регистр ссылку на таблицу директорий, из которой доступны все каталоги таблиц.

Теперь все готово в к переходу на страничный режим адресации, как говорилось ранее, за это отвечает регистр CR0, а именно 31 бит (см. Рис.6).

```
22 void mmu_on(void) {
23
24     asm (
25         "mov %cr0, %eax\n"
26         "or  $0x80000000, %eax\n"
27         "mov %eax, %cr0"
28     );
29 }
```

Рис. 6: Включение страничной адресации

Теперь процессор находится в режиме страничной трансляции адресов, сейчас мы увидим, насколько это так. Для этого в качестве тестирующей процедуры (см. Рис. 7) мы попробуем вызвать разные функции с одного виртуального адреса. А именно,

сначала вызвав первую функцию, мы заменим ее адрес страницы размещения в таблице страниц, на адрес страницы размещения второй функции. Другими словами мы смэппируем (mapping) страницу второй функции на адрес первой.

```
104     printf("\nPaging starting...\n");
105
106     /* enabling paging */
107     mmu_on();
108
109     function1();
110
111     printf("\nMapping a new function to the old address\n");
112     map_region(function1,function2);
113
114     function1();
115
116     printf ("\nEnding mmu testing...\n");
117
118     /* disabling paging */
119     mmu_off();
```

Рис. 7: Тестирующая процедура

Разумеется чтобы такие действия были возможными и процессор не выбрасывал исключений необходимо чтобы функции были выровнены по границе страницы (см. Рис. 8).

```
68 void  __attribute__((aligned(PAGE_SIZE))) function1(void) {
69     printf("\n\tInside the first function\n");
70 }
71
72 void __attribute__((aligned(PAGE_SIZE))) function2(void) {
73     printf("\n\tInside the second function\n");
74 }
```

Рис. 8: Объявление тестовых функций

Результат работы тестирующей процедуры изображен на Рис. 9

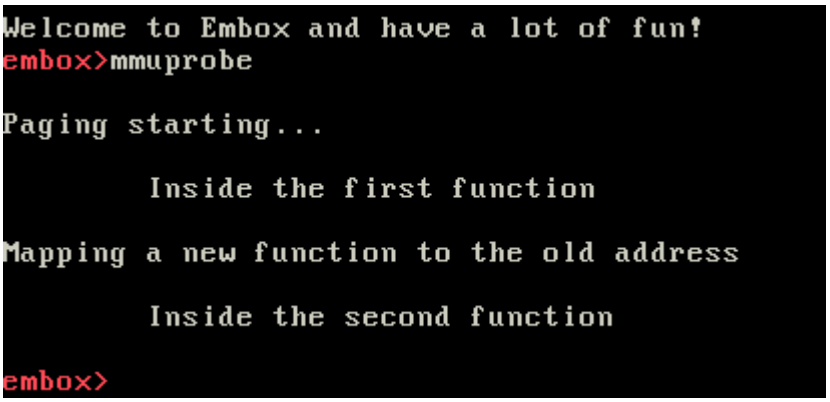

Рис. 9: Результат работы тестирующей процедуры

Из полученных результатов можно сделать вывод, что страничная трансляция работает, страницы были переназначены верно, функции вызывались корректно.

Заключение

В ходе курсовой работы был реализован механизм виртуальной памяти для архитектуры x86 путем страничной адресации. Основываясь на этом результате также были достигнуты возможности создания линейных адресных пространств для каждого процесса и изолирования их друг от друга. Таким образом ОСРВ Embox получила поддержку виртуальной памяти для x86 архитектуры

Во время исследования механизма виртуальной памяти были рассмотрены возможные варианты организации, существовавшие проблемы и их решения. Также была подробна изучена архитектура IA-32 на предмет внутреннего устройства и предоставляемых возможностей для создания виртуальной памяти.

Логичным продолжением данной работы будет доработка интерфейса работы с механизмом виртуальной памяти и в последствии создание полноценного менеджера виртуальной памяти.

Список литературы

[1] Embox – Essential toolbox for embedded development,

http://code.google.com/p/embox

[2] OSDev – Operating System development,

http://wiki.osdev.org/Main_Page

[3] Intel Architecture Software Developer's Manual. Vol.1: Basic Architecture

[4] Intel Architecture Software Developer's Manual. Vol.3: System Programming Guide.

[5] Операционные системы: разработка и реализация / Э.Танненбаум, А. Вудхалл