Зыков Юрий. 327

1. ***Процесс загрузки и инициализации персональных ЭВМ архитектуры x86. Функции загрузчика. Загрузка ядра JOS.***

Lab1 – знать + помнить код.

После включения персонального компьютера (ПК) на базе процессора семейства Intel 80x86, его процессор начинает работу в реальном режиме адресации с сегментной организацией и выполнение инструкций процессора с адреса FFFF:0, инициализированного в паре регистров CS:IP (Code Segment : Instruction Pointer) после снятия сигнала RESET. В конце доступного процессору адресного пространства оперативной памяти из ПЗУ материнской платы отображен загрузчик базовой системы ввода-вывода (BIOS), на который передается выполнение по инструкции безусловного перехода, отображенной по адресу FFFF:0.

Программа инициализации BIOS с помощью программы POST проверяет, что устройства компьютера работают корректно и инициализирует их.

Затем BIOS опрашивает по порядку устройства из заранее сформированного (в консольной утилите CMOS SETUP) списка загрузочных устройств, пока не найдет загрузочное устройство. Если такое устройство не будет найдено, BIOS выведет на консоль ПК сообщение об ошибке и процесс загрузки будет остановлен. При обнаружении BIOS’ом загрузочного устройства, с последнего будет произведена попытка чтения данных его нулевого сектора (512 байт, в которых, как предполагается, должна находиться загрузочная запись). После успешного завершения чтения данных нулевого сектора загрузочного устройства в оперативную память по начальному адресу 0000:7С00, BIOS передает на этот адрес дальнейшее исполнение инструкций процессора.

Нулевой сектор загрузочного жесткого диска содержит так называемую «Главную загрузочную запись» (MBR — Master Boot Recorв), содержащую как данные первых четырёх записей таблицы разделов, так и инструкции процессора Intel 80x86, обеспечивающие поиск по этим записям активного раздела, с которого, посредством одной из функций базовой системы ввода-вывода — INT 13h, реализованной в виде обработчика прерывания процессора, будет считана загрузочная запись операционной системы. Этот загрузочный сектор, как правило, зависит от операционной системы и должен произвести загрузку в оперативную память ядра операционной системы с последующей передачей на него исполнения инструкций процессора. Если активного раздела не существует, или загрузочный сектор активного раздела некорректен, MBR может загрузить резервный начальный загрузчик и передать управление ему. Резервный начальный загрузчик должен выбрать раздел (зачастую с помощью пользователя), загрузить его загрузочный сектор и передать ему управление.

**Загрузчик операционной системы** — системное программное обеспечение, обеспечивающее загрузку операционной системы непосредственно после включения компьютера.

Загрузчик операционной системы:

* обеспечивает необходимые средства для диалога с пользователем компьютера (например, загрузчик позволяет выбрать операционную систему для загрузки);
* приводит аппаратуру компьютера в состояние, необходимое для старта ядра операционной системы (например, на не-x86 архитектурах перед запуском ядра загрузчик должен правильно настроить виртуальную память);
* загружает ядро операционной системы в ОЗУ. Загрузка ядра операционной системы не обязательно происходит с жесткого диска. Загрузчик может получать ядро по сети. Ядро может храниться в ПЗУ или загружаться через последовательные интерфейсы (это может пригодиться на ранней стадии отладки создаваемой компьютерной системы);
* формирует параметры, передаваемые ядру операционной системы (например, ядру Linux передаются параметры, указывающие способ подключения корневой файловой системы);
* передаёт управление ядру операционной системы.
1. ***Переключение контекстов. Кооперативное разделение времени.***

Lab3 – знать.

**Переключение контекста** — в многозадачных ОС и средах, процесс прекращения выполнения процессором одной задачи (процесса, потока, нити) с сохранением всей необходимой информации и состояния, необходимых для последующего продолжения с прерванного места и восстановления и загрузки состояния задачи, к выполнению которой переходит процессор.

В процедуру переключения контекста входит т. н. планирование задачи — процесс принятия решения к какой задаче передать управление.

При переключении контекста происходит сохранение и восстановление следующей информации:

* Регистровый контекст регистров общего назначения (в том числе флаговый регистр)
* Контекст состояния сопроцессора с плавающей точкой
* Состояние регистров MMX/SSE (x86)
* Состояние сегментных регистров (x86)
* Состояние некоторых управляющих регистров (например, регистр CR3 отвечающий за страничное отображение памяти процесса) (x86)

В ядре ОС с каждым потоком связаны следующие структуры:

* Общая информация pid, tid, uid, gid, euid, egid, …
* Состояние процесса/потока
* Права доступа
* Используемые потоком ресурсы и блокировки
* Счетчики использования ресурсов (например, таймеры использованного процессорного времени)
* Регионы памяти, выделенные процессу

**Совместная или кооперативная многозадачность**

Тип многозадачности, при котором следующая задача выполняется только после того, как текущая задача явно объявит себя готовой отдать процессорное время другим задачам. Как частный случай такое объявление подразумевается при попытке захвата уже занятого объекта mutex (ядро Linux), а также при ожидании поступления следующего сообщения от подсистемы пользовательского интерфейса (Windows версий до 3.x включительно, а также 16-битные приложения в Windows 9x).

Кооперативную многозадачность можно назвать многозадачностью «второй ступени», поскольку она использует более передовые методы, чем простое переключение задач, реализованное многими известными программами (например, DOS Shell из MS-DOS 5.0). При простом переключении активная программа получает все процессорное время, а фоновые приложения полностью замораживаются. При кооперативной многозадачности приложение может захватить фактически столько процессорного времени, сколько оно считает нужным. Все приложения делят процессорное время, периодически передавая управление следующей задаче.

Преимущества кооперативной многозадачности: отсутствие необходимости защищать все разделяемые структуры данных объектами типа критических секций и mutex’ов, что упрощает программирование, особенно перенос кода из однозадачных сред в многозадачные.

Недостатки: неспособность всех приложений работать в случае ошибки в одном из них, приводящей к отсутствию вызова операции «отдать процессорное время». Крайне затрудненная возможность реализации многозадачной архитектуры ввода-вывода в ядре ОС, позволяющей процессору исполнять одну задачу в то время, как другая задача инициировала операцию ввода-вывода и ждет её завершения.

1. ***Прерывания в x86. Инициализация таблицы дескрипторов прерываний IDT.***

Lab8 – знать.

**Прерывание** (англ. *interrupt*) — сигнал, сообщающий процессору о наступлении какого-либо события. При этом выполнение текущей последовательности команд приостанавливается, и управление передаётся обработчику прерывания, который реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.

В зависимости от источника возникновения сигнала прерывания делятся на:

* асинхронные, или внешние (аппаратные) — события, которые исходят от внешних источников (например, периферийных устройств) и могут произойти в любой произвольный момент: сигнал от таймера, сетевой карты или дискового накопителя, нажатие клавиш клавиатуры, движение мыши. Факт возникновения в системе такого прерывания трактуется как *запрос на прерывание* (англ. *Interrupt request, IRQ*);
* синхронные, или внутренние — события в самом процессоре как результат нарушения каких-то условий при исполнении машинного кода: деление на ноль или переполнение стека, обращение к недопустимым адресам памяти или недопустимый код операции;
* программные (частный случай внутреннего прерывания) — инициируются исполнением специальной инструкции в коде программы. Программные прерывания как правило используются для обращения к функциям встроенного программного обеспечения (firmware), драйверов и операционной системы.

**Таблица векторов прерываний** (англ. *Interrupt Descriptor Table, IDT*) используется в архитектуре x86 и служит для определения корректного ответа на прерывания и исключения.

В микропроцессорах 8086/80186 таблица векторов прерываний расположена в первом килобайте памяти начиная с адреса 0000:0000 и содержит 256 векторов прерываний в формате сегмент:смещение. Начиная с процессора 80286, адрес в физической памяти и размер таблицы прерываний определяется 48-битным регистром IDTR.

В **IDT** используются следующие типы прерываний: аппаратные прерывания, программные прерывания и прерывания, зарезервированные процессором, называемые исключениями (первые 32) на случай возникновения некоторых событий (деление на ноль, ошибка трассировки, переполнение).

В реальном режиме элементом IDT является 32-битный FAR-адрес обработчика прерывания.

В защищённом режиме элементом IDT является шлюз прерывания длиной 8 байт, содержащий сегментный (логический) адрес обработчика прерывания, права доступа и др.

В длинном режиме размер дескриптора прерывания увеличен до 16 байт.

В режиме V86 при использовании расширения VME таблица векторов располагается по виртуальному адресу 0000:0000. Без использования этого расширения (и в том случае, если переадресация прерываний запрещена) при возникновении прерывания процессор покидает режим V86 и выполняет обычный обработчик защищённого режима.

## **Инициализация IDT**

Первый этап инициализации выполняется BIOS, перед загрузкой ОС. Второй непосредственно самой операционной системой. Операционной системе доступно изменение некоторых адресов прерываний.

1. ***Обработка прерываний таймера. Вытесняющее разделение времени. Алгоритмы планирования процессорного времени.***

**Вытесняющая, или приоритетная, многозадачность (режим реального времени)**

Вид многозадачности, в котором операционная система сама передает управление от одной выполняемой программы другой в случае завершения операций ввода-вывода, возникновения событий в аппаратуре компьютера, истечения таймеров и квантов времени, или же поступлений тех или иных сигналов от одной программы к другой. В этом виде многозадачности процессор может быть переключен с исполнения одной программы на исполнение другой без всякого пожелания первой программы и буквально между любыми двумя инструкциями в её коде. Распределение процессорного времени осуществляется планировщиком процессов. К тому же каждой задаче может быть назначен пользователем или самой операционной системой определенный приоритет, что обеспечивает гибкое управление распределением процессорного времени между задачами (например, можно снизить приоритет ресурсоёмкой программе, снизив тем самым скорость её работы, но повысив производительность фоновых процессов). Этот вид многозадачности обеспечивает более быстрый отклик на действия пользователя.

Преимущества: возможность полной реализации многозадачного ввода-вывода в ядре ОС, когда ожидание завершения ввода-вывода одной программой позволяет процессору тем временем исполнять другую программу. Сильное повышение надежности системы в целом, в сочетании с использованием защиты памяти — идеал в виде «ни одна программа пользовательского режима не может нарушить работу ОС в целом» становится достижимым хотя бы теоретически, вне вытесняющей многозадачности он не достижим даже в теории. Возможность полного использования многопроцессорных и многоядерных систем.

Недостатки: необходимость особой дисциплины при написании кода, особые требования к его реентрантности, к защите всех разделяемых и глобальных данных объектами типа критических секций и mutex’ов.

**Алгоритмы планирования.**

**1. First-Come, First-Served** (**FCFS -** первым пришел, первым обслужен). Представим себе, что процессы, находящиеся в состоянии готовность, выстроены в очередь. Когда процесс переходит в состояние готовность, он помещается в конец этой очереди. Выбор нового процесса для исполнения осуществляется из начала очереди с удалением оттуда ссылки на него. Очередь подобного типа имеет в программировании специальное наименование – **FIFO**, сокращение от First In, First Out (первым вошел, первым вышел).

Такой алгоритм выбора процесса осуществляет невытесняющее планирование. Процесс занимает столько процессорного времени, сколько ему необходимо. После этого для выполнения выбирается новый процесс из начала очереди.

Преимуществом алгоритма FCFS является легкость его реализации, но в то же время он имеет и много недостатков. Если у нас есть процесс с длительным временем исполнения, то короткие процессы, перешедшие в состояние готовность после длительного процесса, будут очень долго ждать начала выполнения. Поэтому алгоритм FCFS практически неприменим для систем разделения времени – слишком большим получается среднее время отклика в интерактивных процессах.

**2. Round Robin (RR).** Модификацией алгоритма FCFS является алгоритм, получивший название Round Robin (Round Robin – это вид детской карусели). По сути дела, это тот же самый алгоритм, только реализованный в режиме вытесняющего планирования. Можно представить себе все множество готовых процессов организованным циклически – процессы сидят на карусели. Карусель вращается так, что каждый процесс находится около процессора небольшой фиксированный квант времени, обычно 10 – 100 миллисекунд. Пока процесс находится рядом с процессором, он получает процессор в свое распоряжение и может исполняться.

На производительность алгоритма RR сильно влияет величина кванта времени. При очень больших величинах кванта времени, когда каждый процесс успевает завершиться до возникновения прерывания по времени, алгоритм RR вырождается в алгоритм FCFS. При очень малых величинах создается иллюзия того, что каждый из *n* процессов работает на собственном виртуальном процессоре с производительностью *~ 1/n* от производительности реального процессора.

Реализуется такой алгоритм так же, как и предыдущий, с помощью организации процессов, находящихся в состоянии готовность, в очередь FIFO. Планировщик выбирает для очередного исполнения процесс, расположенный в начале очереди, и устанавливает таймер для генерации прерывания по истечении определенного кванта времени. При выполнении процесса возможны два варианта:

* Время непрерывного использования процессора, необходимое процессу, меньше или равно продолжительности кванта времени. Тогда процесс по своей воле освобождает процессор до истечения кванта времени, на исполнение поступает новый процесс из начала очереди, и таймер начинает отсчет кванта заново.
* Продолжительность остатка текущего времени исполнения процесса больше, чем квант времени. Тогда по истечении этого кванта процесс прерывается таймером и помещается в конец очереди процессов, готовых к исполнению, а процессор выделяется для использования процессу, находящемуся в ее начале.

**3. Приоритетное планирование.** При приоритетном планировании каждому процессу присваивается определенное числовое значение – **приоритет**, в соответствии с которым ему выделяется процессор. Процессы с одинаковыми приоритетами планируются в порядке FCFS.

Планирование с использованием приоритетов может быть как вытесняющим, так и невытесняющим. При вытесняющем планировании процесс с более высоким приоритетом, появившийся в очереди готовых процессов, вытесняет исполняющийся процесс с более низким приоритетом. В случае невытесняющего планирования он просто становится в начало очереди готовых процессов.

Главная проблема приоритетного планирования заключается в том, что при ненадлежащем выборе механизма назначения и изменения приоритетов низкоприоритетные процессы могут не запускаться неопределенно долгое время. Решение этой проблемы может быть достигнуто с помощью увеличения со временем значения приоритета процесса, находящегося в состоянии готовность.

**4. Алгоритм SJF (Shortest-Job-First)** представляет собой частный случай приоритетного планирования. Для алгоритма SJF в качестве приоритета выступает оценка продолжительности выполнения процесса. Чем меньше значение этой оценки, тем более высокий приоритет имеет процесс.

**5. Многоуровневые очереди (Multilevel Queue).** Для систем, в которых процессы могут быть легко рассортированы по разным группам, был разработан другой класс алгоритмов планирования. Для каждой группы процессов создается своя очередь процессов, находящихся в состоянии готовность. Этим очередям приписываются фиксированные приоритеты. Например, приоритет очереди системных процессов устанавливается выше, чем приоритет очередей пользовательских процессов. А приоритет очереди процессов, запущенных студентами, ниже, чем для очереди процессов, запущенных преподавателями. Это значит, что ни один пользовательский процесс не будет выбран для исполнения, пока есть хоть один готовый системный процесс, и ни один студенческий процесс не получит в свое распоряжение процессор, если есть процессы преподавателей, готовые к исполнению. Внутри этих очередей для планирования могут применяться самые разные алгоритмы. Так, например, для больших счетных процессов, не требующих взаимодействия с пользователем (фоновых процессов), может использоваться алгоритм FCFS, а для интерактивных процессов – алгоритм RR.

1. ***Обработка вложенных прерываний в x86. Средства синхронизации, состояние гонок, взаимные блокировки. Запрет прерываний, спинлоки, мьютексы, семафоры. Read-Copy-Update.***

Lab8 – знать.

**Состояние гонки** (англ. *race condition*) — ошибка проектирования многопоточной системы или приложения, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода.

**Взаимная блокировка** (англ. *deadlock*) — ситуация в многозадачной среде или СУБД, при которой несколько процессов находятся в состоянии бесконечного ожидания ресурсов, занятых самими этими процессами.

Среди наиболее часто используемых средств синхронизации — сигналы и сообщения, семафоры и мьютексы, каналы (англ. *pipe*), совместно используемая память, сокеты, файлы (в т.ч. проецируемые в память), почтовый ящик.

Внешние прерывания в зависимости от возможности запрета делятся на:

* маскируемые — прерывания, которые можно запрещать установкой соответствующих битов в регистре маскирования прерываний (в x86-процессорах — сбросом флага IF в регистре флагов);
* немаскируемые (англ. *Non-maskable interrupt, NMI*) — обрабатываются всегда, независимо от запретов на другие прерывания. К примеру, такое прерывание может быть вызвано сбоем в микросхеме памяти.

**Спинлок** (англ. Spinlock — циклическая блокировка) — низкоуровневый примитив синхронизации, применяемый в многопроцессорных системах для реализации взаимного исключения. Физически спинлок представляет собой переменную в памяти и реализуется на атомарных операциях, которые должны присутствовать в системе команд процессора. Каждый процессор, желающий получить доступ к разделяемому ресурсу, атомарно записывает условное значение «*занято*» в эту переменную, используя аналог операции swap (в архитектуре x86 — xchg). Если предыдущее значение переменной (возвращаемое командой) было «*свободно*», то считается, что данный процессор получил доступ к ресурсу, в противном случае, процессор возвращается к операции swap и крутится в цикле ожидая, пока спинлок будет освобождён. После работы с разделяемым ресурсом процессор-владелец спинлока должен записать в него условное значение «*свободно*».

**Мьютекс** (англ. *mutex*, от *mutual exclusion* — «взаимное исключение») — одноместный семафор, служащий в программировании для синхронизации одновременно выполняющихся потоков.

Мьютексы — это один из вариантов семафорных механизмов для организации взаимного исключения. Они реализованы во многих ОС, их основное назначение — организация взаимного исключения для потоков из одного и того же или из разных процессов.

Мьютексы — это простейшие двоичные семафоры, которые могут находиться в одном из двух состояний — отмеченном или неотмеченном (открыт и закрыт соответственно). Когда какой-либо поток, принадлежащий любому процессу, становится владельцем объекта mutex, последний переводится в неотмеченное состояние. Если задача освобождает мьютекс, его состояние становится отмеченным.

Задача мьютекса — защита объекта от доступа к нему других потоков, отличных от того, который завладел мьютексом. В каждый конкретный момент только один поток может владеть объектом, защищённым мьютексом. Если другому потоку будет нужен доступ к переменной, защищённой мьютексом, то этот поток засыпает до тех пор, пока мьютекс не будет освобождён.

Цель использования мьютексов — защита данных от повреждения в результате асинхронных изменений (состояние гонки), однако могут порождаться другие проблемы — такие, как взаимная блокировка (клинч).

Мьютекс - одна из реализаций спинлока.

Мьютекс отличается от семафора общего вида тем, что только владеющий им поток может его освободить, т.е. перевести в отмеченное состояние.

**Семафор** — объект, ограничивающий количество потоков, которые могут войти в заданный участок кода. Определение введено Эдсгером Дейкстрой. Семафоры используются при передаче данных через разделяемую память.

Read – Copy – Update (RCU) – техника синхронизации, предназначенная для «почти read-only», то есть редко изменяемых, структур данных. Типичными примерами такой структуры являются map и set – в них большинство операций является поиском, то есть чтением данных. Считается, что для типичного map'а более 90% вызываемых операций — это поиск по ключу, поэтому важно, чтобы операция поиска была наиболее быстрой; синхронизация поиска в принципе не нужна — читатели при отсутствии писателей могут работать параллельно. RCU обеспечивает наименьшие накладные расходы как раз для read-операций.
Откуда взялось название Read – Copy – Update? Первоначально идея была очень проста: есть некоторая редко изменяемая структура данных. Если нам требуется *изменить* её, то мы делаем её *копию* и производим изменение — добавление или удаление данных — именно в копии. При этом параллельные читатели работают с первоначальной, не измененной структурой. В некоторый безопасный момент времени, когда нет читателей, мы можем подменить структуру данных на измененную копию. В результате все последующие читатели будут видеть изменения, произведенные писателем.

1. ***Виртуальная память. Модели организации оперативной памяти. Сегментная и страничная трансляция x86. Таблицы трансляции.***

**Виртуальная память** — технология управления памятью ЭВМ, разработанная для многозадачных операционных систем. При использовании данной технологии для каждой программы используются независимые схемы адресации памяти, отображающиеся тем или иным способом на физические адреса в памяти ЭВМ. Позволяет увеличить эффективность использования памяти несколькими одновременно работающими программами, организовав множество независимых адресных пространств, и обеспечить защиту памяти между различными приложениями. Также позволяет программисту использовать больше памяти, чем установлено в компьютере, за счет откачки неиспользуемых страниц на вторичное хранилище.

При использовании виртуальной памяти упрощается программирование, так как программисту больше не нужно учитывать ограниченность памяти, или согласовывать использование памяти с другими приложениями. Для программы выглядит доступным и непрерывным все допустимое адресное пространство, вне зависимости от наличия в ЭВМ соответствующего объёма ОЗУ.

Применение механизма виртуальной памяти позволяет:

* упростить адресацию памяти клиентским программным обеспечением;
* рационально управлять оперативной памятью компьютера (хранить в ней только активно используемые области памяти);
* изолировать процессы друг от друга (процесс полагает, что монопольно владеет всей памятью).

*Одиночное непрерывное распределение*

Данная модель распределения оперативной памяти является одной из самых простых и основывается на том, что все адресное пространство подразделяется на два компонента. В одной части памяти располагается и функционирует операционная система, а другая часть выделяется для выполнения прикладных процессов.

*Распределение неперемещаемыми разделами*

Данная модель строится по следующим принципам. Опять же, все адресное пространство оперативной памяти делится на две части. Одна часть отводится под операционную систему, все оставшееся пространство отводится под работу прикладных процессов, причем это пространство заблаговременно делится на N частей (назовем их ***разделами***), каждая из которых в общем случае имеет произвольный фиксированный размер. Эта настройка происходит на уровне операционной системы. Соответственно, очередь прикладных процессов разделяется по этим разделам.

*Распределение перемещаемыми разделами*

Данная модель распределения разрешает загрузку произвольного (нефиксированного) числа процессов в оперативную память, и под каждый процесс отводится раздел необходимого размера. Соответственно, система допускает перемещение раздела, а, следовательно, и процесса. Такой подход позволяет избавиться от фрагментации.

*Страничное распределение*

Данная модель основывается на том, что все адресное пространство может быть представлено совокупностью блоков фиксированного размера, которые называются ***страницами***. Есть ***виртуальное адресное пространство*** — это то пространство, в котором оперирует программа, и ***физическое адресное пространство*** — это то пространство, которое есть в наличии у компьютера. Соответственно, при страничном распределении памяти существуют программно-аппаратные средства, позволяющие устанавливать соответствие между виртуальными и физическими страницами. Механизм преобразования виртуального адреса в физический обсуждался выше, он достаточно прост: берется номер виртуальной страницы и заменяется соответствующим номером физической страницы. Также отмечалось, что для этих целей используется т.н. ***таблица страниц***, которая целиком является аппаратной, что на самом деле является большим упрощением.

*Сегментное распределение*

Данная модель представляет каждый процесс в виде совокупности сегментов, каждый из которых может иметь свой размер. Каждый из сегментов может иметь собственную функциональность: существуют сегменты кода, сегменты статических данных, сегмент стека и т.д. Для организации работы с сегментами может использоваться некоторая таблица, в которой хранится информация о каждом сегменте (его номер, размер и пр.). Тогда виртуальный адрес может быть проинтерпретирован, как номер сегмента и величина смещения в нем.

*Сегментно-страничное распределение*

Эта модель рассматривает виртуальный адрес, как номер сегмента и смещение в нем. Имеется также аппаратная таблица сегментов, посредством которой из виртуального адреса получается т.н. ***линейный адрес***, который, в свою очередь, представляется в виде номера страницы и величины смещения в ней. А затем, используя таблицу страниц, получается непосредственно физический адрес.

**Сегментная адресация памяти** — схема логической адресации памяти компьютера в архитектуре x86. Линейный адрес конкретной ячейки памяти, который в некоторых режимах работы процессора будет совпадать с физическим адресом, делится на две части: *сегмент* и *смещение*. *Сегментом* называется условно выделенная область адресного пространства определённого размера, а *смещением* — адрес ячейки памяти относительно начала сегмента. *Базой сегмента* называется линейный адрес (адрес относительно всего объёма памяти), который указывает на начало сегмента в адресном пространстве. В результате получается *сегментный (логический) адрес*, который соответствует линейному адресу *база сегмента*+*смещение* и который выставляется процессором на шину адреса. *Селектором* называется число (в x86 — 16-битное), однозначно определяющее сегмент. Селектор загружается в сегментные регистры. В реальном и защищённом режимах x86-процессора функционирование сегментной адресации отличается.

##### *Страничная трансляция*

Механизм сегментации обеспечивает превосходную защиту, но он не очень удобен для реализации виртуальной памяти (подкачки). В дескрипторе сегмента есть бит присутствия, по нему процессор определяет, находится ли данный сегмент в физической памяти или он находится на внешнем запоминающем устройстве (на винчестере). В последнем случае генерируется исключение #11, обработчик которого может подгрузить сегмент в память. Неудобство заключается в том, что различные сегменты могут иметь различную длину. Этого можно избежать, если механизм подкачки реализовывать на основе страничного преобразования. Особенностью этого преобразования является то, что процессор в этом случае оперирует с блоками физической памяти равной длины (4Кбайт) - *страницами*. Страницы не имеют непосредственного отношения к логической структуре программы.

Страничное преобразование действует только в защищенном режиме и включается установкой в 1 бита PG в регистре CR0.

В страничном преобразовании участвуют два типа структур: каталоги таблиц (Page Directory) и таблицы страниц (Page Table). Эти структуры состоят из 1024 32-битных элементов. Элементы содержат старшие 20 бит физического адреса адресуемых объектов. Элементы таблицы страниц (Page Table Entry - PTE) адресуют страницы, а элементы каталога таблиц (Page Directory Entry - PDE) адресуют таблицы страниц. Старшие 20 бит физического адреса каталога таблиц хранится в регистре CR3 (Page Directory Base Register - PDBR). (Это единственный регистр процессора, который содержит физический адрес памяти). Все структуры выравниваются по границе страницы.



В процессе страничной трансляции адресов полученный линейный адрес разбивается на три части. Старшие десять бит (Directory) линейного адреса являются индексом элемента из каталога таблиц. По этому элементу определяется физический адрес таблицы страниц. Биты 21-12 (Table) линейного адреса выбирают элемент из этой таблицы страниц. Выбранный элемент определяет физический адрес страницы. Младшие 12 бит (Offset) линейного адреса определяют смещение от начала страницы.

Страницы начинаются на границах 4Кбайт областей памяти, поэтому младшие 12 бит адреса страницы всегда равны нулю. В каталоге таблиц элементы хранят физические адреса таблиц страниц. В таблице страниц элементы хранят физические адреса самих страниц.

1. ***Переключение между режимами работы процессора. Прерывания и системные вызовы. Выполнение системных вызовов без переключения в привилегированный режим.***

**Защищенный режим (Protected Mode)**

Основным режимом работы микропроцессора является защищенный режим. Ключевыми особенностями защищенного режима являются: виртуальное адресное пространство, защита и многозадачность.

В защищенном режиме программа оперирует адресами, которые могут относиться к физически отсутствующим ячейкам памяти, поэтому такое адресное пространство называется *виртуальным*. Размер виртуального адресного пространства программы может превышать емкость физической памяти и достигать 64Тбайт. Для адресации виртуального адресного пространства используется сегментированная модель, в которой адрес состоит из двух элементов: селектора сегмента и смещения внутри сегмента. С каждым сегментом связана особая структура, хранящая информацию о нем, - дескриптор. Кроме "виртуализации" памяти на уровне сегментов существует возможность "виртуализации" памяти при помощи страниц - *страничная трансляция*. Страничная трансляция предоставляет удобные средства для реализации в операционной системе функций подкачки, а кроме того в процессорах P6+ обеспечивает 36-битную физическую адресацию памяти (64Гбайт).

Встроенные средства переключения задач обеспечивают *многозадачность* в защищенном режиме. Среда задачи состоит из содержимого регистров МП и всего кода с данными в пространстве памяти. Микропроцессор способен быстро переключаться из одной среды выполнения в другую, имитируя параллельную работу нескольких задач. Для некоторых задач может эмулироваться управление памятью как у процессора 8086. Такое состояние задачи называется *режимом виртуального 8086 (Virtual 8086 Mode)*. О пребывании задачи в таком состоянии сигнализирует бит VM в регистре флагов. При этом задачи виртуального МП 8086 изолированы и защищены, как от друг друга, так и от обычных задач защищенного режима.

*Защита* задач обеспечивается следующими средствами: контроль пределов сегментов, контроль типов сегментов, контроль привилегий, привилегированные инструкции и защита на уровне страниц. Контроль пределов и типов сегментов обеспечивает целостность сегментов кода и данных. Программа не имеет права обращаться к виртуальной памяти, выходящей за предел того или иного сегмента. Программа не имеет права обращаться к сегменту данных как к коду и наоборот. Архитектура защиты микропроцессора обеспечивает 4 иерархических уровня привилегий, что позволяет ограничить задаче доступ к отдельным сегментам в зависимости от ее текущих привилегий. Кроме того, текущий уровень привилегий задачи влияет на возможность выполнения тех или иных специфических команд (привилегированных инструкций). Функции страничной трансляции, впервые появившиеся в МП Intel386, обеспечивают дополнительные механизмы защиты на уровне страниц.

**Реальный режим (Real Mode)**

В реальном режиме микропроцессор работает как очень быстрый 8086 с возможностью использования 32-битных расширений. Механизм адресации, размеры памяти и обработка прерываний (с их последовательными ограничениями) МП Intel386 в реальном режиме полностью совпадают с аналогичными функциями МП 8086. В отличие от 8086 микропроцессоры 286+ в определенных ситуациях генерируют исключения, например, при превышении предела сегмента, который для всех сегментов в реальном режиме - 0FFFFh.

Имеется две фиксированные области в памяти, которые резервируются в режиме реальной адресации:

* область инициализации системы
* область таблицы прерываний

Ячейки от 00000h до 003FFH резервируются для векторов прерываний. Каждое из 256 возможных прерываний имеет зарезервированный 4-байтовый адрес перехода. Ячейки от FFFFFFF0H до FFFFFFFFH резервируются для инициализации системы.

**Режим системного управления (System Management Mode)**

Режим системного управления предназначен для выполнения некоторых действий с возможностью их полной изоляции от прикладного программного обеспечения и даже операционной системы. Переход в этот режим возможен только аппаратно. Когда процессор находится в режиме SMM, он выставляет сигнал SMIACT#. Этот сигнал может служить для включения выделенной области физической памяти (System Management RAM), так что память SMRAM можно сделать доступной только для этого режима. При входе в режим SMM процессор сохраняет свой контекст в SMRAM (контекст сопроцессора не сохраняется) по адресу SMM Base и передает управление процедуре, называемой обработчиком System Management Interrupt, по адресу SMM Base+8000h (по умолчанию SMM Base содержит значение 30000h). Состояние процессора в этот момент точно определено: EFLAGS обнулен (кроме зарезервированных битов), сегментные регистры содержат селектор 0000, базы сегментов установлены в 00000000, пределы - 0FFFFFFFFh.

Следует отметить, что в режиме SMM не предусмотрена работа с прерываниями и особыми случаями: прерывания по IRQ и SMI# замаскированы, пошаговые ловушки и точки останова отключены, обработка прерывания по NMI откладывается до выхода из режима SMM. Если необходимо обеспечить работу с прерываниями или особыми случаями, то надо инициализировать IDT и разрешить прерывания, выставив флаг IF в регистре EFLAGS. Прерывания по NMI будут разблокированы автоматически после первой же команды IRET.

При возврате из SMM (по инструкции RSM) процессор восстанавливает свой контекст из SMRAM. Обработчик может программно внести изменения в образ контекста процессора, тогда процессор перейдет не в то состояние, в котором произошло SMI. Если SMI было получено во время выполнения инструкции HLT, то дальнейшие действия при выходе из SMM определяются значением поля "Auto HALT Restart": процессор может снова вернуться к инструкции останова или перейти к выполнению следующей команды. Если SMI произошло при выполнении инструкции ввода-вывода, то в зависимости от значения поля "I/O Instruction Restart" возможен рестарт инструкции ввода вывода.

Эти особенности режима системного управления позволяют использовать его для реализации системы управления энергосбережением компьютера или функций безопасности и контроля доступа.

**Переключение между режимами**



Прерывания используются для обработки асинхронных внешних к процессору событий. Исключения обрабатывают нарушения, обнаруженные самим процессором в ходе выполнения инструкций. Прерывания: маскируемые (приходят на контакт INTR), немаскируемые (приходят на контакт NMI). Исключения: программные прерывания (int n), отказы (fault) – обнаруживаются до или во время выполнения инструкции, возможна повторная попытка выполнения, ловушки – обнаруживаются сразу после инструкции, вызвавшей исключение, прерывания (abort) – исключения, повлекшие серьезные ошибки, без возможности повторного выполнения программы.

## *Системные вызовы*

Код, работающий в пользовательском режиме, не может запускать какой-либо код, который расположен в области памяти супервизора, или получать доступ к области супервизора (смотрите флаги записей таблицы страниц) и не может запускать код, в котором используются привилегированные команды, например, *hlt*. Поэтому в большинстве ядер предоставляется интерфейс, который можно пользоваться в обычных функциях. Обращение к ядру через этот интерфейс называется "системным вызовом".

Способ, с помощью которого в архитектуре x86 реализуются системные вызовы, представляет собой работу с программными прерываниями. В пользовательской программе задается регистр, в котором указывается, какие системные функции следует выполнять, а затем в других регистрах настраиваются параметры. Затем будет выполнено программное прерывание для определенного вектора - в linux используется 0x80. Программное прерывание вызывает изменение режима — происходит переход в кольцо 0; в ядре должен быть обработчик для этого вектора прерывания и, соответственно, должна быть выполнена диспетчеризация этого системного вызова.

1. ***Аппаратно-программные средства поддержки мультипрограммного режима – система прерываний, защита памяти, привилегированный режим.***

Для корректного мультипрограммирования система должна обеспечивать эксклюзивное владение программ выделенными им участками памяти. Если возникает задача обеспечения множественного доступа к памяти, то это должно осуществляться с согласия владельца этой памятью. Итак, первое требование к системе — это наличие т.н. ***аппарата защиты памяти***. Сразу отметим, что режим защиты памяти нельзя делать чисто программным способом, поскольку если данный режим будет обеспечивать операционная система (т.е. каждый раз сравнивать получаемый исполнительный адрес, не вышел ли он за границы дозволенного программе диапазона адресов), то производительность вычислительной системы в целом будет крайне низкой.

Реализация аппарата защиты памяти может быть достаточно простой: в процессоре могут быть специальные регистры (***регистры границ***), в которых устанавливаются границы диапазона доступных для исполняемой задачи адресов оперативной памяти. Соответственно, когда устройство управления в центральном процессоре вычисляет очередной исполнительный адрес (это может быть адрес следующей команды или же адрес необходимого операнда), **автоматически** проверяется, принадлежит ли полученный адрес заданному диапазону. Если адрес принадлежит диапазону, то продолжается обработка задачи, иначе же в системе возникает прерывание (т.н. ***прерывание по защите памяти***).

Система должна каким-то способом ранжировать и в соответствии с этим ранжированием ограничивать доступ пользователей различных категорий к машинным командам. Решением стала **аппаратная** возможность работы центрального процессора в двух режимах: в ***режиме работы операционной системы*** (или ***привилегированном режиме***, или ***режиме супервизора***) и в ***пользовательском режиме*** (или ***непривилегированном режиме***, еще раньше использовался термин ***математического режима***). В режиме работы ОС процессор исполняет абсолютно все команды, представленные в программе. Если же программа исполняется в пользовательском режиме, то ей доступны для исполнения лишь некоторое подмножество машинных команд (если же при обработке такой программы встретится недопустимая команда, то в системе возникнет прерывание по запрещенной команде).

Тогда возникает вопрос, что должна делать программа, обрабатываемая в пользовательском режиме, для печати, например, своих данных. Решений здесь может быть достаточно много, одним из которых может быть наличие в системе специальных команд, интерпретируемых как обращения к операционной системе.

***Прерыванием*** называется событие в компьютере, при возникновении которого в процессоре происходит предопределенная последовательность действий. Состав прерываний — множество разновидностей событий, на возникновение которых предусмотрена стандартная реакция центрального процессора, — фиксирован и определяется конструктивно при разработке компьютера. **Аппарат прерываний** компьютера позволяет организовывать стандартную обработку всех прерываний, возникающих при функционировании вычислительной системы. Традиционно прерывания разделяются на две группы: **внутренние прерывания** и **внешние прерывания**.

1. ***Файловые системы. Основные задачи файловых систем.***

Под ***файловой системой*** (ФС) мы будем понимать часть операционной системы, представляющую собой совокупность организованных наборов данных, хранящихся на внешних запоминающих устройствах, и программных средств, гарантирующих именованный доступ к этим данным и их защиту. ***Файловая система*** — это компонент операционной системы, обеспечивающий **корректный именованный доступ** к данным пользователя. Данные в файловой системе представляются в виде файлов, каждый из которых имеет имя. Главными словами в определении файловой системы являются **именованный доступ** и **корректная работа**. Последнее означает, что файловая система обеспечивает корректное управление свободным и занятым пространством на ВЗУ (заметим, что не обязательно на физическом устройстве: в качестве ВЗУ может выступить и виртуальное устройство), а также защиту от несанкционированного доступа к информации. Большинство современных файловых систем обеспечивают корректную организацию распределенного доступа к одному и тому же файлу (когда с ним могут работать два и более пользователя).

Основные функции любой файловой системы нацелены на решение следующих задач:

* именование файлов;
* программный интерфейс работы с файлами для приложений;
* отображения логической модели файловой системы на физическую организацию хранилища данных;
* организация устойчивости файловой системы к сбоям питания, ошибкам аппаратных и программных средств;
* содержание параметров файла, необходимых для правильного его взаимодействия с другими объектами системы (ядро, приложения и пр.).

В многопользовательских системах появляется ещё одна задача: защита файлов одного пользователя от несанкционированного доступа другого пользователя, а также обеспечение совместной работы с файлами, к примеру, при открытии файла одним из пользователей, для других этот же файл временно будет доступен в режиме «только чтение».