4.6 Оперативная память. Способы адресации. Виртуальная память. Страничная организация памяти. Файлы подкачки, алгоритмы выгрузки страниц.

Операти́вная па́мять, (ОЗУ — оперативное запоминающее устройство) — в информатике — память, это часть системы памяти ЭВМ, в которую процессор может обратиться за одну операцию (Jump, Move и пр.). Предназначена для временного хранения данных и команд, необходимых процессору для выполнения им операций. Оперативная память передаёт процессору команды и данные непосредственно, либо через кэш-память. Каждая ячейка оперативной памяти имеет свой индивидуальный адрес.

В современных вычислительных устройствах, оперативная память выполнена по технологии *динамической памяти с произвольным доступом* (англ. *dynamic random access memory, DRAM*). Понятие памяти с произвольным доступом предполагает, что текущее обращение к памяти не учитывает порядок предыдущих операций и расположения данных в ней. ОЗУ может изготавливаться как отдельный блок, или входить в конструкцию однокристальной ЭВМ или микроконтроллера.

Ячейка памяти — часть запоминающего устройства ЭВМ для хранения 1 машинного слова (числа) или его части (например, 1 байта). Общее число ячеек памяти всех запоминающих устройств определяет ёмкость памяти ЭВМ.

DRAM (Dynamic Random Access Memory) — один из видов компьютерной памяти с произвольным доступом (RAM), наиболее широко используемый в качестве ОЗУ современных компьютеров.

Конструктивно память DRAM состоит из *«ячеек»* размером в 1 или 4 бит, в каждой из которых можно хранить определённый объём данных. Совокупность «ячеек» такой памяти образуют условный *«прямоугольник»*, состоящий из определённого количества *строк и столбцов*. Один такой «прямоугольник» называется *страницей*, а совокупность страниц называется *банком*. Весь набор «ячеек» условно делится на несколько областей.

Физическое представление

В современных компьютерах физически память представляет собой электрическую плату — модуль, на котором расположены микросхемы памяти и разъём, необходимый для подключения модуля к материнской плате. Роль «ячеек» играют конденсаторы и транзисторы, расположенные внутри микросхем памяти. Конденсаторы заряжаются в случае, когда в «ячейку» заносится единичный бит, либо разряжаются в случае, когда в «ячейку» заносится нулевой бит. Транзисторы необходимы для удержания заряда внутри конденсатора. При отсутствии подачи электроэнергии к оперативной памяти, происходит разряжение конденсаторов, и память опустошается. Это динамическое изменение заряда конденсатора является основополагающим принципом работы памяти типа DRAM. Элементом памяти типа DRAM является чувствительный усилитель (англ. *sense amp*), подключенный к каждому из столбцов «прямоугольника». Он, реагируя на слабый поток электронов, устремившихся через открытые транзисторы с обкладок конденсаторов, считывает всю страницу целиком. Именно страница является минимальной порцией обмена с динамической памятью, потому что обмен данными с отдельно взятой ячейкой невозможен.

Характеристики памяти DRAM

Основными характеристиками DRAM являются тайминги и рабочая частота. Для обращения к ячейке, контроллер задаёт номер банка, номер страницы в нём, номер строки и номер столбца, на

все запросы тратится время, помимо этого довольно большая затрата уходит на открытие и закрытие банка после самой операции. На каждое действие требуется время, называемое *таймингом*. Основными таймингами DRAM являются: задержка между подачей номера строки и номера столбца, называемая временем полного доступа (англ. *RAS to CAS delay*), задержка между подачей номера столбца и получением содержимого ячейки, называемая временем рабочего цикла (англ. *CAS delay*), задержка между чтением последней ячейки и подачей номера новой строки (англ. *RAS precharge*). Тайминги измеряются в наносекундах, и чем меньше величина этих таймингов, тем быстрее работает оперативная память. Рабочая частота измеряется в мегагерцах, и увеличение рабочей частоты памяти приводит к увеличению её быстродействия.

Регенерация

В отличие от *статической* памяти типа SRAM (англ. *static random access memory*), которая является конструктивно более сложным и более дорогим типом памяти RAM и используется в основном в кэш-памяти, память DRAM изготавливается на основе конденсаторов небольшой ёмкости, которые быстро теряют заряд, поэтому информацию приходится обновлять через определённые периоды времени, во избежание потерь данных. Этот процесс называется регенерацией памяти. Он реализуется специальным контроллером, установленным на материнской плате или на кристалле центрального процессора. На протяжении времени, называемого *шагом регенерации*, в DRAM перезаписывается целая строка «ячеек», и через *8-64 мс* обновляются все строки памяти.

Процесс регенерации памяти в *классическом варианте* существенно «тормозит» работу системы, поскольку в это время обмен данными с памятью невозможен. Регенерация, основанная на обычном *переборе строк*, не применяется в современных типах DRAM. Существует несколько более экономичных вариантов этого процесса — *расширенный, пакетный, распределенный*; наиболее экономичной является *скрытая регенерация*.

Из новых технологий регенерации можно выделить тип регенерации PASR (англ. *Partial Array Self Refresh*), применяемый компанией Samsung в чипах памяти SDRAM с низким уровнем энергопотребления. Регенерация «ячеек» выполняется только в период ожидания в *тех* банках памяти, в которых *имеются данные*. Параллельно с этой технологией реализуется метод TCSR (англ. *Temperature Compensated Self Refresh*), который предназначен для регулировки скорости процесса регенерации в зависимости от рабочей температуры.

Типы DRAM

На протяжении долгого времени разработчиками создавались различные типы памяти. Они обладали разными характеристиками, в них были использованы разные технические решения. Основной движущей силой развития памяти было развитие ЭВМ и центральных процессоров. Постоянно требовалось увеличение быстродействия и объёма оперативной памяти.

Страничная память

Страничная память (англ. *page mode DRAM, PM DRAM*) являлась одним из первых типов выпускаемой компьютерной оперативной памяти. Память такого типа выпускалась в начале 90-х годов, но с ростом производительности центральных процессоров и ресурсоёмкости приложений требовалось увеличивать не только объём памяти, но и скорость её работы.

Быстрая страничная память

Быстрая страничная память (англ. *fast page mode DRAM, FPM DRAM*) появилась в 1995 году. Принципиально новых изменений память не претерпела, а увеличение скорости работы достигалось

путём повышенной нагрузки на аппаратную часть памяти. Данный тип памяти в основном применялся для компьютеров с процессорами Intel 486 или аналогичных процессоров других фирм. Память могла работать на частотах 25 МГц и 33 МГц с временем полного доступа 70 нс и 60 нс и с временем рабочего цикла 40 нс и 35 нс соотвественно.

Память с усовершенствованным выходом

Но с появлением процессоров Intel Pentium II память FPM DRAM оказалась совершенно неэффективной. Поэтому следующим шагом стала память с усовершенствованным выходом (англ. *extended data out DRAM, EDO DRAM*). Эта память появилась на рынке в 1996 году и стала активно использоваться на компьютерах с процессорами Intel Pentium и выше. Её производительность оказалась на 10-15 % выше по сравнению с памятью типа FPM DRAM. Её рабочая частота была 40 МГц и 50 МГц, соответственно, время полного доступа — 60 нс и 50 нс, а время рабочего цикла — 25 нс и 20 нс. Эта память содержит регистр-защелку (data latch) выходных данных, что обеспечивает некоторую конвейеризацию работы для повышения производительности при чтении.

Синхронная DRAM

В связи с выпуском новых процессоров и постепенным увеличением частоты системной шины, стабильность работы памяти типа EDO DRAM стала заметно падать. Ей на смену пришла синхронная память (англ. *synchronous DRAM, SDRAM*). Новыми особенностями этого типа памяти являлись использование тактового генератора для синхронизации всех сигналов и использование конвейерной обработки информации. Также память надёжно работала на более высоких частотах системной шины (100 МГц и выше). Недостатками данного типа памяти являлась его высокая цена, а также его несовместимость со многими чипсетами и материнскими платами в силу своих новых конструктивных особенностей. Рабочие частоты этого типа памяти могли равняться 66 МГц, 100 МГц или 133 МГц, время полного доступа — 40 нс и 30 нс, а время рабочего цикла — 10 нс и 7.5 нс.

Пакетная EDO RAM

Пакетная память EDO RAM (англ. *burst extended data output DRAM, BEDO DRAM*) стала дешёвой альтернативой памяти типа SDRAM. Основанная на памяти EDO DRAM, её ключевой особенностью являлась технология поблочного чтения данных (блок данных читался за один такт), что сделало её работу быстрее, чем у памяти типа SDRAM. Однако невозможность работать на частоте системной шины более 66 МГц не позволила данному типу памяти стать популярным.

Video RAM

Специальный тип оперативной памяти Video RAM (VRAM) был разработан на основе памяти типа SDRAM для использования в видеоплатах. Он позволял обеспечить непрерывный поток данных в процессе обновления изображения, что было необходимо для реализации изображений высокого качества. На основе памяти типа VRAM, появилась спецификация памяти типа Windows RAM (WRAM), иногда её ошибочно связывают с операционными системами семейства Windows. Её производительность стала на 25 % выше, чем у оригинальной памяти типа SDRAM, благодаря некоторым техническим изменениям.

DDR SDRAM

По сравнению с обычной памятью типа SDRAM, в памяти SDRAM с удвоенной скоростью передачи данных (англ. *double data rate SDRAM, DDR SDRAM, или SDRAM II*) была вдвое увеличена пропускная способность. Первоначально память такого типа применялась в видеоплатах, но

позднее появилась поддержка DDR SDRAM со стороны чипсетов. Она работает на частотах в 100 МГц и 133 МГц, её время полного доступа — 30 нс и 22.5 нс, а время рабочего цикла — 5 нс и 3.75 нс.

Direct RDRAM, или Direct Rambus DRAM

Тип памяти RDRAM является разработкой компании Rambus. Высокое быстродействие этой памяти достигается рядом особенностей, не встречающихся в других типах памяти. Первоначальная очень высокая стоимость памяти RDRAM привела к тому, что производители мощных компьютеров предпочли менее производительную, зато более дешевую память DDR SDRAM. Рабочие частоты памяти — 400 МГц, 600 МГц и 800 МГц, время полного доступа — до 30 нс, время рабочего цикла — до 2.5 нс.

DDR2 SDRAM

Конструктивно новый тип оперативной памяти DDR2 SDRAM был выпущен в 2004 году. Основываясь на технологии DDR SDRAM, этот тип памяти за счет технических изменений показывает более высокое быстродействие и предназначен для использования на современных компьютерах. Память может работать на частотах в 200 МГц, 266 МГц, 333 МГц и 400 МГц. Время полного доступа — 25 нс, 11.25 нс, 9 нс, 7.5 нс. Время рабочего цикла — 5 нс, 3.75 нс, 3 нс, 3.5 нс.

Структура памяти любой машины является многоуровневой. Можно выделить уровни регистровой, оперативной и внешней памяти. Уровень оперативной памяти имеет первостепенное значение, т. к. ни один процесс в системе не может развиваться без такого ресурса как ОП. В составе любой ОС имеется распределитель ОП. Алгоритмы распределения ОП весьма разнообразны в силу следующих обстоятельств:

1) оперативная память, несмотря на развитие схемотехники, остается дорогим ресурсом, поэтому необходимо заботиться о ее эффективной с точки зрения ОС загрузке;

2) с помощью распределителя ОП (и, возможно, других уровней памяти и других ресурсов) пользователь должен получить виртуальную память, характеристики и способы работы с которой отличаются от реально существующих (т.е. пользователи должны работать с ОП не на физическом уровне, а на более высоком);

3) ОП является особым ресурсом, поскольку ее распределение должно происходить не изолированно, а согласованно с распределением ЦП.

С точки зрения связи с распределением ЦП различают два уровня распределения ОП. На уровне долгосрочного планирования ОП распределяется статически таким образом, что каждому заданию отводится область памяти для исполнения программ и размещения данных, называемая *зоной*. Внутри зон память распределяется динамически на уровне краткосрочного планирования.

В любой момент времени все пространство ОП представляет собой чередующуюся последовательность из занятых (распределенных) и свободных (не распределенных) участков памяти. Свободные участки ОП принято называть *дырами*.

Задача распределения ОП как на уровне долгосрочного, так и краткосрочного планирования распадается на три взаимосвязанных задачи: *учета*, *выделения* и *возврата*, которые заключаются в следующем:

1) Учету подвергаются по определенным правилам либо только дыры, либо, более часто, и дыры и занятые участки.

2) Решение задачи выделения происходит всякий раз, когда приходит запрос на выделение некоторой области памяти для отдельного задания или процесса в составе задания. В

наиболее простых случаях выделение требуемого непрерывного участка памяти происходит из резерва свободной памяти, представленной некоторой совокупностью дыр. По определенным правилам выбирается дыра, внутри которой и выделяется участок памяти требуемого размера. Если все дыры имеют размер, меньший требуемого непрерывного участка памяти, то используется более сложный алгоритм. Тогда, если это допустимо в системе, распределение обеспечивается за счет переупорядочивания (перемещения) занятых участков и дыр с целью получить дыру большего размера. В других случаях память выделяется непрерывно так, что выделяемая область частично захватывает и дыры, и смежные занятые участки. Для обеспечения сохранности информации, находящейся в занятых участках распределяемой памяти, осуществляется ее переписывание на внешнюю память. В последующем эта информация возвращается либо в то же место ОП, либо в другое (тогда происходит перенастройка адресов средствами ОС) без нарушения логики работы других процессов.

3) Задача возврата решается при освобождении занятых участков по требованию самих процессов или без их ведома (в случае необходимости выделения памяти за счет занятых участков).

При решении указанных задач стремятся:

1) Минимизировать время их решения. При этом улучшение временных параметров за счет некоторого алгоритма в рамках одной задачи нередко приводит к их ухудшению при решении других задач. В качестве примера можно рассмотреть следующую ситуацию: формирование неупорядоченного списка дыр при решении задачи учета требует меньше времени, чем упорядоченного, но поиск самой подходящей дыры на этапе выделения памяти в этом случае происходит существенно медленнее.

2) Минимизировать потери памяти путем борьбы с фрагментацией. На это направлено статическое разбиение памяти на зоны до начала работы ОС (каждому заданию – своя зона). Более гибкий алгоритм предполагает, что количество и размер зон могут динамически меняться (на этапе долгосрочного планирования очередного задания). Предполагается также вмешательство пользователя с помощью средств общения с системой.

Независимо от того, какой из двух алгоритмов распределения памяти применяется – самая подходящая или первая подходящая дыра, распределение и освобождения памяти требует выполнения следующих действий:

1) найти свободную область не меньше требуемого размера;
2) поделить ее на две части – распределяемую и свободную;
3) при освобождении, если возможно, объединить освободившуюся область с любой свободной смежной областью.

Сегментная модель памяти

Когда-то давно, на заре рождения компьютерной техники, оперативная память была очень маленькой и для ее адресации использовались 2 байта (так называемое "слово"). Такой подход позволял адресовать 64 Кб памяти, и адресация была линейной - для указания адреса использовалось одно-единственное число. Позже, с усовершенствованием техники, производители поняли, что имеется возможность поддерживать бОльшие объемы памяти, но для этого нужно сделать размер адреса больше. Для совместимости с уже написанным программным обеспечением было решено сделать так: адресация теперь двухкомпонентная (сегмент и смещение), каждая из которых 16-битная, а старые программы как использовали одну 16-битную компоненту и ничего не знают о сегментах, так и продолжают работать. Физический адрес в такой модели вычисляется так: 16 * <сегментный адрес> + <смещение>, таким образом, модель позволяла адресовать от 0 до 16 * 65 535 + 65 535 = 1 114 095, т. е. 1 114 096 байт, что чуть больше 1 Мб. Одно важное свойство сегментной адресации: один и тот же физический адрес можно представить различными комбинациями сегмент + смещение. При этом смещения могут указываться в процессорных

командах как непосредственно, так и с помощью регистров, а сегменты непосредственно указываться не могут: для указания сегмента используются специальные 16-битные сегментные регистры. Поначалу их было 4: CS, DS, SS, ES (сегмент команд, сегмент данных, сегмент стека, дополнительный сегмент). Появившиеся позже 2 дополнительных сегментных регистра мы рассмотрим позже. Большинство команд процессора по умолчанию используют тот или иной сегмент - например, команды чтения и записи данных используют сегмент данных, стековые команды (push, pop, pusha, popa, pushf, popf) используют SS. В некоторых командах есть возможность использовать другой сегмент, нежели сегмент по умолчанию - для этого используется специальная команда-префикс. Содержимое сегментных регистров также можно менять (кроме CS - изменить содержимое CS равнозначно дальнему переходу, поэтому для этого и используются дальние переходы).

Такая сегментная модель давно устарела - с появлением 32-битных процессоров - и пригодится только при разработке программ DOS-режима, а также операционных систем (например, загрузчик ОС), или менеджеров загрузки операционных систем. Поэтому заостряться сильно на этой модели не будем.

С появлением 32-битных процессоров необходимость в такой модели отпала и современные 32-битные программы, как правило, с сегментами и сегментными регистрами не работают вообще. Но благодаря принципу совместимости, которого придерживается Intel как "законодатель" архитектур, сегментную модель было решено оставить. Следует отметить, что с появлением 32-битных процессоров сегментная часть адреса осталась 16-битной, а вот смещение стало 32-битным, а также все регистры, кроме сегментных, стали 32-битными. Изменился и принцип вычисления физического адреса: сегмент ни на что не умножается, просто есть специальная таблица, адрес и размер которой "знает" процессор, эта таблица содержит для каждого сегмента стартовый адрес. Так вот, само понятие "сегмент" теперь означает некоторую область данных, начинающуюся с произвольного 32-битного адреса и заканчивающуюся так же произвольным 32-битным адресом, а вышеупомянутая таблица хранит данные об этих сегментах - собственно стартовые адреса и размеры. Номер сегмента в этой таблице (нумерация начинается с нуля) называется селектором. Программист, указывая сегмент и смещение, на самом деле указывает селектор и смещение, а процессор, обращаясь к таблице сегментов, находит там стартовый адрес сегмента и прибавляет к нему смещение. Таким образом, теперь сегменты не обязательно начинаются на 16-байтной границе - их стартовый адрес произволен.

Следует оговориться: 32-битные процессоры имеют два режима работы - реальный и защищенный, и старая модель сегментации работает в реальном режиме процессора, а новая - в защищенном. (На самом деле здесь есть свои нюансы - например, можно "обхитрить" процессор и в реальном режиме адресовать все 4 Гб, благодаря так называемым "теневым" сегментным регистрам - но это выходит за рамки статьи).

Остается добавить, что большинство современного ПО вообще не трогает сегментные регистры - операционная система дает им один-единственный сегмент, которые начинаются с нулевого адреса и заканчиваются 2^32 - 1 (таким образом охватывая всю возможную оперативную память) - и загружает этими данными регистры CS, DS, SS ES.

Виртуальное адресное пространство процесса. Страничная трансляция

С появлением первых версий мультизадачности на 286-х процессорах перед разработчиками ПО встала проблема. Оперативной памяти мало для всех выполняющихся программ. Более того, не было четкого механизма разделения памяти программ - большинство пользовалось общими областями памяти, куда затем любая недружелюбная программа могла записать всё что угодно, испортив данные другой программы. Это давало широкое поле действия для вирусов и прочих

вредоносных программ. Нужно было, во-первых, обеспечить полную изоляцию процессов друг от друга, во-вторых, механизмы "подкачки" - сброса неиспользуемых областей памяти на диск, за счет чего освобождалась так необходимая оперативная память. Всё это привело к появлению механизма страничной трансляции.

С появлением страничной трансляции возникло два адреса - физический и виртуальный (или линейный). Память (как физическая, так и виртуальная) разбита на равного размера блоки - страницы. Каждая виртуальная страница памяти физически хранится в некоторой физической странице, причем на одну физическую страницу может приходиться несколько виртуальных. Самое интересное - то, что программа пользуется виртуальными адресами и ничего не знает о том, где физически находится элемент данных, к которому она обращается по виртуальному адресу - да ей это и не нужно знать. Собственно трансляция (преобразование виртуального адреса в физический) выполняется аппаратно процессором. Кроме того, механизм страничной трансляции предусматривает возможность реализации "подкачки".

Выполнен такой механизм следующим образом. Каждое приложение имеет свою таблицу страниц, в которой прописаны соответствия линейных и физических адресов для каждой страницы, а также информация о правах доступа к странице и специальные признаки - признак отсутствия страницы и признак "загрязненности" страницы. В случае, когда приложению требуются какие-то данные, оно указывает виртуальный адрес, процессор преобразует его в физический и считывает с этого физического адреса требуемые данные. Допустим, тут в работу вступает другое приложение и выделяет себе большую область данных, но физической памяти не хватает для выделения столь большого объема. Тут в работу вступает механизм подкачки: операционная система ищет давно не использовавшиеся виртуальные страницы первого приложения и сбрасывает их содержимое в файл подкачки и тут же помечает эти виртуальные страницы как "отсутствующие", т. е. сброшенные в файл подкачки. Тут же выделяются виртуальные страницы второму приложению и в соответствие им ставятся физические адреса, содержимое которых было только что сброшено в файл. С этими адресами второе приложение и работает. Когда механизм переключения задач переключается снова на первую программу, она, пытается обратиться к сброшенным в файл подкачки данным, и тут процессор обнаруживает признак отсутствия страницы, и вызывает специальные функции операционной системы, которые проделывают работу в обратном направлении - теперь виртуальные страницы второго приложения сбрасываются на диск и помечаются отсутствующими, а содержимое виртуальных страниц первого приложения загружается из файла в оперативную память и страницы эти помечаются как присутствующие. Дополнительный признак - признак "загрязненности" - выставляется всякий раз, как только происходит запись в страницу, и позволяет операционной системе не сохранять данные страницы в файле в момент ее сброса, так как изменений не было, тем самым сэкономив время.

Следует отметить, что совершенно необязательно иметь два приложения, чтобы механизм подкачки начал работать - одна физическая страница может использоваться и для хранения нескольких виртуальных страниц одного и того же приложения, при этом механизм подкачки работает точно таким же образом - при обращении к виртуальной странице другая страница, та, что находилась в этой физической странице прежде, сбрасывается на диск и помечается как отсутствующая, а затребованная страница подгружается из файла подкачки. Естественно, одной физической странице может соответствовать сколь угодно много виртуальных.

Таким образом, механизм страничной трансляции убивает сразу двух зайцев - каждое приложение имеет свое виртуальное адресное пространство (т. к. виртуальные страницы и таблица страниц у каждого приложения своя) и обеспечивается механизм подкачки, совершенно невидимый для приложений - операционная система, имея весьма ограниченный объем памяти, имитирует наличие всех четырех гигабайт пространства для каждого приложения, причем приложения никак не смогут

обратиться к адресному пространству других процессов - это физически невозможно (если только страницы не являются "общими", об этом ниже).

Механизм подкачки на самом деле убивает еще и третьего зайца: он предоставляет такие возможности, как отображение файлов на память и обмен данными между процессами. Дело в том, что приложение может запросить операционную систему "отобразить файл на память" - при этом создается виртуальная страница (страницы), с которыми приложение работает как с обычной памятью, но содержимое которых будет сбрасываться не в файл подкачки, а в указанный приложением файл. Таким образом нет необходимости записывать данные в файл с помощью обычных файловых операций. Кроме того, если несколько приложений отобразят один и тот же файл в память, они получат "разделенную" область памяти - у каждого приложения, конечно, будет своя виртуальная страница, содержащая данные файла, но эти виртуальные страницы разных приложений будут содержаться в одной и той же физической странице и не будут поочередно сбрасываться - они будут присутствующими одновременно. Таким образом опять же убиваются два зайца - появляется простой и удобный способ работы с файлами без явных операций чтения и записи в файл, и одновременно появляется "мгновенный" способ обмена данными - ведь поскольку виртуальные страницы каждого приложения "сидят" на одном физическом адресе, после записи данных одним приложением они мгновенно доступны для другого.

Физически таблица страниц может быть реализована как двухуровневая таблица - имеется каталог таблиц страниц с 1024 строками, каждая строка содержит адрес таблицы страниц, каждая таблица страниц состоит из 1024 строк, содержащих информацию об одной странице размером 4 Кб. Последние процессоры позволяют реализовать таблицу одним уровнем - есть таблица страниц, содержащая 1024 строки, каждая из которых хранит информацию о странице размером 4 Мб.

Остается отметить, что механизм страничной трансляции не обязателен - он включается и выключается специальными командами процессора. Кроме того, страничная трансляция вполне сочетается с 32-битной сегментной моделью.

В современных Windows-приложениях и приложениях других операционных систем прикладному программисту не нужно заботиться о сегментной организации - программе выделяется один большой сегмент, как уже было указано выше. Страничная трансляция также недоступна программисту для непосредственного управления. Однако все эти знания пригодятся системному программисту, разрабатывающему операционные системы, драйверы, менеджеры загрузки ОС и прочее системное ПО.

Организация виртуальной памяти

За исключением случаев написания программы непосредственно в машинных кодах, пользователю предоставляется возможность работать не с реальной, а с виртуальной памятью. Виртуализации подвергается не только оперативная, но и другие виды памяти (напомним, что физическая память современных машин является многоуровневой). Пользователи воспринимают память с характеристиками, отличными от характеристик реально существующей памяти. Помимо обычных свойств ОП (пословный вид доступа, непосредственная адресация, времена доступа, сравнимые с реальными) привносятся новые (неограниченный или очень большой объем непрерывного адресного пространства). Виртуализация может осуществляться и в отношении регистровой памяти. Виртуализация внешней памяти также имеет место и называется *файловой системой*. Кроме стандартных свойств внешней памяти (долговременное хранение информации, произвольность объемов хранения и т.д.) пользователь получает доступ к данным отличными от реальных способами (выборка информации с учетом ее структурно-логической упорядоченности).

4.6

Виртуальная память функционально рассматривается по уровням, эквивалентным уровням реальной памяти (*вертикальная схема*). Отличие алгоритмов построения виртуальной памяти в пределах уровня соответствует расслоению по *горизонтальной схеме*.

Физически память представляет собой некоторую среду хранения, составленную из однотипных элементов. Каждый элемент способен хранить информацию и является адресуемым в соответствии с принятым для данной среды способом адресации. Всё доступное множество адресов элементов хранения, упорядоченное по какому-либо признаку, называют *адресным пространством* памяти. Форма задания адреса полностью определяется механизмом доступа к элементам хранения, принятым для данного вида памяти.

Обычно адрес – это число, которое однозначно определяет номер требуемого элемента хранения. Адресное пространство в этом случае есть последовательность целых чисел, начинающаяся с 0. Число различных адресов конечно и равно N, т.е. размеру адресного пространства (или объему памяти V).

В качестве адреса может быть использовано некоторое символьное имя, однозначно определяющее элемент хранения. Такие адреса называют виртуальными, а их совокупность – виртуальным адресным пространством. Виртуальное адресное пространство поддерживается с помощью программно-аппаратного механизма доступа, который является средством виртуализации физического адресного пространства. Программный слой обеспечивает преобразование каждого виртуального адреса в некоторый физический адрес.

Адресное пространство символических адресов, используемых в программах на ассемблере, можно рассматривать как виртуальное адресное пространство. Оно преобразуется статически в адресное пространство физической памяти средствами транслятора и загрузчика, которые в данном случае представляют собой программный компонент механизма доступа к ОП.

При построении виртуальной ОП приходиться распределять не только виртуальное и физическое адресные пространства ОП, но и адресное пространство внешней памяти.

В представлении как виртуального, так и физического адресного пространства оперативной памяти применяется несколько схем структуризации. В основе их лежит два класса схем структуризации: *страничная схема* и *сегментная схема*.

Рассмотрим сначала страничную схему. На первом шаге ее реализации сгруппируем адреса в адресном пространстве 0£A £N-1. В каждую группу, называемую страницей, должно входить одинаковое число адресов L. Обычно $N=2^n$, $L=2^m$, тогда $2^{(n-m)}$ есть число страниц. Последовательность номеров страниц есть k= 0,1,2,…, $2^{(n-m)}$-1. Адресом начала страницы номер k является $A_{k0}=kL$.

Второй шаг структуризации заключается в том, что нумеруются адреса независимым образом внутри каждой страницы. В пределах каждой страницы адрес R меняется 0 до $2^{(m-1)}$ и называется *смещением*. В результате адресное пространство превращается в двумерное, состоящее из адресных пар (k, R), где k – номер страницы, R – смещение. Каждая страница при этом рассматривается как отдельное адресное пространство.

Переход от странично-структурированного адресного пространства к одномерному непрерывному осуществляется по формуле A=kL+R по принципу *база + смещение*.

В случае страничной схемы с переменными страницами для задания адреса необходимы две координаты – базовый адрес страницы A_{k0} и смещение R , которые в совокупности образуют

адресную пару (A_{k0}, R). В этом случае переход от структурного представления (A_{k0}, R) к непрерывному выполняется по формуле A= A_{k0}+R, обратный же переход невозможен.

Второй класс схем структуризации называется сегментным. Он представляет собой обобщение варианта структуризации переменными страницами.

На первом шаге структуризации производят объединение адресов в группы, называемые сегментами. Размеры сегментов 1£L_s£N, где s – номер сегмента. Упорядочивание номеров необязательно, номер выступает лишь как идентификатор сегмента.

На втором шаге структуризации каждому сегменту ставится в соответствие его базовый адрес A_{S0} , так что каждый сегмент определяется своими координатами (s, A_{S0}). В каждом сегменте производится перенумерация адресов, адрес внутри сегмента называется смещением. Получаем трехмерное адресное пространство: (s, A_{S0}, R). В рассматриваемой схеме можно осуществить переход от трехмерного адреса к одномерному непрерывному, обратный же переход невозможен. На практике при использовании сегментной схемы структуризации задают две координаты (s, R). Тогда при переходе к непрерывному адресу необходима операция установления соответствия (сегменту с номером s присваивается базовый адрес A_{S0}).

Используются также комбинированные схемы структуризации, называемые *сегментно-страничными*: адресное пространство сначала структуризуется фиксированными страницами, затем происходит группировка страниц в сегменты. Каждый сегмент с номером s содержит некоторое количество страниц, нумеруемых с нуля в возрастающем порядке. Адрес задается четырьмя координатами (s, A_{S0}, R', R), где s – номер сегмента, A_{S0} – базовый адрес сегмента, R' – смещение страницы в пределах сегмента, R – смещение в пределах страницы.

Виртуальная память строится как для удовлетворения нужд пользователя, так и для повышения эффективности работы ОС в плане управления главными ресурсами – оперативной памятью и центральным процессором.

Использование виртуальной памяти позволяет ОС исключить потери памяти и уменьшить задержку при обращении процессов к данным, обеспечивает условия для функционирования максимального числа параллельных процессов, не мешающих друг другу, т.е. повышает пропускную способность системы. Виртуальная память дает возможность ОС решать и другие важные задачи: эффективное использования программных ресурсов, защита информации при развитии процессов, обеспечение взаимодействие между процессами и т. д.

Требования пользователей разнообразны и весьма противоречивы: одни пишут короткие программы, другие – длинные (больше объема ОП). Некоторые программы не структурированы (не содержат в своём составе дистанционно вызываемых программных единиц типа подпрограмм, процедур, блоков и т. д.), другие, наоборот, состоят из ведущей программы и некоторые числа вызываемых ей программных единиц.

Процесс исполнения программы одного пользователя может взаимодействовать с процессом исполнения программы другого пользователя посредством разделения одного и того же программного ресурса (процедуры). Программа отдельного пользователя при погружении в виртуальную память характеризуется двумя компонентами: текстом и адресным пространством, используемым в ней. Текст логически состоит из двух частей: кодовый сегмент и сегмент данных. Кодовая часть программы и данные могут быть структурированы (разбиты на логически обособленные части), такая структуризация распространяется и на адресное пространство.

4.6

Возможны две ситуации. В первом случае пользователь пишет программы без структуризации в пределах непрерывного адресного пространства. Во втором случае пользователь пишет уже структурированную программу, состоящую из функционально обособленных элементов. Адресное пространство каждого такого элемента рассматривается в виртуальном адресном пространстве как отдельный сегмент. Имена сегментов и смещения задает программист.

Каждая программа, которая погружена в виртуальную память, должна подвергаться структуризации. Если она на выходе не структурирована, то структуризацию проводят системные средства – трансляторы, программно-аппаратные средства. Даже если программа структурирована, она может быть подвергнута дополнительной структуризации.

Структуризация программы нужна для того, чтобы в процессе функционирования виртуальной памяти отдельные фрагменты (кодовые сегменты, массивы данных) можно было автоматически переносить на уровень внешней памяти и обратно на уровень ОП. Уровень внешней памяти используется для хранения, как копий, так и оригиналов информации, помещенной в ОП. В ОП хранится информация, непосредственно используемая процессами. Поэтому эти среды хранения называются соответственно *архивной* и *рабочей*. Элементы хранения каждой из них адресуются в пределах физических адресных пространств внешней и оперативной памяти.

Структуризация программ (программных текстов и соответствующих им адресных пространств) позволяет наиболее просто и эффективно решать системные задачи распределения рабочей и архивной сред хранения при помещении в них таких программ и их частей.

Используются четыре *логические схемы функционирования виртуальной памяти*, основанные на различных схемах структуризации памяти (страничной, сегментной и странично-сегментной).

1. *Чисто страничная схема функционирования виртуальной оперативной памяти*. Разбиению на страницы одинакового размера $L=2^m$ подвергается как виртуальное, так и реальное адресные пространства. Структуризация осуществляется системой, а не пользователем. Весь текст программы размером Q слов полностью помещается в оперативную память, для чего требуется K=[Q/L]+1 виртуальных страниц, где [x] означает целую часть числа x.

При этом страницы располагаются не в одной непрерывной области (как предполагает пользователь), а в нескольких несмежных областях. При такой схеме ОС поставлена в наиболее благоприятные условия, поскольку вместо того, чтобы выделить один непрерывный участок памяти размером Q, ОС может найти K несмежных блоков памяти, что более вероятно.

2. *Страничная (по требованию) схема функционирования виртуальной оперативной памяти*. Если отказаться от основного принципа предыдущей схемы (программа должна полностью находиться в оперативной памяти), то получим данную схему. Этот подход возможен в связи с двумя предпосылками:

1) В типичной программе различные части адресного пространства исполняются с различной интенсивностью. Поэтому следует стремиться к тому, чтобы соответствующие части программы не загружалась в память.

2) Имеет место локальность вычислений – в течение малого промежутка времени Dt происходит обращение к памяти в пределах ограниченного интервала адресов. Обращение к другому блоку памяти может произойти в более редких случаях либо по командам перехода (условного или безусловного), либо после исполнения последней команды данного блока.

Поэтому основной принцип такой схемы состоит в следующем – в ОП достаточно помещать лишь некоторую часть программы поблочно в соответствие со страничной организацией. Система должна отслеживать обращение к незагруженным частям программы, выделять для них страницы и помещать туда соответствующие блоки. Исполнение далее происходит на основе преобразования адресов. Отображение виртуальных страниц на физические происходит динамически и по частям, поэтому виртуальное адресное пространство может сколь угодно превышать адресное пространство реальной ОП. При данном подходе выигрывает как система, так и пользователь

3. *Сегментная схема функционирования виртуальной оперативной памяти*. Адресное пространство и виртуальной памяти и физической ОП сегментно-структурировано. Предполагается также, что пользователь пишет сегментно-структурированную программу, текст которой состоит из произвольного числа сегментов. Размер N_i каждой i-й составной части программы произволен, но не превосходит размера оперативной памяти. Перед исполнением каждая часть программы заносится в архивную среду. Система производит замену пользовательского имени виртуального сегмента s’ на системный номер сегмента s, последовательно размещая сегменты в едином виртуальном адресном пространстве пользователя. При этом не требуется отображения всего адресного пространства программы на адресное пространство оперативной памяти. Перед исполнением программы выделяется непрерывная область в оперативной памяти (физический сегмент) для первой исполняемой части программы, написанной в пределах некоторого виртуального сегмента. По мере исполнения этой части производится отображение данного виртуального сегмента на выделенный физический. Система отслеживает обращение из одного сегмента к другому в составе одной программы и динамически обеспечивает связь с вызываемым сегментом. Вызываемая часть программы переносится из архивной среды в рабочую, после чего в процессе ее исполнения происходит отображение виртуального сегмента адресов на выделенный физический сегмент и т.д. Данная схема функционирования ОП ориентирована на пользователя. Он имеет возможность не только писать произвольно большие программы, но и разрабатывать их на основе принципов модульного программирования.

4. *Сегментно-страничная схема функционирования виртуальной оперативной памяти*. Здесь адресное пространство оперативной памяти разбивается на фиксированные страницы, причем размеры виртуальной и физической страниц берутся одинаковыми. Данная схема также ориентирована на то, что пользователь пишет многомодульные программы в сегментно-структурированном виртуальном адресном пространстве. Но в отличие от предыдущей схемы при погружении программы в виртуальную память каждый виртуальный сегмент не только получает системный номер, но и подвергается дополнительной структуризации. Каждому сегменту длиной q_i отводится целое число виртуальных смежных страниц, перенумерованных в пределах каждого сегмента. При длине страницы $L=2^m$ их потребуется $K=[q_i/L]+1$.

При выполнении программы сначала выбирается модуль, который должен быть выполнен первым. Но соответствующий ему виртуальный сегмент не переносится полностью в ОП. В этом модуле выбирается совокупность команд, с которой начинается счет программы, находящаяся в пределах адресов некоторой виртуальной страницы. Именно для этой страницы и выделяется блок оперативной памяти. После перенесения в него текста программы устанавливается соответствие между адресами выбранной виртуальной страницы и выделенной физической страницы. По мере исполнения загруженной части программы происходит соответствующее преобразование виртуальных адресов в физические. При обращении к виртуальной странице, еще не отображенной на соответствующую физическую, система определяет требуемый сегмент, отыскивает в нем необходимую виртуальную страницу и производит ее отображение на некоторую физическую страницу.

Данная схема работы с виртуальной памятью обеспечивает должный компромисс при удовлетворении требований как пользователя, так и ОС. Система имеет возможность эффективно

распределять ОП и увеличивать пропускную способность на основе преимуществ страничной по требованию схемы, а пользователь – использовать преимущества модульного программирования и писать сколь угодно большие программы.

Свопинг

Один из механизмов реализации виртуальной памяти, при котором отдельные запущенные процессы (обычно неактивные) перемещаются из ОЗУ на жёсткий диск, освобождая ОЗУ для загрузки других процессов. Основное отличие этого механизма от страничного заключается в том, что процессы перемещаются между ОЗУ и жестким диском целиком, поэтому иногда некоторые процессы могут полностью отсутствовать в ОЗУ. При наступлении условий активизации процесса он возвращается диспетчером памяти в ОЗУ.

Фрагментация файла подкачки

В процессе работы файл (раздел диска, или файл на разделе) подкачки может стать фрагментированым, то есть непрерывные виртуальные области памяти будут состоять из многочисленных отдельных (разрывных) областей в файле подкачки. При считывании и записи данных страниц много времени будет уходить на перепозиционирование головок жёсткого диска на начало очередной области. Это может привести к падению производительности всей системы.

Использование свопинга особенно эффективно, если запущено много интерактивных приложений, которые потребляют большой объем ОЗУ, но при этом практически не занимают процессорное время.

Наиболее эффективным способом выделения места для swap‑файла считается кратное выделение памяти, то есть объём оперативной памяти, умноженный на 1, на 2, на 3 и т. д.

Алгоритмы определения устаревших страниц

При выделении места для новой страницы необходимо удалить какую-либо страницу, в данный момент находящуюся в памяти. Правила замещения страниц служат для принятия решения о том, какую именно страницу следует удалить из памяти. Идеальным кандидатом является «мёртвая» страница, которая больше не потребуется кому-либо (например, относится к завершённому процессу). Если же таких страниц нет в памяти (или их количества недостаточно), используется правило локального или глобального замещения страниц:

- Правило локального замещения выделяет каждому процессу или группе взаимосвязанных процессов определённое количество страниц. Если процессу нужна новая страница, он должен заменить одну из собственных.
- Правило глобального замещения страниц позволяет брать страницы любого процесса, используя глобальные критерии выбора. Для реализации данного подхода необходимо выбрать критерий, по которому будет приниматься решение о страницах, хранимых в памяти.

Наиболее часто используемые критерии поиска:

- Less Recently Used. Удаляются те страницы, доступ к которым производился наиболее давно. Считается, что в последующем к таким страницам будет происходить минимум обращений.
- Last Recently Used. Удаляются недавно освободившиеся страницы. Подразумеваются страницы только что завершившихся процессов.

4.6