

Функции системы управления памятью

Чтобы обеспечить эффективный контроль использования памяти, ОС должна выполнять следующие функции:

- отображение *адресного пространства* процесса на конкретные области физической памяти;
- распределение памяти между конкурирующими процессами;
- контроль доступа к *адресным пространствам* процессов;
- выгрузка процессов (целиком или частично) во внешнюю память, когда в *оперативной памяти* недостаточно места;
- учет свободной и занятой памяти.

Ниже рассматривается ряд конкретных схем управления памятью. Каждая схема включает в себя определенную идеологию управления, а также алгоритмы и структуры данных и зависит от архитектурных особенностей используемой системы. Вначале будут рассмотрены простейшие схемы. Доминирующая на сегодня схема виртуальной памяти будет описана в последующих лекциях.

Простейшие схемы управления памятью.

Первые ОС применяли очень простые методы управления памятью. Вначале каждый процесс пользователя должен был полностью поместиться в *основной памяти*, занимать непрерывную область памяти, а система принимала к обслуживанию дополнительные пользовательские процессы до тех пор, пока все они одновременно помещались в *основной памяти*.

Затем появился "простой свопинг". Система по-прежнему размещает каждый процесс в *основной памяти* целиком, но иногда на основании некоторого критерия целиком сбрасывает образ некоторого процесса из *основной памяти* во внешнюю и заменяет его в *основной памяти* образом другого процесса. Такого рода схемы имеют не только историческую ценность. В настоящее время они применяются в учебных и научно-исследовательских модельных ОС, а также в ОС для встроенных (embedded) компьютеров.

Схема с фиксированными разделами.

Самым простым способом управления *оперативной памятью* является ее предварительное (обычно на этапе генерации или в момент загрузки системы) разбиение на несколько разделов фиксированной величины. Поступающие процессы помещаются в тот или иной раздел. При этом происходит условное разбиение физического *адресного пространства*. Связывание логических и физических адресов процесса происходит на этапе его загрузки в конкретный раздел, иногда – на этапе компиляции. Каждый раздел может иметь свою очередь процессов, а может существовать и глобальная очередь для всех разделов. Эта схема была реализована в IBM OS/360 (MFT), DEC RSX-11 и ряде других систем. Подсистема управления памятью оценивает размер поступившего процесса, выбирает подходящий для него раздел, осуществляет загрузку процесса в этот раздел и настройку адресов.

Очевидный недостаток этой схемы – число одновременно выполняемых процессов ограничено числом разделов. Другим существенным недостатком является то, что

предлагаемая схема сильно страдает от *внутренней фрагментации* – потери части памяти, выделенной процессу, но не используемой им. *Фрагментация* возникает потому, что процесс не полностью занимает выделенный ему раздел или потому, что некоторые разделы слишком малы для выполняемых пользовательских программ.

Один процесс в памяти.

Частный случай схемы с *фиксированными разделами* – работа менеджера памяти однозадачной ОС. В памяти размещается один пользовательский процесс. Остается определить, где располагается пользовательская программа по отношению к ОС – в верхней части памяти, в нижней или в средней. Причем часть ОС может быть в ROM (например, BIOS, драйверы устройств). Главный фактор, влияющий на это решение, – расположение вектора прерываний, который обычно локализован в нижней части памяти, поэтому ОС также размещают в нижней. Примером такой организации может служить ОС MS-DOS. Защита *адресного пространства* ОС от пользовательской программы может быть организована при помощи одного граничного регистра, содержащего адрес границы ОС.

Оверлейная структура.

Так как размер логического *адресного пространства* процесса может быть больше, чем размер выделенного ему раздела (или больше, чем размер самого большого раздела), иногда используется техника, называемая оверлей (overlay) или организация структуры с перекрытием.

Основная идея – держать в памяти только те инструкции программы, которые нужны в данный момент. Потребность в таком способе загрузки появляется, если логическое *адресное пространство* системы мало, например 1 Мбайт (MS-DOS) или даже всего 64 Кбайта (PDP-11), а программа относительно велика. На современных 32-разрядных системах, где виртуальное *адресное пространство* измеряется гигабайтами, проблемы с нехваткой памяти решаются другими способами (см. раздел "Виртуальная память"). Коды ветвей *оверлейной структуры* программы находятся на диске как абсолютные образы памяти и считываются драйвером оверлеев при необходимости. Для описания *оверлейной структуры* обычно используется специальный несложный язык (overlay description language). Совокупность файлов исполняемой программы дополняется файлом (обычно с расширением .odl), описывающим дерево вызовов внутри программы.

Синтаксис подобного файла может распознаваться загрузчиком. Привязка к *физической памяти* происходит в момент очередной загрузки одной из ветвей программы. Оверлеи могут быть полностью реализованы на пользовательском уровне в системах с простой файловой структурой. ОС при этом лишь делает несколько больше операций ввода-вывода. Типовое решение – порождение линкером специальных команд, которые включают загрузчик каждый раз, когда требуется обращение к одной из перекрывающихся ветвей программы.

Тщательное проектирование *оверлейной структуры* отнимает много времени и требует знания устройства программы, ее кода, данных и языка описания *оверлейной структуры*. По этой причине применение оверлеев ограничено компьютерами с небольшим логическим *адресным пространством*. Как мы увидим в дальнейшем, проблема оверлейных *сегментов*, контролируемых программистом, отпадает благодаря появлению систем виртуальной памяти. Заметим, что возможность организации структур с перекрытиями во многом обусловлена свойством локальности, которое позволяет

хранить в памяти только ту информацию, которая необходима в конкретный момент вычислений.

Динамическое распределение. Свопинг

Имея дело с пакетными системами, можно обходиться *фиксированными разделами* и не использовать ничего более сложного. В системах с разделением времени возможна ситуация, когда память не в состоянии содержать все пользовательские процессы. Приходится прибегать к свопингу (swapping) – перемещению процессов из главной памяти на диск и обратно целиком. Частичная выгрузка процессов на диск осуществляется в системах со страничной организацией (paging).

Выгруженный процесс может быть возвращен в то же самое *адресное пространство* или в другое. Это ограничение диктуется методом *связывания*. Для схемы *связывания* на этапе выполнения можно загрузить процесс в другое место памяти.

Свопинг не имеет непосредственного отношения к управлению памятью, скорее он связан с подсистемой планирования процессов. Очевидно, что свопинг увеличивает время переключения контекста. Время выгрузки может быть сокращено за счет организации специально отведенного пространства на диске (раздел для свопинга). Обмен с диском при этом осуществляется блоками большего размера, то есть быстрее, чем через стандартную файловую систему. Во многих версиях Unix свопинг начинает работать только тогда, когда возникает необходимость в снижении загрузки системы.

Схема с переменными разделами.

В принципе, система свопинга может базироваться на *фиксированных разделах*. Более эффективной, однако, представляется схема динамического распределения или схема с переменными разделами, которая может использоваться и в тех случаях, когда все процессы целиком помещаются в памяти, то есть в отсутствие свопинга. В этом случае вначале вся память свободна и не разделена заранее на разделы. Вновь поступающей задаче выделяется строго необходимое количество памяти, не более. После выгрузки процесса память временно освобождается. По истечении некоторого времени память представляет собой переменное число разделов разного размера. Смежные свободные участки могут быть объединены.

В какой раздел помещать процесс? Наиболее распространены три стратегии.

- **Стратегия первого подходящего (First fit).** Процесс помещается в первый подходящий по размеру раздел.
- **Стратегия наиболее подходящего (Best fit).** Процесс помещается в тот раздел, где после его загрузки останется меньше всего свободного места.
- **Стратегия наименее подходящего (Worst fit).** При помещении в самый большой раздел в нем остается достаточно места для возможного размещения еще одного процесса.

Моделирование показало, что доля полезно используемой памяти в первых двух случаях больше, при этом первый способ несколько быстрее. Попутно заметим, что перечисленные стратегии широко применяются и другими компонентами ОС, например для размещения файлов на диске.

Типовой цикл работы менеджера памяти состоит в:

- **анализе запроса на выделение свободного участка** (раздела),
 - **выборе его** среди имеющихся в соответствии с одной из стратегий (первого подходящего, наиболее подходящего и наименее подходящего),
 - **загрузке процесса** в выбранный раздел и
 - **последующих изменениях таблиц свободных и занятых областей.**
- Аналогичная корректировка необходима и после завершения процесса.

Связывание адресов может осуществляться на этапах загрузки и выполнения. Этот метод более гибок по сравнению с методом *фиксированных разделов*, однако ему присуща **внешняя фрагментация** – наличие большого числа участков неиспользуемой памяти, не выделенной ни одному процессу. Выбор стратегии размещения процесса между первым подходящим и наиболее подходящим слабо влияет на величину *фрагментации*. Любопытно, что метод наиболее подходящего может оказаться наихудшим, так как он оставляет множество мелких незанятых блоков. Статистический анализ показывает, что пропадает в среднем 1/3 памяти! Это известное правило 50% (два соседних свободных участка в отличие от двух соседних процессов могут быть объединены).

Одно из решений проблемы *внешней фрагментации* – организовать сжатие, то есть перемещение всех занятых (свободных) участков в сторону возрастания (убывания) адресов, так, чтобы вся свободная память образовала непрерывную область. Этот метод иногда называют схемой с перемещаемыми разделами. В идеале *фрагментация* после сжатия должна отсутствовать. Сжатие, однако, является дорогостоящей процедурой, алгоритм выбора оптимальной стратегии сжатия очень труден и, как правило, сжатие осуществляется в комбинации с выгрузкой и загрузкой по другим адресам.

Страничная память

Описанные выше схемы недостаточно эффективно используют память, поэтому в современных схемах управления памятью не принято размещать процесс в оперативной памяти одним непрерывным блоком.

В самом простом и наиболее распространенном случае страничной организации памяти (или paging) как логическое адресное пространство, так и физическое представляются состоящими из наборов блоков или страниц одинакового размера. При этом образуются логические страницы (page), а соответствующие единицы в физической памяти называют физическими страницами или страничными кадрами (page frames). Страницы (и страничные кадры) имеют фиксированную длину, обычно являющуюся степенью числа 2, и не могут перекрываться. Каждый кадр содержит одну страницу данных. При такой организации внешняя фрагментация отсутствует, а потери из-за внутренней фрагментации, поскольку процесс занимает целое число страниц, ограничены частью последней страницы процесса.

Логический адрес в страничной системе – упорядоченная пара (p, d) , где p – номер страницы в виртуальной памяти, а d – смещение в рамках страницы p , на которой размещается адресуемый элемент. Заметим, что разбиение адресного пространства на страницы осуществляется вычислительной системой незаметно для программиста. Поэтому адрес является двумерным лишь с точки зрения операционной системы, а с точки зрения программиста адресное пространство процесса остается линейным. Описываемая схема позволяет загрузить процесс, даже если нет непрерывной области кадров, достаточной для размещения процесса целиком. Но одного базового регистра для осуществления трансляции адреса в данной схеме недостаточно. Система отображения логических адресов в физические сводится к системе отображения логических страниц в физические и представляет собой таблицу страниц, которая хранится в оперативной

памяти. Иногда говорят, что таблица страниц – это кусочно-линейная функция отображения, заданная в табличном виде.

Интерпретация логического адреса показана на . Если выполняемый процесс обращается к логическому адресу $v = (p, d)$, механизм отображения ищет номер страницы p в таблице страниц и определяет, что эта страница находится в страничном кадре p' , формируя реальный адрес из p' и d .

Таблица страниц (page table) адресуется при помощи специального регистра процессора и позволяет определить номер кадра по логическому адресу. Помимо этой основной задачи, при помощи атрибутов, записанных в строке таблицы страниц, можно организовать контроль доступа к конкретной странице и ее защиту.

Отметим еще раз различие точек зрения пользователя и системы на используемую память. С точки зрения пользователя, его память – единое непрерывное пространство, содержащее только одну программу. Реальное отображение скрыто от пользователя и контролируется ОС. Заметим, что процессу пользователя чужая память недоступна. Он не имеет возможности адресовать память за пределами своей таблицы страниц, которая включает только его собственные страницы.

Для управления физической памятью ОС поддерживает структуру таблицы кадров. Она имеет одну запись на каждый физический кадр, показывающий его состояние.

Отображение адресов должно быть осуществлено корректно даже в сложных случаях и обычно реализуется аппаратно. Для ссылки на таблицу процессов используется специальный регистр. При переключении процессов необходимо найти таблицу страниц нового процесса, указатель на которую входит в контекст процесса.

Сегментная и сегментно-страничная организация памяти

Существуют две другие схемы организации управления памятью: сегментная и сегментно-страничная. Сегменты, в отличие от страниц, могут иметь переменный размер. Идея сегментации изложена во введении. При сегментной организации виртуальный адрес является двумерным как для программиста, так и для операционной системы, и состоит из двух полей – номера сегмента и смещения внутри сегмента.

Подчеркнем, что в отличие от страничной организации, где линейный адрес преобразован в двумерный операционной системой для удобства отображения, здесь двумерность адреса является следствием представления пользователя о процессе не в виде линейного массива байтов, а как набор сегментов переменного размера (данные, код, стек...).

Программисты, пишущие на языках низкого уровня, должны иметь представление о сегментной организации, явным образом меняя значения сегментных регистров (это хорошо видно по текстам программ, написанных на Ассемблере). Логическое адресное пространство – набор сегментов. Каждый сегмент имеет имя, размер и другие параметры (уровень привилегий, разрешенные виды обращений, флаги присутствия). В отличие от страничной схемы, где пользователь задает только один адрес, который разбивается на номер страницы и смещение прозрачным для программиста образом, в сегментной схеме пользователь специфицирует каждый адрес двумя величинами: именем сегмента и смещением.

Каждый сегмент – линейная последовательность адресов, начинающаяся с 0. Максимальный размер сегмента определяется разрядностью процессора (при 32-разрядной адресации это 232 байт или 4 Гбайт). Размер сегмента может меняться динамически (например, сегмент стека). В элементе таблицы сегментов помимо физического адреса начала сегмента обычно содержится и длина сегмента. Если размер

смещения в виртуальном адресе выходит за пределы размера сегмента, возникает исключительная ситуация.

Логический адрес – упорядоченная пара $v=(s,d)$, номер сегмента и смещение внутри сегмента.

В системах, где сегменты поддерживаются аппаратно, эти параметры обычно хранятся в таблице дескрипторов сегментов, а программа обращается к этим дескрипторам по номерам-селекторам. При этом в контекст каждого процесса входит набор сегментных регистров, содержащих селекторы текущих сегментов кода, стека, данных и т. д. и определяющих, какие сегменты будут использоваться при разных видах обращений к памяти. Это позволяет процессору уже на аппаратном уровне определять допустимость обращений к памяти, упрощая реализацию защиты информации от повреждения и несанкционированного доступа.

Аппаратная поддержка сегментов распространена мало (главным образом на процессорах Intel). В большинстве ОС сегментация реализуется на уровне, не зависящем от аппаратуры.

Хранить в памяти сегменты большого размера целиком так же неудобно, как и хранить процесс непрерывным блоком. Напрашивается идея разбиения сегментов на страницы. При сегментно-страничной организации памяти происходит двухуровневая трансляция виртуального адреса в физический. В этом случае логический адрес состоит из трех полей: номера сегмента логической памяти, номера страницы внутри сегмента и смещения внутри страницы. Соответственно, используются две таблицы отображения – таблица сегментов, связывающая номер сегмента с таблицей страниц, и отдельная таблица страниц для каждого сегмента.

Сегментно-страничная и страничная организация памяти позволяет легко организовать совместное использование одних и тех же данных и программного кода разными задачами. Для этого различные логические блоки памяти разных процессов отображают в один и тот же блок физической памяти, где размещается разделяемый фрагмент кода или данных.