УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите приведенную лекцию, законспектируйте основные понятия, дайте ответы на контрольные вопросы.

Ответы на вопросы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 08.05.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция
Тема: Функции системы управления памятью. Простейшие системы управления памятью.

Введение.

Главная задача компьютерной системы - выполнять программы. Программы, в течение выполнения, вместе с данными, к которым они имеют доступ должны (по крайней мере, частично) находиться в *главной (основной, оперативной)* памяти. Таким образом, память (storage, memory) является важнейшим ресурсом, требующим тщательного управления. В недавнем прошлом память - самый дорогой ресурс.

Часть ОС, которая управляет памятью, называется *менеджером памяти*. В процессе эволюции в менеджерах памяти современных ОС было реализовано несколько основополагающих идей.

Во-первых, это идея *сегментации.* По-видимому, вначале сегменты памяти появились в связи с необходимостью обобществления процессами фрагментов программного кода (текстовый редактор, тригонометрические библиотеки и т.д.), без чего каждый процесс должен был хранить в своем адресном пространстве дублирующую информацию. Эти отдельные участки памяти, хранящие информацию, которую система отображает в память нескольких процессов, получили название *сегментов*. Память, таким образом, стала двумерной. Адрес состоит из двух компонентов: номер сегмента,

смещение внутри сегмента. Далее оказалось удобным размещать в разных сегментах данные разных типов (код программы, данные, стек и т. д.). Попутно выяснилось, что можно контролировать характер работы с конкретным сегментом, приписав ему атрибуты, например, права доступа или типы операций, разрешенные с данными, хранящимися в сегменте. Большинство современных ОС поддерживают сегментную организацию памяти (см. раздел, где описана модель памяти процесса). В некоторых архитектурах (Intel, например) сегментация поддерживается оборудованием.

Вторая идея, о которой можно упомянуть, рассматривая поддержку памяти в ОС, это разделение памяти *на физическую и логическую*. Адреса, к которым обращается процесс, отделяются от адресов, реально существующих в оперативной памяти. Адрес, сгенерированный программой, обычно называют *логическим (в системах с виртуальной памятью он обычно называется виртуальным)* адресом, тогда как адрес, который видит устройство памяти (то есть нечто, загруженное в адресный регистр) обычно называется *физическим* адресом. Задача ОС, в какой-то момент времени осуществить связывание (или отображение) логического адресного пространства с физическим (см. раздел 8.2).

И, наконец, идея *локальности.* Свойство локальности присуще природе. Пространственная локальность - соседние объекты характеризуются похожими свойствами (если в данной местности хорошая погода, то вероятнее всего, что в близкой окрестности также хорошая погода). Временная локальность - если в 15:00 была хорошая погода, то, вероятно, что и в 14:30 и в 15:30 также наблюдалась хорошая погода. Свойство локальности (скорее эмпирическое) присуще и работе ОС. Фактически свойство локальности объяснимо, если учесть, как пишутся программы и организованы данные, то есть обычно в течение какого-то отрезка времени ограниченный фрагмент кода работает с ограниченным набором данных. Понимание данной особенности позволяет организовать *иерархию памяти,* используя быструю дорогостоящую память для хранения минимума необходимой информации, размещая оставшуюся часть данных на устройствах с более медленным доступом и

подкачивая их в быструю память по мере необходимости. Типичный пример иерархии: регистры процессора, кэш процессора, главная память, внешняя память на магнитных дисках (*вторичная память*).

Главная память - это массив слов или байт. Каждое слово имеет свой адрес. Использование вторичной памяти (хранение данных на дисках) в качестве расширения главной дает дополнительные преимущества. Во-первых, главная память слишком мала, чтобы содержать все необходимые программы и данные постоянно. Во-вторых, главная память есть изменчивое (volatile) устройство, которое теряет свое содержимое, когда питание отключено или по другим причинам. Одно из требований к вторичной памяти - умение хранить большие объемы данных постоянно.

Функциями ОС по управлению памятью являются: отображение адресов программы на конкретную область физической памяти, распределение памяти между конкурирующими процессами и защита адресных пространств процессов, выгрузка процессов на диск, когда в оперативной памяти недостаточно места для всех процессов, учет свободной и занятой памяти.

Существует несколько схем управления памятью. Выбор той или иной схемы зависит от многих факторов. Рассматривая ту или иную схему важно учитывать:

- Механизм управления памятью или идеологию построения системы управления.
- Архитектурные особенности используемой системы.
- Структуры данных в ОС, используемые для управления памятью.
- Алгоритмы, используемые для управления памятью.

Вначале будут рассмотрены простейшие схемы, затем, описана доминирующая на сегодня схема виртуальной памяти, ее аппаратная и программная поддержка.

1 Связывание адресов.

Одна из функций управления памятью отображение информации в память. Отображение обычно понимается как преобразование адресных пространств.

Как уже упоминалось в разделе 8.1, адреса, с которыми имеет дело менеджер памяти, бывают логические (виртуальные для систем с виртуальной памятью) и физические.

Пользовательская программа не видит реальных физических адресов, а имеет дело с логическими адресами, которые являются результатом трансляции символьных имен программы. Логические адреса обычно образуются на этапе создания загрузочного модуля (линковки программы).

Набор адресов, сгенерированный программой, называют *логическим (виртуальным) адресным пространством*, которому соответствует *физическое адресное пространство*.

Максимальный размер логического адресного пространства обычно определяется разрядностью процессора (например, 2**32) и в современных системах значительно превышает размер физического адресного пространства.

Связывание логического адреса, порожденного оператором программы, с физическим должно быть осуществлено до начала выполнения оператора или в момент его выполнения.

Обычно программа проходит нескольких шагов:

- текст на алгоритмическом языке,
- объектный модуль,
- загрузочный модуль,
- бинарный образ в памяти.

Используемые программой адреса в каждом конкретном случае могут быть представлены различными способами. Например, адреса в исходных текстах обычно символические. Компилятор связывает эти символические адреса с перемещаемыми адресами (такими как n байт от начала модуля). Загрузчик или линкер, в свою очередь, связывают эти перемещаемые адреса с виртуальными адресами. Каждое связывание - отображение одного адресного пространства в другое.

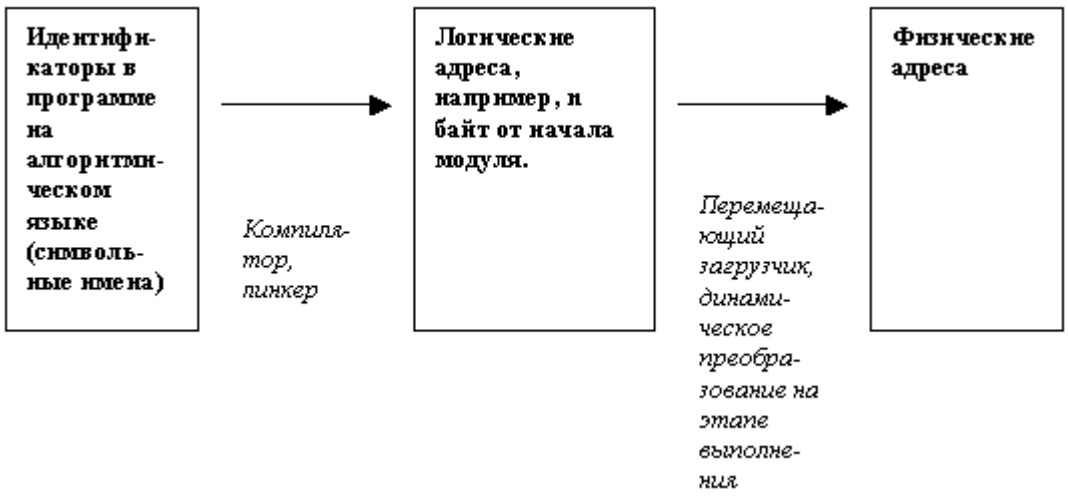

Рис. 1 Этапы связывания адресов.

Привязка инструкций и данных к памяти в принципе может быть, таким образом, сделана на следующих шагах:

- Этап компиляции (Compile time). Когда на стадии компиляции известно точное место размещения процесса в памяти, тогда генерируются абсолютные адреса. Если стартовый адрес программы меняется, необходимо перекомпилировать код. В качестве примера можно привести .com программы MS-DOS, которые связывают ее с физическими адресами на стадии компиляции.
- Этап загрузки (Load time). Если на стадии компиляции не известно где процесс будет размещен в памяти, компилятор генерирует перемещаемый код. В этом случае окончательное связывание откладывается до момента загрузки. Если стартовый адрес меняется, нужно всего лишь перезагрузить код с учетом измененной величины.
- Этап выполнения (Execution time). Если процесс может быть перемещен во время выполнения из одного сегмента памяти в другой, связывание откладывается до времени выполнения. Здесь желательно специализированное оборудование, например регистры перемещения. Их значение прибавляется к каждому адресу, сгенерированному процессом. Например, MS-DOS использует четыре таких (сегментных) регистра.

2 Простейшие схемы управления памятью.

ОС начали свое существование с применения очень простых методов управления памятью. Применявшаяся техника распространялась от статического распределения памяти (каждый процесс пользователя должен полностью поместиться в основной памяти, и система принимает к обслуживанию дополнительные пользовательские процессы до тех пор, пока все они одновременно помещаются в основной памяти), с промежуточным решением в виде "простого свопинга" (система по-прежнему располагает каждый процесс в основной памяти целиком, но иногда на основании некоторого критерия целиком сбрасывает образ некоторого процесса из основной памяти во внешнюю память и заменяет его в основной памяти образом некоторого другого процесса). Схемы такого рода 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.

2.1 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. По мере прибытия процесс помещается в тот или иной раздел.

Как правило, происходит условное разбиение физического адресного пространства. Связывание логических адресов процесса и физических происходит на этапе его загрузки в конкретный раздел.

Каждый раздел может иметь свою очередь или может существовать глобальная очередь для всех разделов.

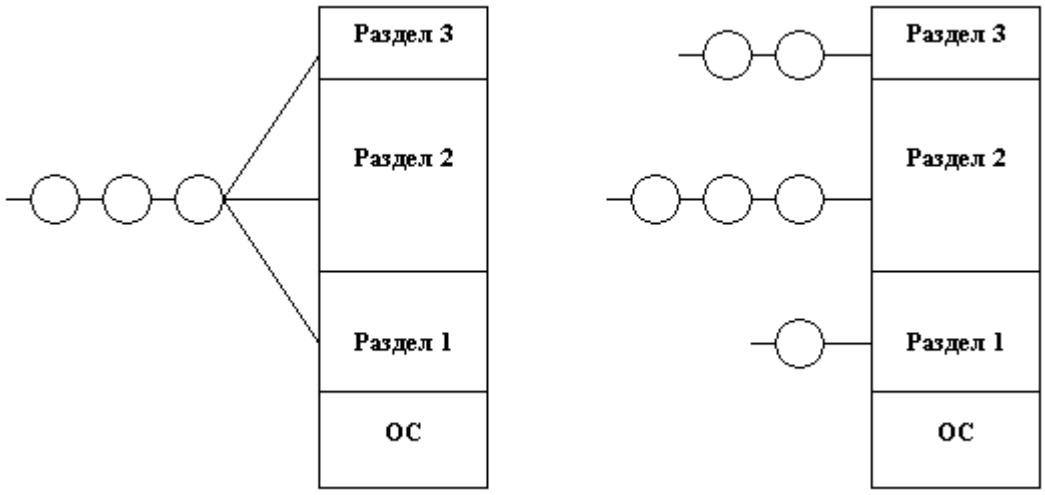

Рис. 2 Схема с фиксированными разделами: (a) с общей очередью процессов, (b) с отдельными очередями процессов.

Эта схема была реализована в IBM OS/360 (MFT) и в DEC RSX-11.

Подсистема управления памятью сравнивает размер программы, поступившей на выполнение, выбирает подходящий раздел, осуществляет загрузку программы и настройку адресов.

В какой раздел помещать программу? Распространены три стратегии:

- Стратегия первого подходящего (First fit). Задание помещается в первый подходящий по размеру раздел.
- Стратегия наиболее подходящего (Best fit). Задание помещается в тот раздел, где ему наиболее тесно.
- 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 с точки зрения утилизации памяти и уменьшения времени первые два способа лучше. С точки зрения утилизации первые два примерно одинаковы, но первый способ быстрее. Попутно заметим, что перечисленные стратегии широко применяются и другими компонентами ОС, например, для размещения файлов на диске.

Связывание (настройка) адресов для данной схемы возможны как на этапе компиляции, так и на этапе загрузки.

Очевидный недостаток этой схемы число одновременно выполняемых процессов ограничено числом разделов.

Другим существенным недостатком является то, что предлагаемая схема сильно страдает от *внешней фрагментации* потери памяти, не используемой ни одним процессом. Фрагментация возникает потому, что процесс не полностью занимает выделенный ему раздел или вследствие не использования некоторых разделов, которые слишком малы для выполняемых пользовательских программ.

2.2 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 - сверху, снизу или посередине. 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.

Чтобы пользовательская программа не портила кода ОС, требуется защита ОС, которая может быть организована при помощи одного граничного регистра, содержащего адрес границы ОС.

2.3 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 времени.

Потребность в таком способе загрузки появляется, если логическое адресное пространство системы мало, например 1 мегабайт (MS-DOS) или даже всего 64 килобайта (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).

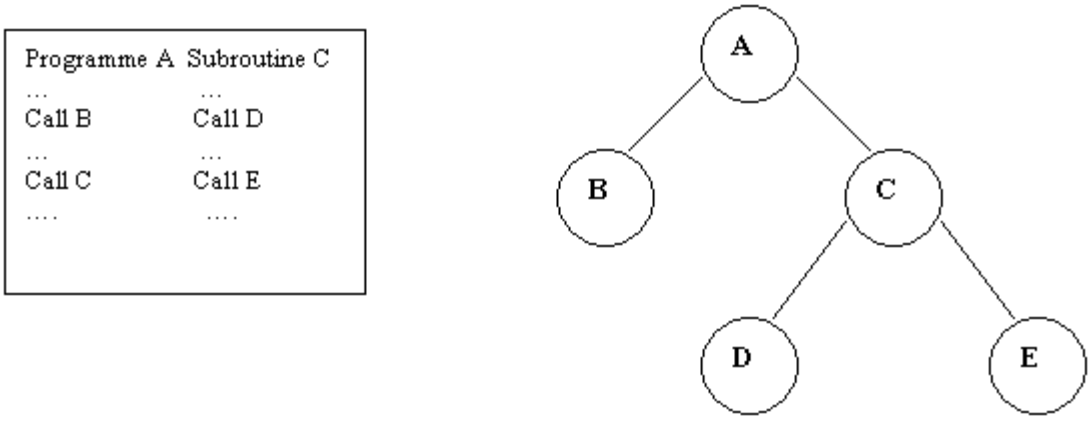

Рис 3 Организация структуры с перекрытием. Можно поочередно загружать в память ветви A-B, A-C-D и A-C-E программы.

Коды ветвей оверлейной структуры программы находятся на диске как абсолютные образы памяти и считываются драйвером оверлеев при необходимости. Для конструирования оверлеев необходимы специальные алгоритмы перемещения и связывания. 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. Например, для примера, приведенного на рис. 8.3 , текст этого файла может выглядеть так:

A-(B,C)

C-(D,E)

Синтаксис подобного файла может распознаваться загрузчиком. Привязка к памяти происходит в момент очередной загрузки одной из ветвей программы.

Оверлеи не требуют специальной поддержки со стороны ОС. Они могут быть полностью реализованы на пользовательском уровне с простой файловой структурой. ОС лишь делает несколько больше операций ввода-вывода. Типовое решение порождение линкером специальных команды, которые

включают загрузчик каждый раз: когда требуется обращение к одной из перекрывающихся ветвей программы.

Программист должен тщательно проектировать оверлейную структуру. Это требует полного знания структуры программы, кода, данных, языка описания оверлейной структуры. По этой причине применение оверлеев ограничено компьютерами с лимитами на память и т.д. Как мы увидим в дальнейшем проблема оверлейных сегментов, контролируемых программистом, отпадает благодаря появлению систем виртуальной памяти.

Заметим, что здесь мы впервые сталкиваемся со свойством локальности, которое дает возможность хранить в памяти только ту информацию, которая необходима в каждый конкретный момент вычислений.

3 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 связана с пейджингом (paging) будет рассмотрена ниже.

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. В системах со свопингом время переключения контекстов лимитируется временем загрузки выгрузки процессов. Для эффективной утилизации процессора необходимо, чтобы величина кванта времени существенно его превышала

Оптимизация свопинга может быть связана с выгрузкой лишь реально используемой памяти или выгрузкой процессов, реально не функционирующих. Кроме того, выгрузка обычно осуществляется в специально отведенное

пространство для свопинга, то есть быстрее, чем через стандартную файловую систему (пространство выделяется большими блоками, поиск файлов и методы непосредственного выделения не используются).

Во многих версиях Unix свопинг обычно запрещен, однако он стартует, когда возрастает загрузка системы.

4 Мультипрограммирование с переменными разделами.

В принципе, система свопинга может базироваться на фиксированных разделах. На практике, однако, использование фиксированных разделов приводит к большим потерям используемой памяти, когда задача существенно меньше раздела.

Более эффективной представляется схема с переменными (динамическими) разделами. В этом случае вначале вся память свободна и не разделена заранее на разделы. Вновь поступающей задаче выделяется необходимая память. После выгрузки процесса память временно освобождается. По истечении некоторого времени память представляет собой набор занятых и свободных участков (рис. 8.4) Смежные свободные участки могут быть объединены в один.

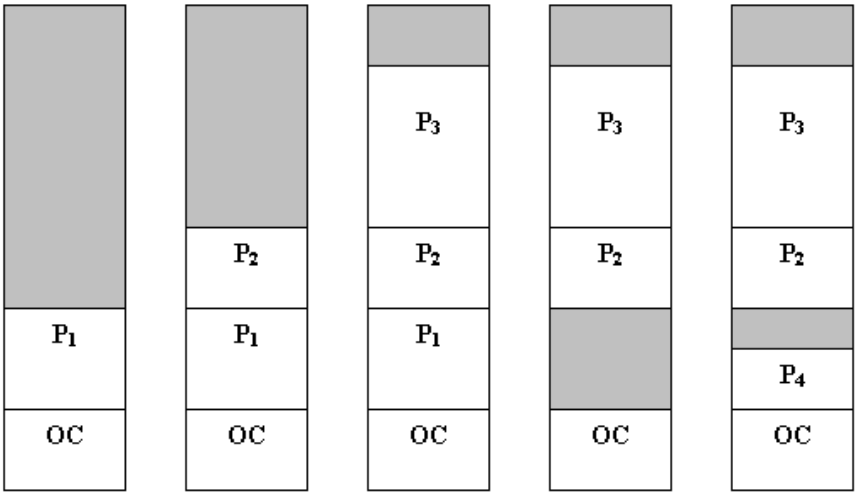

Рис. 4 Динамика распределения памяти между процессами. Серым цветом показана неиспользуемая память.

Типовой цикл работы менеджера памяти состоит в анализе запроса на выделение свободного участка (раздела), выборке его среди имеющихся в соответствие с одной из стратегий (first fit, best fit, worst fit), загрузке процесса

в выбранный раздел и последующем внесении изменений в таблицы свободных и занятых областей. Аналогичная корректировка необходима и после завершения процесса. Связывание адресов может быть осуществлено на этапах загрузки и выполнения.

Этот метод более гибок по сравнению с методом фиксированных разделов

Этому методу также присуща внешняя фрагментация вследствие наличия большого числа участков свободной памяти. Проблемы фрагментации могут быть различными. В худшем случае мы можем иметь участок свободной (потерянной) памяти между двумя процессами. Если все эти куски объединить в один блок, мы смогли бы разместить больше процессов. Выбор между first-fit и best-fit слабо влияет на величину фрагментации.

В зависимости от суммарного размера памяти и среднего размера процесса эта проблема может быть большей или меньшей. Статистический анализ показывает, что при наличии n блоков пропадает n/2 блоков, то есть 1/3 памяти! Это известное 50% правило (два соседних свободных участка в отличие от двух соседних процессов могут быть объединены в один).

Одно из решений проблемы внешней фрагментации - разрешить адресному пространству процесса не быть непрерывным, что разрешает выделять процессу память в любых доступных местах. Один из способов реализации такого решения - это paging , используемый во многих современных ОС (будет рассмотрен ниже).

Другим способом борьбы с внешней фрагментацией является *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.