Абстракция памяти в ОС

Память представляет собой очень важный ресурс, требующий четкого управления.

В наши дни объем памяти среднего домашнего ПК в десятки тысяч раз превышает ресурсы первых IBM начала 60-х годов, несмотря на это программы увеличиваются в размерах, стремясь заполнить всю память, доступную для их размещения.

Историческим путем была разработана концепция иерархии памяти, согласно которой компьютеры обладают:

- несколькими мегабайтами очень быстродействующей, дорогой и энергозависимой кэш-памяти;
- несколькими гигабайтами памяти, средней как по скорости, так и по цене;
- несколькими терабайтами памяти на довольно медленных, сравнительно дешевых дисковых накопителях,
- сменными накопителями, таких как CD, DVD, флеш-устройства USB.

Превратить эту иерархию в абстракцию, то есть в удобную модель, а затем управлять этой абстракцией – и есть одна из задач операционной системы.

Если очень просто, то абстракция памяти – запрет программам прямого доступа к памяти, и предоставление программам вместо подлинных адресов памяти - фиктивных (определенных операционной системой). Выделение под каждую программу собственного адресного пространства, которое, собственно, и является абстракцией памяти.

Часть операционной системы, которая управляет иерархией памяти (или ее частью), называется **менеджером**, или **диспетчером** памяти. Он предназначен для действенного управления памятью и должен следить за тем, какие части памяти используются, выделять память процессам, которые е ней нуждаются, и освобождать память, когда процессы завершат свою работу.

Рассмотрим несколько возможных реализованных схем.

1. Память без использования абстракций

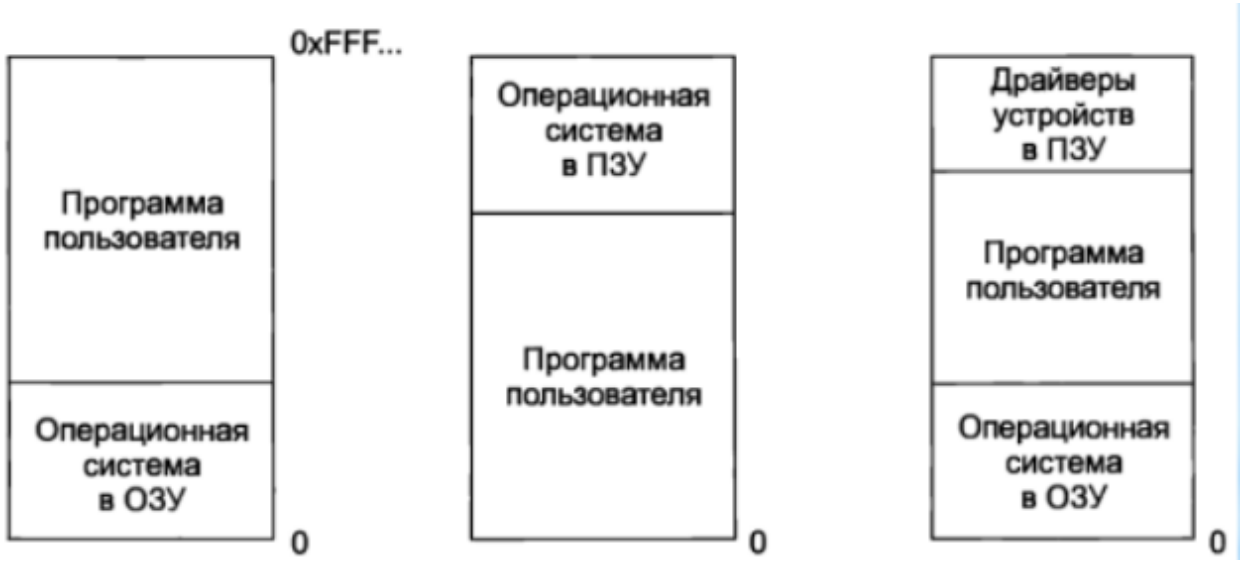
Ранние универсальные машины (до 1960 года), ранние мини компьютеры (до 1970 года) и ранние персональные компьютеры (до 1980 года) не использовали абстракции памяти. Каждая программа просто видела физическую память. Когда программа выполняла следующую команду MOV REGISTER1,1000 компьютер просто перемещал содержимое физической ячейки памяти 1000 в REGISTER1. Таким образом, модель памяти, предоставляемая программисту, была простой физической памятью, набором адресов от 0 до некоторого максимального значения, где каждый адрес соответствовал ячейке, содержащей какое-нибудь количество бит, которое обычно равнялось восьми.

Содержание в памяти сразу двух работающих программ не представлялось возможным.

Если первая программа, к примеру, записывала новое значение в ячейку 2000, то она тем самым стирала то значение, которое сохранялось там второй программой. Работа становилась невозможной, и обе программы практически сразу же давали сбой.

Даже в условиях, когда в качестве модели памяти выступает сама физическая память, возможны несколько вариантов использования памяти.

Три из них показаны на рисунке.



а **б** **в**

ОС может (рис. а) размещаться в нижней части адресов, в оперативном запоминающем устройстве (ОЗУ), или, по-другому, в памяти с произвольным доступом – RAM (Random Access Memory).

ОС может размещаться также в постоянном запоминающем устройстве (ПЗУ), или, иначе, в ROM (ReadOnly Memory), в верхних адресах памяти (рис. б).

Или же драйверы устройств могут быть в верхних адресах памяти, в ПЗУ, а остальная часть системы – в ОЗУ, в самом низу (рис. в).

Первая модель прежде использовалась на универсальных машинах и мини-компьютерах, а на других машинах – довольно редко.

Вторая модель использовалась на некоторых КПК и встроенных системах.

Третья модель использовалась на ранних персональных компьютерах (например, на тех, которые работали под управлением MS-DOS), где часть системы, размещавшаяся в ПЗУ, называлась базовой системой ввода-вывода – BIOS (Basic Input Output System).

Недостаток моделей, изображенных на рис. а и в, заключается в том, что ошибка в программе пользователя может затереть операционную систему, и, возможно, с весьма пагубными последствиями.

Вторая схема предполагала прибавление постоянного значения к каждому адресу физической памяти. При всей исправности работы этого механизма он был не самым универсальным решением, и к тому же замедлял загрузку. Более того, это решение требовало дополнительной информации обо всех исполняемых

программах, сообщающей, в каких словах содержатся, а в каких не содержатся перемещаемые адреса.

2 Абстракция памяти: адресные пространства

Предпосылки

– если пользовательские программы могут обращаться к каждому байту памяти, они легко могут преднамеренно или случайно испортить операционную систему, раздробить ее код и довести до остановки работы

– довольно сложно организовать одновременную (поочередную, если имеется лишь один центральный процессор) работу нескольких программ. На персональных компьютерах вполне естественно наличие нескольких одновременно открытых программ (текстовый процессор, программа электронной почты, веб-браузер), с одной из которых в данный момент взаимодействует пользователь, а работа других возобновляется щелчком мыши. Этого трудно достичь при отсутствии абстракций на основе физической памяти.

Понятие адресного пространства создает своеобразную абстрактную память, в которой существуют программы.

Адресное пространство – это набор адресов, который может быть использован процессом для обращения к памяти.

У каждого процесса имеется свое собственное адресное пространство, независимое от того адресного пространства, которое принадлежит другим процессам (за исключением тех особых обстоятельств, при которых процессам требуется совместное использование их адресных пространств).

Примеры адресных пространств:

- В США и многих других странах местный телефонный номер состоит обычно из семизначного номера. Поэтому адресное пространство телефонных номеров простирается от
- 0000000 до 9999999, хотя некоторые номера, к примеру, те, что начинаются с 000, не используются.
- С ростом количества сотовых телефонов, модемов и факсов это пространство стало слишком тесным, а в этом случае необходимо использовать больше цифр.
- Адресное пространство портов ввода-вывода процессора Pentium простирается от 0 до 16 383.
- Протокол IPv4 обращается к 32-разрядным номерам, поэтому его адресное пространство простирается от 0 до 232 - 1 (опять-таки с некоторым количеством зарезервированных номеров).

Классическое решение заключается в оснащении каждого центрального процессора двумя специальными аппаратными регистрами, которые обычно называются **базовым** и **ограничительным** регистрами.

При использовании этих регистров программы загружаются в последовательно расположенные свободные области памяти без модификации адресов в процессе загрузки.

При запуске процесса в базовый регистр загружается физический адрес, с которого начинается размещение программы в памяти, а в ограничительный регистр загружается длина программы.

Недостатком перемещений с использованием базовых и ограничительных регистров является необходимость применения операций сложения и сравнения к каждой ссылке на ячейку памяти. Сравнение может осуществляться довольно быстро, но сложение является слишком медленной операцией из-за затрат времени на вспомогательный сигнал переноса, если, конечно, не используются специальные сумматоры.

На практике суммарный объем оперативной памяти, необходимый для размещения всех процессов, часто значительно превышает имеющийся объем ОЗУ.

3. Абстракция памяти: свопинг

Для преодоления перегрузки памяти были разработаны два основных подхода.

1. Самый простой из них, называемый **свопингом**[1], заключается в размещении в памяти всего процесса целиком, в запуске его на некоторое время, а затем в сбросе его на диск. Бездействующие процессы большую часть времени хранятся на диске и в нерабочем состоянии не занимают пространство оперативной памяти.
2. Второй подход называется **виртуальной памятью**, он позволяет программам запускаться даже в том случае, если они находятся в оперативной памяти лишь частично.

Недостатки

По сути, недостаток только один: очень медленная скорость. Если рабочий процесс помещается в файл подкачки, то его работоспособность резко снижается. Дело в том, что скорость чтения данных с жесткого диска в несколько десятков раз меньше, чем из флеш-памяти физической "оперативки". Потому и скорость работы процесса будет никакой. Возможны различные тормоза и глюки. Но это случается редко, так как в файл подкачки обычно помещаются процессы, которые не работают в данный конкретный момент времени.

Также весьма распространена проблема с твердотельными накопителями. Если разместить файл подкачки на SSD, то свопинг быстро убьет его. Ресурс такого диска все еще ограничен. Потому и рекомендуется размещать файлы подкачки исключительно на механических жестких дисках. У них ресурс возможных операций куда больше, чем у твердотельных накопителей.

---

[1] Свопинг в мире акций. Таким термином брокеры на биржах обозначали быстрый обмен акциями. На «Форексе» тоже есть такое понятие, но там оно обозначает быструю продажу всех акций.

Свопинг в компьютерном мире. В мире ПК и ноутбуков этим термином обзывают процесс освобождения части оперативной памяти и перенесение данных в заранее созданный файл подкачки на жестком диске.