Организация памяти. Виртуальная память.

Выполнил студент группы ИС-142 Наумов Алексей.

Небольшое отступление: Размер компьютерных программ растет быстрее, чем объем памяти!

Мечта программиста: иметь предоставленную только ему неограниченную по объему и скорости работы ДЕШЕВУЮ память, которая к тому же не теряет своего содержимого при отключении питания.

Концепция иерархии памяти, согласно которой компьютеры обладают:

• несколькими мегабайтами очень быстродействующей, дорогой и энергозависимой кэш-памяти,

• несколькими гигабайтами памяти (ОЗУ), средней как по скорости, так и по цене,

• несколькими терабайтами памяти на довольно медленных, сравнительно дешевых дисковых накопителях,

• сменными накопителями, CD(DVD)-диски и флешустройства USB.

Превратить эту иерархию в абстракцию, то есть в удобную модель, а затем управлять этой абстракцией — и есть задача операционной системы.

Часть операционной системы, которая управляет иерархией памяти (или ее частью), называется менеджером, или диспетчером, памяти.

Память без использований 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 REGISTER!.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.

Три простых способа организации памяти при наличии операционной системы и одного пользовательского процесса:

Запуск нескольких программ без абстракций памяти (Вариант 1) Для одновременного запуска нескольких программ операционная система должна:

1. сохранить все текущее содержимое памяти в файле на диске, а затем

2. загрузить и запустить следующую программу. Поскольку одновременно в памяти присутствует только одна программа, конфликтов не возникает. Эта концепция называется заменой данных (свопинг)

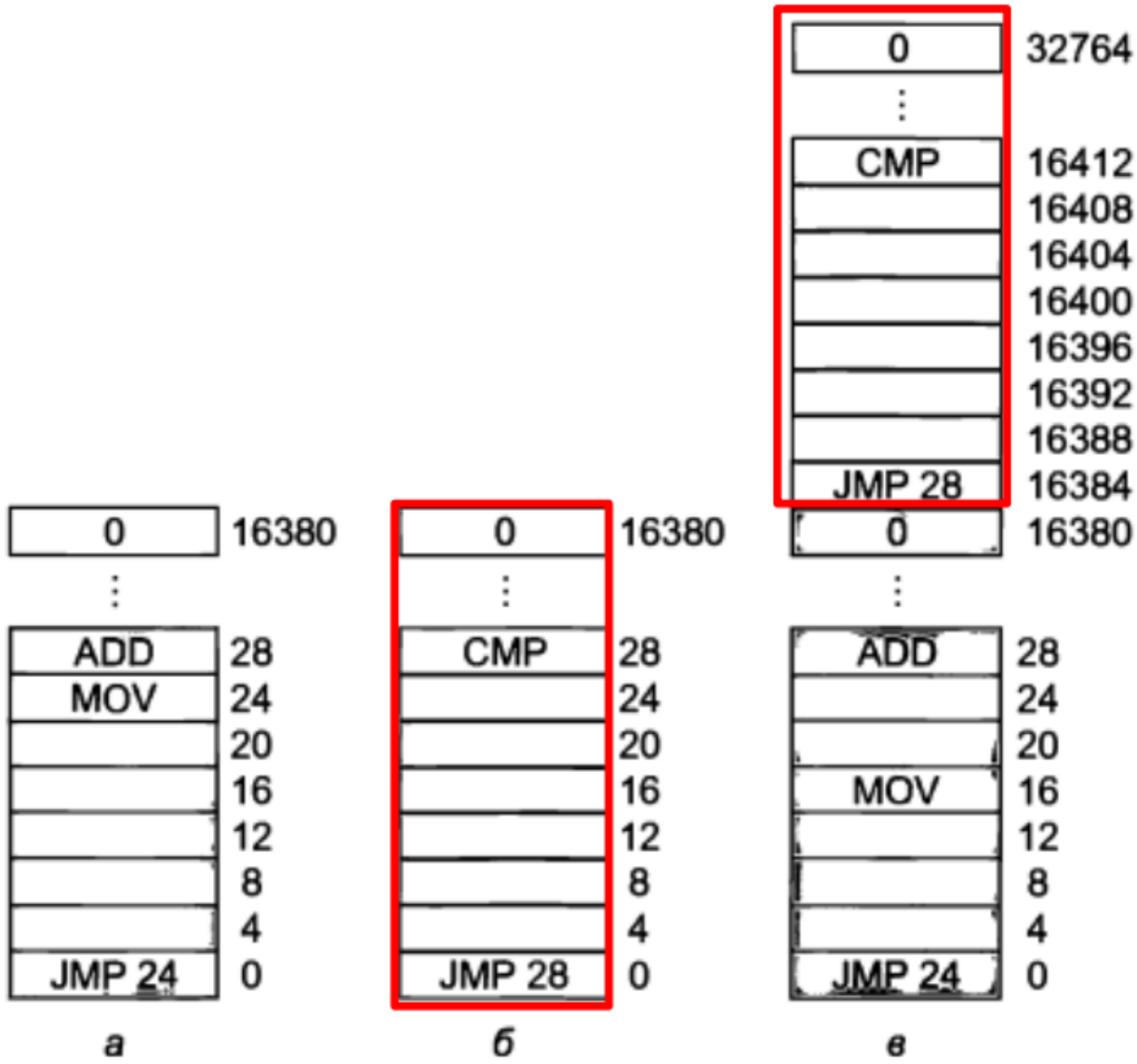
Запуск нескольких программ без абстракций памяти (Вариант 2) Наличие специального дополнительного оборудования позволяет осуществлять параллельный запуск нескольких программ без использования свопинга.

IBM 360.
Память делилась на блоки по 2 Кбайта, каждому из которых присваивался 4-битный защитный ключ, содержащийся в специальных регистрах за пределами центрального процессора. Машине с объемом памяти в 1 Мбайт нужно было иметь лишь 512 таких 4-битных регистров, и все хранилище ключей занимало в итоге 256 байт памяти.

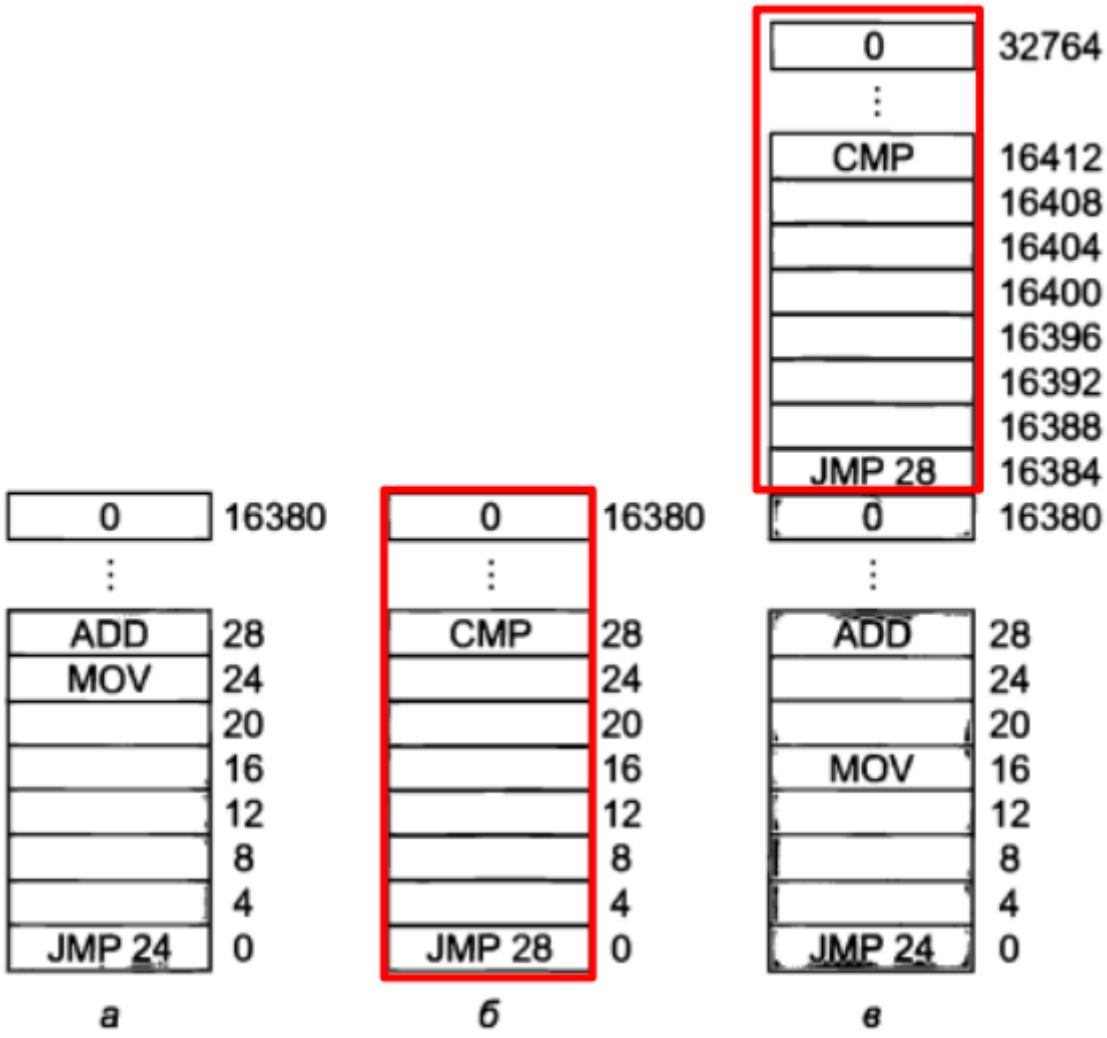
Слово состояния программы — PSW (Program Status Word) также содержало 4-битный ключ. Аппаратное обеспечение IBM 360 перехватывало любую попытку запущенного процесса получить доступ к памяти с ключом защиты, отличающимся от ключа PSW. Поскольку изменить ключи защиты могла только операционная система, пользовательские процессы были защищены

от вмешательства в работу друг друга и в работу самой операционной системы.

Запуск нескольких программ без абстракций памяти (Вариант 2) Проблема абсолютного адреса физической памяти!



Запуск нескольких программ без абстракций памяти (Вариант 2) Проблема абсолютного адреса физической памяти!



Решение: технология статического перемещения. Она работала следующим образом: когда программа загружалась с адреса 16384, в процессе загрузки к каждому адресу в программе прибавлялось постоянное значение 16384.

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:

• В США и многих других странах местный телефонный номер состоит обычно из семизначного номера. Поэтому адресное пространство телефонных номеров простирается от

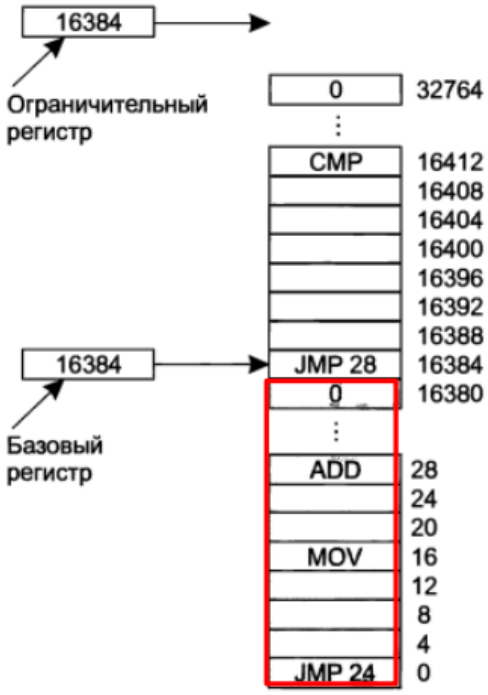
• 0000000 до 9999999, хотя некоторые номера, к примеру, те, что начинаются с 000, не используются.

• 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).

Базовый и ограничительный регистры Динамическое перераспределения памяти. При этом адресное пространство каждого процесса проецируется на различные части физической памяти. Классическое решение заключается в оснащении каждого центрального процессора двумя специальными аппаратными регистрами, которые обычно называются **базовым и ограничительным регистрами.**



Базовый и ограничительный регистры

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.

Базовый и ограничительный регистры

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.

Абстракция памяти: свопинг

Если у компьютера достаточный объем памяти для размещения всех процессов, то все рассмотренные до сих пор схемы будут в той или иной степени работоспособны. Но на практике суммарный объем оперативной памяти, необходимый для размещения всех процессов, зачастую значительно превышает имеющийся объем ОЗУ.

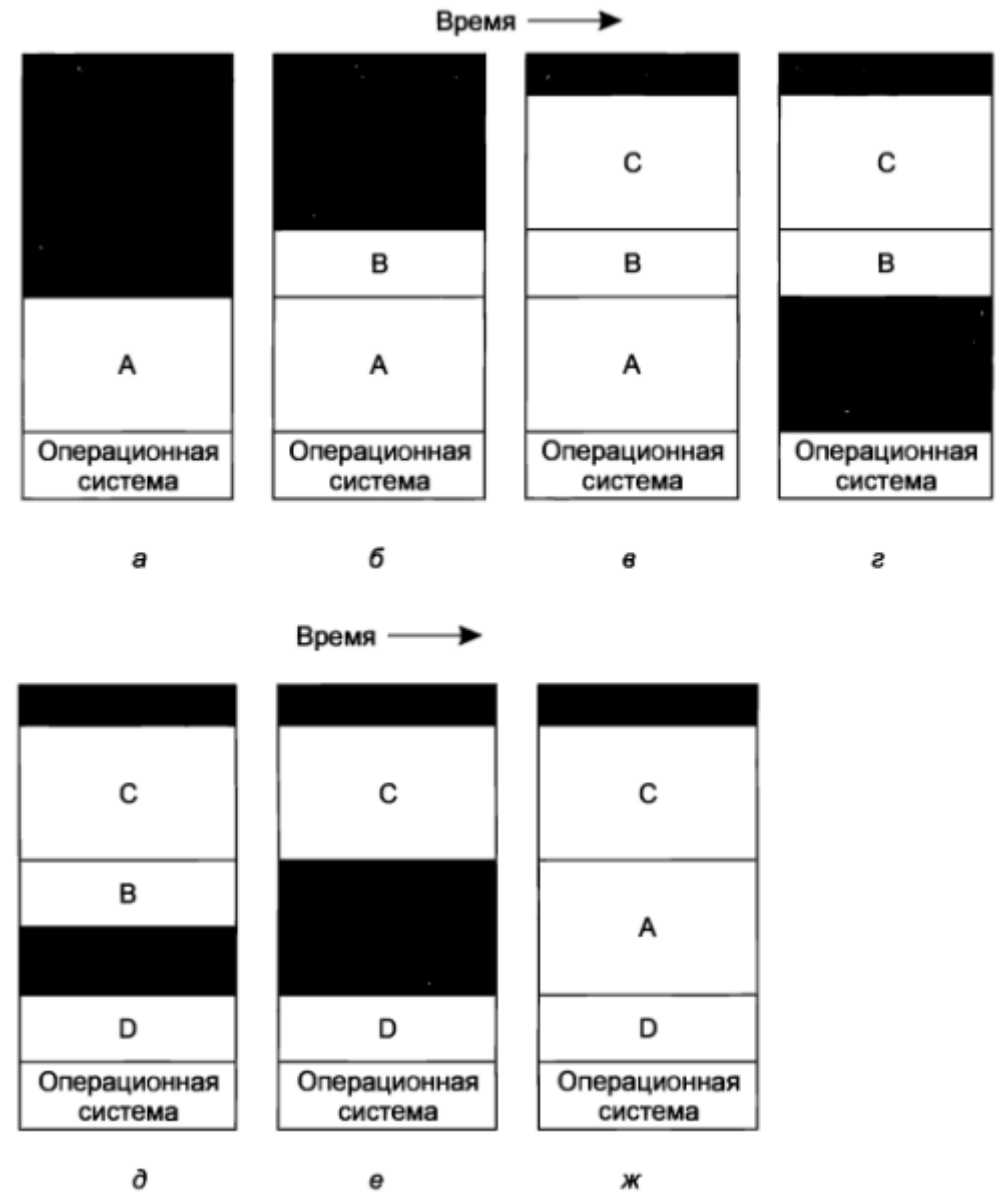
На обычных Windows- или Linux-системах при запуске компьютера могут быть запущены около 40-60 или более процессов.

Постоянное содержание всех процессов в памяти требует огромных объемов и не может быть осуществлено при дефиците памяти.

Для преодоления перегрузки памяти были выработаны два основных подхода.

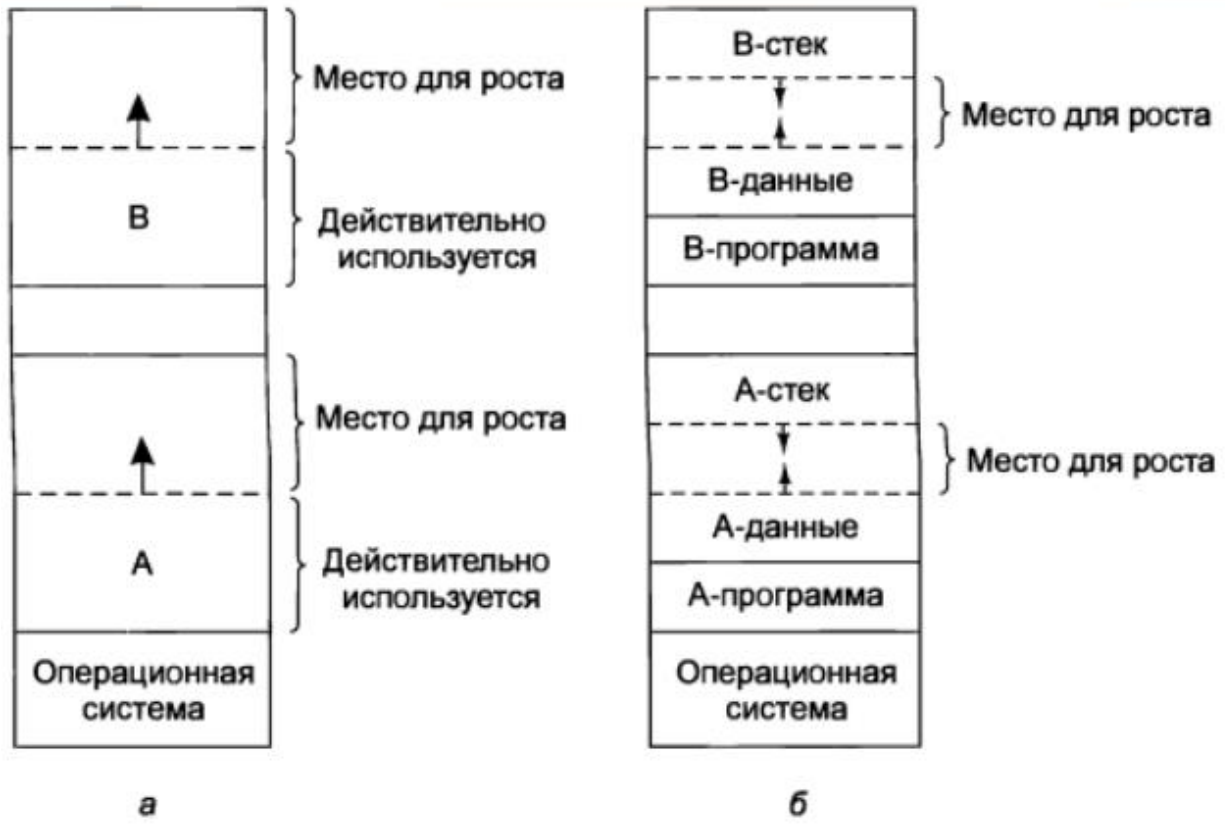
1. Самый простой из них, называемый **свопингом**, заключается в размещении в памяти всего процесса целиком, в запуске его на некоторое время, а затем в сбросе его на диск. Бездействующие процессы большую часть времени хранятся на диске и в нерабочем состоянии не занимают пространство оперативной памяти.

2. Второй подход называется **виртуальной памятью**, он позволяет программам запускаться даже в том случае, если они находятся в оперативной памяти лишь частично.

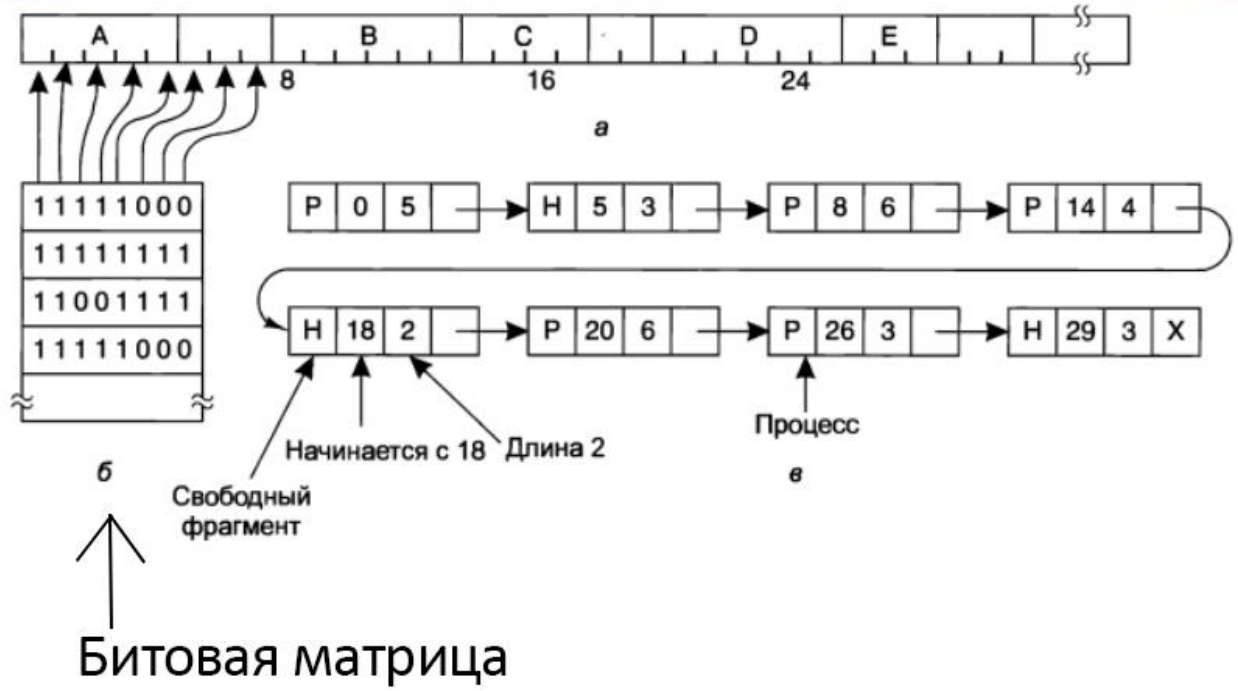


Если предполагается, что большинство процессов по мере выполнения будут разрастаться, то будет лучше распределять небольшой объем дополнительной памяти при каждой загрузке из области свопинга на диске в память или перемещении процесса, чтобы сократить потери, связанные со свопингом или перемещением процессов, которые больше не помещаются в отведенной им памяти.

Свопингу на диск должна подвергаться только реально задействованная память, копировать при этом еще и дополнительно выделенную память будет слишком расточительно.



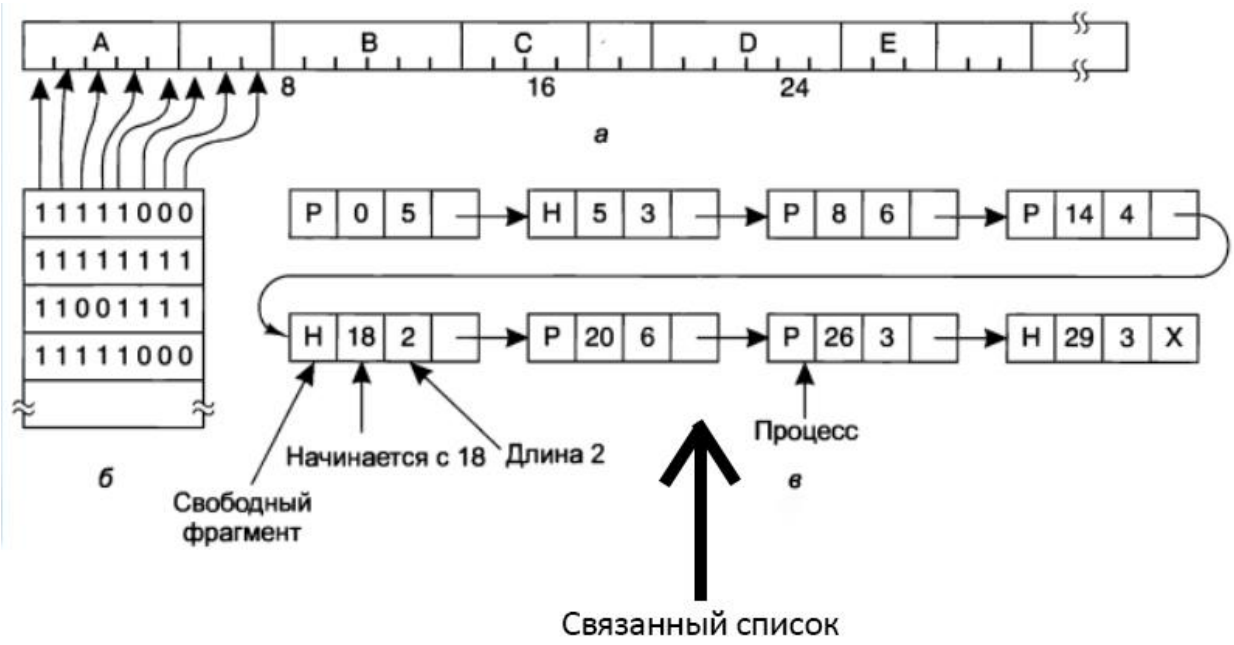
Абстракция памяти: управление свободной памятью



Битовая матрица предоставляет довольно простой способ отслеживания слов памяти в фиксированном объеме памяти, поскольку ее размер зависит только от размера памяти и размера единичного блока памяти.

Битовая матрица (Недостатки):

При решении поместить в память процесс, занимающий k единичных блоков, диспетчер памяти должен искать в битовой матрице непрерывную последовательность нулевых битов. Поиск в битовой матрице последовательности заданной длины — довольно медленная операция (поскольку последовательность может пересекать границы слов в матрице), и это обстоятельство служит аргументом против применения битовых матриц.



Связанный список - Когда процессы и пустые пространства содержатся в списке отсортированными по адресам, то для выделения памяти новому создаваемому процессу (или существующему процессу, загружаемому в результате свопинга с диска) могут быть использованы несколько алгоритмов.

Связанный список (Алгоритмы):

1)Диспетчер памяти знает, сколько памяти нужно выделить. **Простейший алгоритм называется первое подходящее.**

Диспетчер памяти сканирует список сегментов до тех пор, пока не найдет пустое пространство подходящего размера. Затем пустое пространство разбивается на две части: одна для процесса и одна для неиспользуемой памяти, за исключением того статистически маловероятного случая, когда процесс в точности помещается в пустое пространство.

«Первое подходящее» — это быстрый алгоритм, поскольку поиск ведется с наименьшими затратами времени.

2)Диспетчер памяти знает, сколько памяти нужно выделить. **Алгоритм следующее подходящее**

работает так же, как и «первое подходящее», за исключением того, что отслеживает свое местоположение, как только находит подходящее пустое пространство. При следующем вызове для поиска пустого пространства он начинает поиск в списке с того места, на котором остановился в прошлый раз, а не приступает к поиску с самого начала, как при работе алгоритма «первое подходящее».

Его производительность несколько хуже, чем алгоритма «первое подходящее».

3)Диспетчер памяти знает, сколько памяти нужно выделить. **Алгоритм — наиболее подходящее**.

При нем поиск ведется по всему списку от начала до конца и выбирается наименьшее соответствующее пустое пространство. Вместо того чтобы разбивать большое пустое пространство, которое может пригодиться чуть позже, алгоритм «наиболее подходящее» пытается подыскать пустое пространство, близкое по размеру к необходимому, чтобы наилучшим образом соответствовать запросу и имеющимся пустым пространствам.

Алгоритм «наиболее подходящее» работает медленнее, чем «первое подходящее», поскольку он должен при каждом вызове вести поиск по всему списку. Как ни странно, но его применение приводит к более расточительному использованию памяти, чем при использовании алгоритмов «первое подходящее» и «следующее подходящее», поскольку он стремится заполнить память, оставляя небольшие бесполезные пустые пространства.

4)Диспетчер памяти знает, сколько памяти нужно выделить. **Алгоритм — наименее подходящее.**

попытка обойти проблему разбиения практически точно подходящих пространств памяти на память, отводимую под процесс, и на небольшие пустые пространства, то есть к неизменному выбору самого большого подходящего пустого пространства, чтобы вновь образующееся пустое пространство было достаточно большим для дальнейшего использования.

Моделирование показало, что применение алгоритма «наименее подходящее» также далеко не самая лучшая идея.

Виртуальная память

Решения проблемы программ, превышающих по объему размер имеющейся памяти.

1)разбивать программы на небольшие части, называемые оверлеями. При запуске программы в память загружался только администратор оверлейной загрузки, который тут же загружал и запускал оверлей с порядковым номером 0. Когда этот оверлей завершал свою работу, он мог сообщить администратору загрузки оверлеев о необходимости загрузки оверлея 1, либо выше оверлея 0, находящегося в памяти (если для него было достаточно пространства), либо поверх оверлея 0 (если памяти не хватало).

Некоторые оверлейные системы имели довольно сложное устройство, позволяя одновременно находиться в памяти множеству оверлеев. Оверлеи хранились на диске, и их свопинг с диска в память и обратно осуществлялся администратором загрузки оверлеев.

Разбиение больших программ на небольшие модульные части было очень трудоемкой, скучной и не застрахованной от ошибок работой.

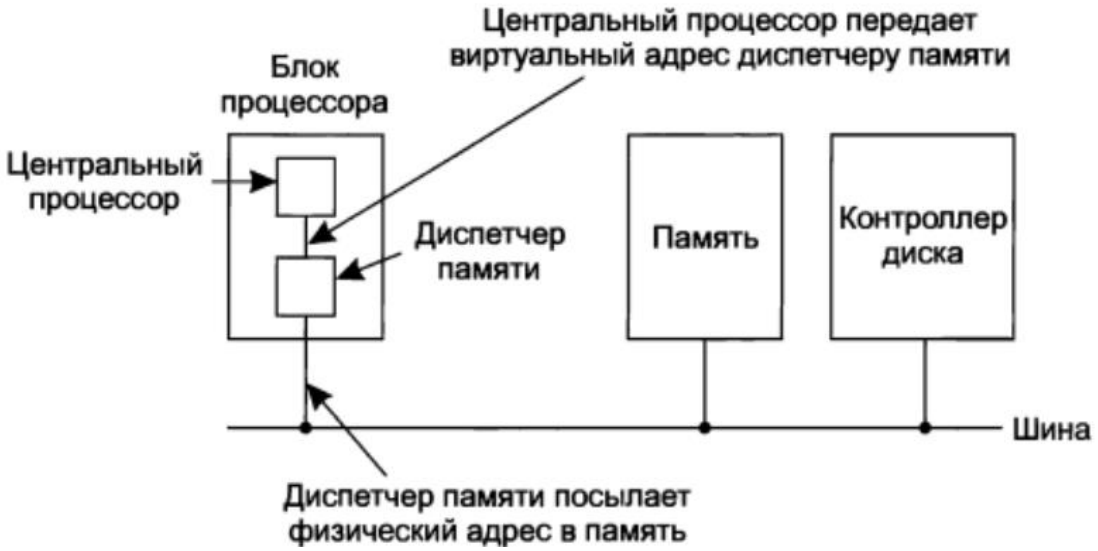
Решения проблемы программ, превышающих по объему размер имеющейся памяти.

2. Способ, позволяющий возложить всю работу по разбиению программы на части на компьютер, (Fotheringham, 1961), стал известен как **виртуальная память.**

В основе виртуальной памяти лежит идея, что у каждой программы имеется свое собственное адресное пространство, которое разбивается на участки, называемые страницами. Каждая страница представляет собой непрерывный диапазон адресов. Эти страницы отображаются на физическую память, но для запуска программы присутствие в памяти всех страниц не обязательно.

Когда программа ссылается на часть своего адресного пространства, находящегося в физической памяти, аппаратное обеспечение осуществляет необходимое отображение на лету.

Когда программа ссылается на часть своего адресного пространства, которое не находится в физической памяти, операционная система предупреждается о том, что необходимо получить недостающую часть и повторно выполнить потерпевшую неудачу команду.



Подкачка. Алгоритмы замещения страниц

Подкачка страниц (англ. Paging; иногда используется термин swapping от swap, /swɔp/) — **один из механизмов виртуальной памяти**, при котором отдельные фрагменты памяти (обычно неактивные) перемещаются из ОЗУ на жѐсткий диск (или другой внешний накопитель, такой как Флешпамять), освобождая ОЗУ для загрузки других активных фрагментов памяти. Такими фрагментами в современных ЭВМ являются страницы памяти.

Временно выгруженные из памяти страницы могут сохраняться на внешних запоминающих устройствах как в файле, так и в специальном разделе на жѐстком диске (partition), называемые соответственно swapфайл и swap-раздел. В случае откачки страниц, соответствующих содержимому какого-либо файла (например, memory-mapped files), они могут удаляться. При запросе такой страницы она может быть считана из оригинального файла.

Когда приложение обратится к откачанной странице, произойдет исключительная ситуация PageFault. Обработчик этого события должен проверить, была ли ранее откачана запрошенная страница, и, если она есть в swap-файле, загрузить ее обратно в память.

Изначально под свопингом понималась выгрузка процесса из оперативной памяти целиком, в результате чего неактивные процессы могли полностью отсутствовать в ОЗУ. При наступлении условий активизации процесса диспетчер памяти загружал образ процесса обратно.

Смысл термина изменился в 60-х годах, когда в операционных системах появилась поддержка виртуальной памяти: под свопингом стали понимать загрузку и выгрузку отдельных страниц.

При выделении места для новой страницы бывает 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 «мѐртвая» страница, которая больше не потребуется кому-либо (например, относится к завершѐнному 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 определѐнное 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:

FIFO - Удаляются те страницы, доступ к которым производился наиболее давно. Считается, что в последующем к таким страницам будет происходить минимум обращений (FIFO – первая пришла – первая ушла).

Last Recently Used. Удаляются недавно освободившиеся страницы. Подразумеваются страницы только что завершившихся процессов.

Оптимальный алгоритм замещения страниц гласит, что должна быть удалена страница, имеющая пометку с наибольшим значением количества команд до вызова страницы.

Если какая-то страница не будет использоваться на протяжении 8 миллионов команд, а другая какая-нибудь страница не будет использоваться на протяжении 6 миллионов команд, то удаление первой из них приведет к ошибке отсутствия страницы, в результате которой она будет снова выбрана с диска в самом отдаленном будущем.

Варианты алгоритмов:

- Алгоритм «второй шанс»
- Алгоритм «часы»
- Алгоритм замещения наименее востребованной страницы
- Алгоритм нечастого востребования — NFU (Not Frequently Used).
- Алгоритм «Рабочий набор» Алгоритм WSCIock

Алгоритм	Особенности
Оптимальный	Не может быть реализован, но полезен в качестве оценочного критерия
NRU (Not Recently Used) — алгоритм исключения недавно использовавшейся страницы	Является довольно грубым приближением к алгоритму LRU
FIFO (First-In, First-Out) — алгоритм «первой пришла, первой и ушла»	Может выгрузить важные страницы
Алгоритм «второй шанс»	Является существенным усовершенствованием алгоритма FIFO
Алгоритм «часы»	Вполне реализуемый алгоритм
LRU (Least Recently Used) — алгоритм замещения наименее востребованной страницы	Очень хороший, но труднореализуемый во всех тонкостях алгоритм
NFU (Not Frequently Used) — алгоритм нечастого востребования	Является довольно грубым приближением к алгоритму LRU
Алгоритм старения	Вполне эффективный алгоритм, являющийся неплохим приближением к алгоритму LRU
Алгоритм рабочего набора	Весьма затратный для реализации алгоритм
WSClock	Вполне эффективный алгоритм