# Лекция 11. Память. Основные понятия. Характеристики, назначение. Цели и задачи управления памятью. Функции ОС по управлению памятью. Кэш. Пространственная и временная локальности.

**Функции ОС по управлению памятью**

Под памятью (*memory*) в данном случае подразумевается оперативная (основная) *память* компьютера. В однопрограммных операционных системах основная *память* разделяется на две части. Одна часть для операционной системы , а вторая – для выполняющейся в текущий момент времени программы. В многопрограммных ОС "пользовательская" часть памяти – важнейший *ресурс* вычислительной системы – должна быть распределена для размещения нескольких процессов, в том числе процессов ОС. Эта задача распределения выполняется операционной системой динамически специальной подсистемой управления памятью .

Эффективное *управление памятью* жизненно важно для многозадачных систем. Если в памяти будет находиться небольшое число процессов, то значительную часть времени процессы будут находиться в состоянии ожидания ввода-вывода и *загрузка* процессора будет низкой.

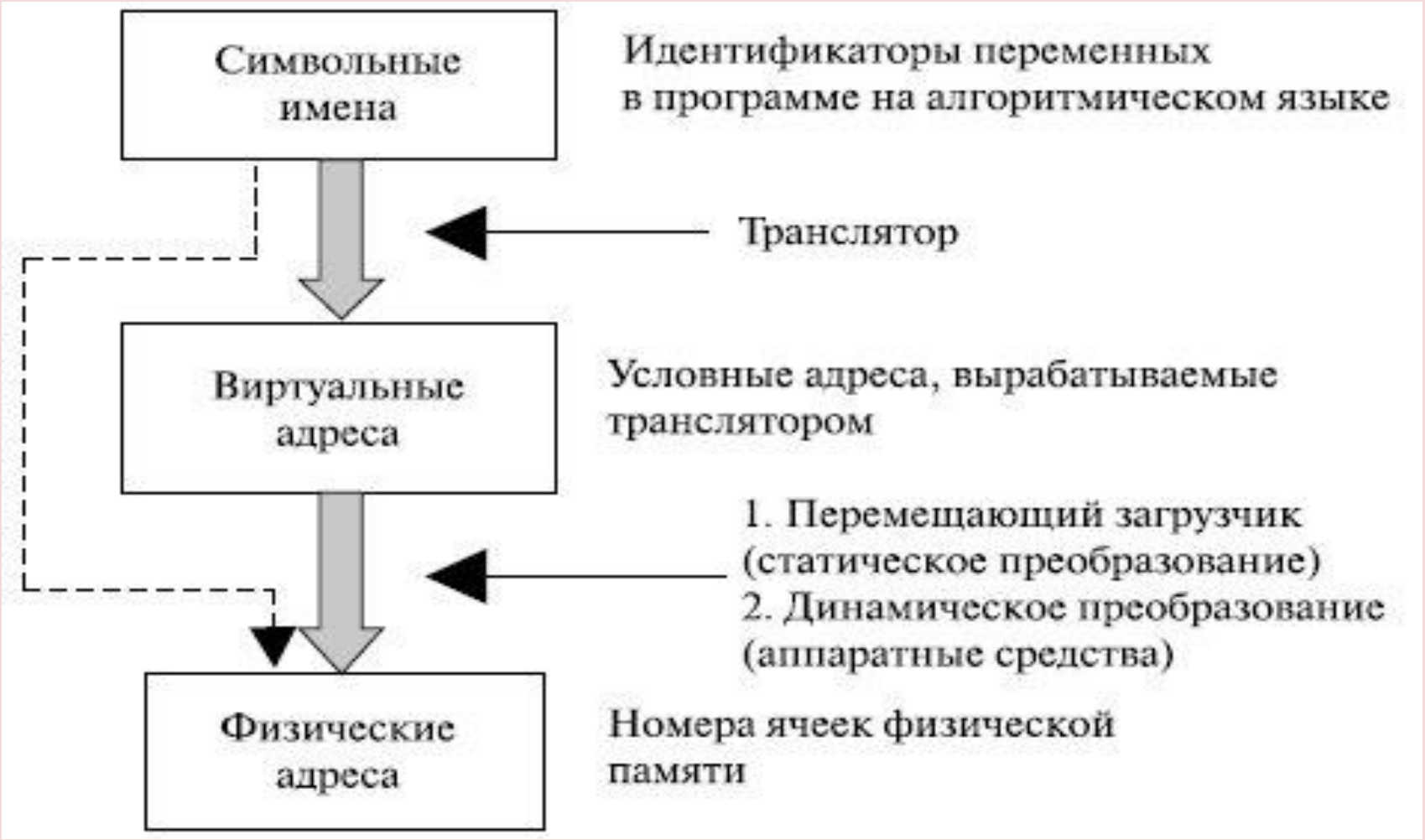
В ранних ОС *управление памятью* сводилось просто к загрузке программы и ее данных из некоторого внешнего накопителя (перфоленты, магнитной ленты или магнитного диска) в *ОЗУ*. При этом *память* разделялась между программой и ОС. На рис. 6.3показаны три варианта такой схемы. Первая модель раньше применялась на мэйнфреймах и мини-компьютерах. Вторая схема сейчас используется на некоторых карманных компьютерах и встроенных системах, третья модель была характерна для ранних персональных компьютеров с *MS-DOS*.

Функциями ОС *по* управлению памятью в мультипрограммных системах являются:

- отслеживание (учет) свободной и занятой памяти;
- первоначальное и *динамическое выделение памяти* процессам приложений и самой операционной системе и освобождение памяти по завершении процессов;
- настройка адресов программы на конкретную область физической памяти;
- полное или частичное вытеснение кодов и данных процессов из ОП на диск, когда размеры ОП недостаточны для размещения всех процессов, и возвращение их в ОП;
- защита памяти, выделенной процессу, от возможных вмешательств со стороны других процессов;
- *дефрагментация памяти*.

Перечисленные функции особого пояснения не требуют, остановимся только на задаче преобразования адресов программы при ее загрузке в ОП.

Для идентификации переменных и команд на разных этапах жизненного *цикла* программы используются символьные имена, виртуальные (математические, условные, логические – все это синонимы) и физические адреса (рис. 6.4).



Символьные имена присваивает *пользователь* при написании программ на алгоритмическом языке или ассемблере.

Виртуальные адреса вырабатывает *транслятор*, переводящий программу на *машинный язык*. Поскольку во время трансляции неизвестно, в какое*место* оперативной памяти будет загружена *программа*, *транслятор* присваивает переменным и командам виртуальные (условные) адреса, считая *по* умолчанию, что начальным адресом программы будет нулевой *адрес*.

Физические адреса соответствуют номерам ячеек оперативной памяти, где в действительности будут расположены переменные и команды.

Совокупность виртуальных адресов процесса называется виртуальным адресным пространством. *Диапазон* адресов виртуального пространства у всех процессов один и тот же и определяется разрядностью адреса процессора (для Pentium *адресное пространство*составляет объем, равный $2^{32}$ *байт*, с диапазоном адресов от $0000.0000_{16}$ до $FFFF.FFFF_{16}$).

Существует два принципиально отличающихся подхода к преобразованию виртуальных адресов в физические. В первом случае такое преобразование выполняется один раз для каждого процесса во время начальной загрузки программы в *память*. Преобразование осуществляет перемещающий *загрузчик* на основании имеющихся у него данных о начальном адресе физической памяти, в которую предстоит загружать программу, а также информации, предоставляемой транслятором об адресно-зависимых элементах программы.

Второй способ заключается в том, что *программа* загружается в *память* в виртуальных адресах. Во *время выполнения* программы при каждом обращении к памяти *операционная система* преобразует виртуальные адреса в физические.

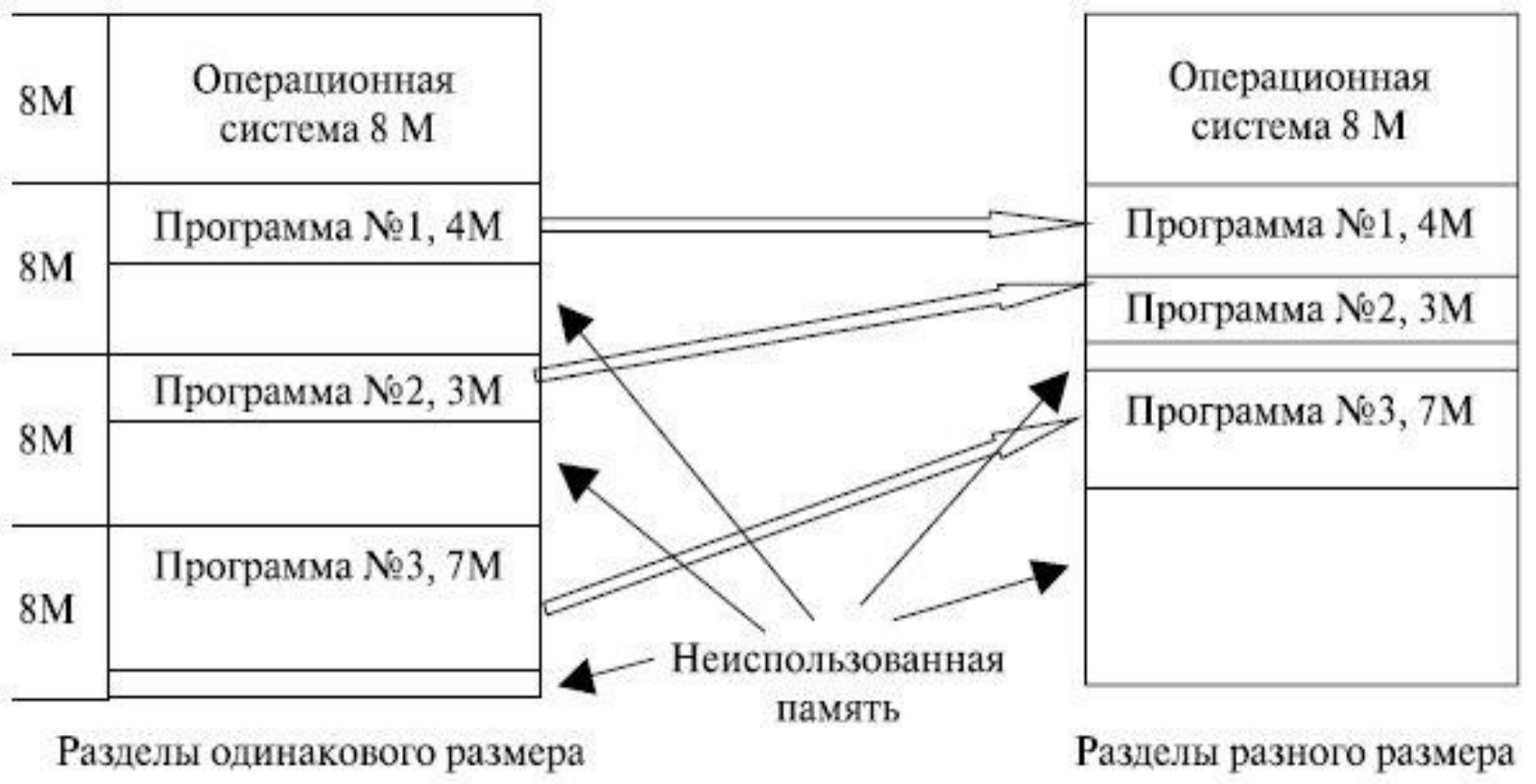
**Распределение памяти**

Существует ряд базовых вопросов управления памятью, которые в различных ОС решаются *по*-разному. Например, следует ли назначать каждому процессу одну непрерывную область физической памяти или можно выделять *память* участками? Должны ли*сегменты* программы, загруженные в *память*, находиться на одном месте в течение всего периода выполнения процесса или их можно время от времени сдвигать? Что делать, если *сегменты* программы не помещаются в имеющуюся *память*? Как сократить*затраты* ресурсов системы на *управление памятью*? Имеется и ряд других не менее интересных проблем управления памятью.

Ниже приводится классификация методов распределения памяти, в которой выделено два класса методов – с перемещением сегментов процессов между ОП и ВП (диском) и без перемещения, т.е. без привлечения внешней памяти (рис. 6.5). Данная классификация учитывает только основные признаки методов. Для каждого метода может быть использовано несколько различных алгоритмов его реализации.


Методы распределения памяти
Без использования внешней памяти
С использованием внешней памяти
Фиксированными разделами
Динамическими разделами
Перемещаемыми разделами
Страничное распределение
Сегментное распределение
Сегментно-страничное распределение

На рис. 6.6 показаны два примера фиксированного распределения. Одна возможность состоит в использовании разделов одинакового размера. В этом случае любой процесс, размер которого не превышает размера раздела, может быть загружен в любой доступный раздел. Если все *разделы* заняты и нет ни одного процесса в состоянии готовности или работы, ОС может выгрузить процесс из любого раздела и загрузить другой процесс, обеспечивая тем самым *процессор* работой.



При использовании разделов с одинаковым размером имеются две проблемы.

•Программа может быть слишком велика для размещения в разделе. В этом случае программист должен разрабатывать программу, использующую оверлеи, чтобы в любой момент времени требовался только один раздел памяти. Когда требуется модуль, отсутствующий в данный момент в ОП, пользовательская программа должна сама его загрузить в раздел памяти программы. Таким образом, в данном случае управление памятью во многом возлагается на программиста.

•Использование ОП крайне неэффективно. Любая программа, независимо от ее размера, занимает раздел целиком. При этом могут оставаться неиспользованные участки памяти большого размера. Этот феномен появления неиспользованной памяти называется внутренней фрагментацией (internal fragmentation).

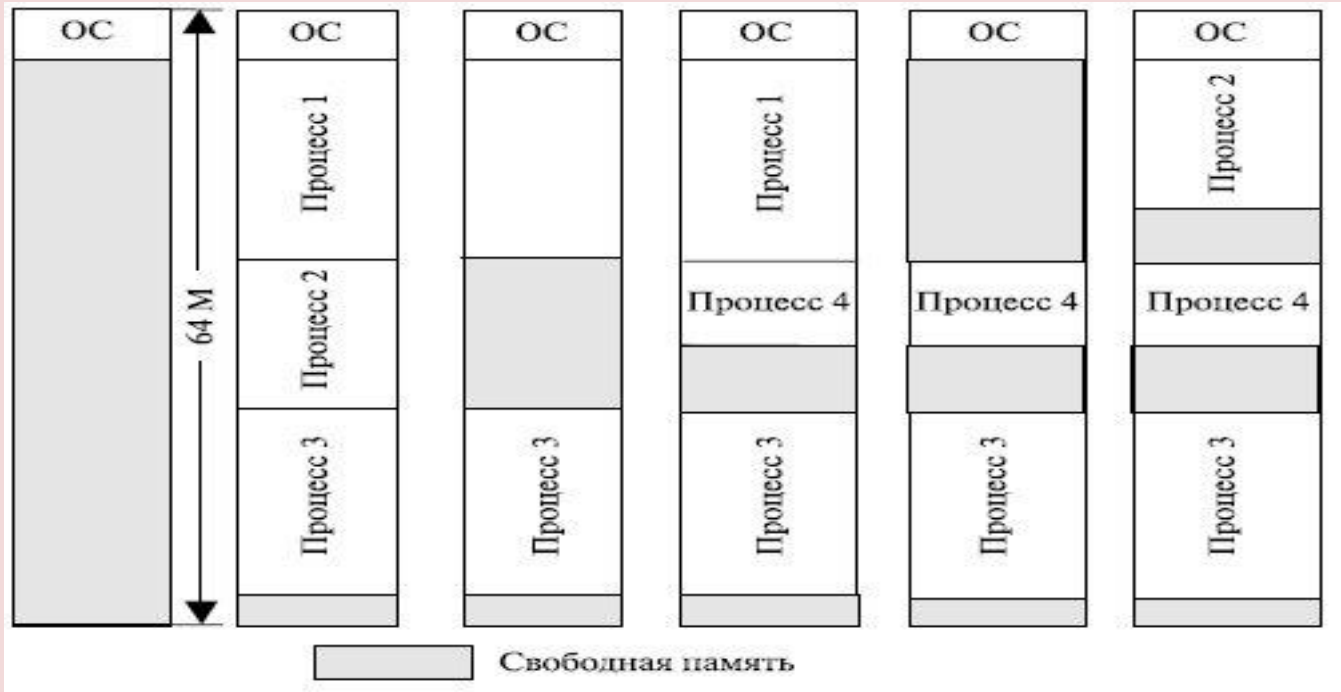
Бороться с этими трудностями (хотя и не устранить полностью) можно посредством использования разделов разных размеров. В этом случае *программа* размером до 8 Мбайт может обойтись без оверлеев, а *разделы* малого размера позволяют уменьшить внутреннюю фрагментацию при загрузке небольших программ.

В целом можно отметить, что схемы с *фиксированными разделами* относительно просты, предъявляют минимальные требования к операционной системе; накладные *расходы* работы процессора на *распределение памяти* невелики. Однако у этих схем имеются серьезные недостатки.

- Количество разделов, определенное в момент генерации системы, ограничивает количество активных процессов (т.е. уровень мультипрограммирования).
- Поскольку размеры разделов устанавливаются заранее во время генерации системы, небольшие задания приводят к неэффективному использованию памяти. В средах, где заранее известны потребности в памяти всех задач, применение рассмотренной схемы может быть оправдано, но в большинстве случаев эффективность этой технологии крайне низка.

Для преодоления сложностей, связанных с фиксированным распределением, был разработан альтернативный подход, известный как динамическое распределение. В свое время этот подход был применен фирмой *IBM* в операционной системе для мэйнфреймов в OS/MVT (*мультипрограммирование* с переменным числом задач – *Multiprogramming With* a Variable number of Tasks). Позже этот же подход к распределению памяти использован в ОС ЕС ЭВМ .

При динамическом распределении образуется переменное количество разделов переменной длины. При размещении процесса в основной памяти для него выделяется строго необходимое количество памяти. В качестве примера рассмотрим использование 64 Мбайт (рис. 6.7) основной памяти. Изначально вся *память* пуста, за исключением области, задействованной ОС. Первые три процесса загружаются в *память*, начиная с адреса, где заканчивается ОС, и используют столько памяти, сколько требуется данному процессу. После этого в конце ОП остается свободный участок памяти, слишком малый для размещения четвертого процесса. В некоторый момент времени все процессы в памяти оказываются неактивными, и *операционная система* выгружает второй процесс, после чего остается достаточно памяти для загрузки нового, четвертого процесса.



Перечислим функции операционной системы *по* управлению памятью в этом случае.

•Перемещение всех занятых участков в сторону старших или младших адресов при каждом завершении процесса или для вновь создаваемого процесса в случае отсутствия раздела достаточного размера.

•Коррекция таблиц свободных и занятых областей.

•Изменение адресов команд и данных, к которым обращаются процессы при их перемещении в памяти, за счет использования*относительной адресации*.

•Аппаратная поддержка процесса динамического преобразования относительных адресов в абсолютные адреса основной памяти.

•Защита памяти, выделяемой процессу, от взаимного влияния других процессов.

Уплотнение может выполняться либо при каждом завершении процесса, либо только тогда, когда для вновь создаваемого процесса нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц свободных и занятых областей, а во втором – реже выполняется процедура сжатия.