|  |  |  |  |
| --- | --- | --- | --- |
| [1](#page3) | [вопрос. Определение операционные системы. Системное и прикладное программное](#page3) | | |
| [обеспечение. Место операционной системы в общей структуре компьютера](#page3) | | | [3](#page3) |
| [2](#page3) | [вопрос. Функции ОС, Операционные среды. Операционные оболочки](#page3) | | [3](#page3) |
| [3](#page4) | [вопрос. Классификация ОС. Однозадачные и многозадачные ОС](#page4) | | [4](#page4) |
| [4](#page4) | [вопрос. Классификация ОС. Микроядерные и макроядерные ОС](#page4) | | [4](#page4) |
| [5](#page5) | [вопрос. Классификация ОС. Специализированные операционные системы](#page5) | | [5](#page5) |
| [6](#page6) | [вопрос. Классификация ОС. Операционные системы общего назначения](#page6) | | [6](#page6) |
| [7](#page6) | [вопрос. Классификация ОС. Системы с асимметричной процессорной обработкой](#page6) | | [6](#page6) |
| [8](#page7) | [вопрос. Классификация ОС. Системы с симметричной процессорной обработкой](#page7) | | [7](#page7) |
| [9](#page7) | [вопрос. Процессы. Граф существования процесса](#page7) | | [7](#page7) |
| [10](#page8) | | [вопрос. Процессы. Классификация процессов](#page8) | [8](#page8) |
| [11 вопрос. Процессы. Программные потоки](#page8) | | | [8](#page8) |
| [12](#page10) | | [вопрос. Управление ресурсами. Состояние процесса (НЕПОЛНЫЙ)](#page10) | [10](#page10) |
| [13](#page11) | | [вопрос. Сохранение состояния процесса (НЕ НАЙДЕН)](#page11) | [11](#page11) |
| [14](#page11) | | [вопрос. Потоки. Средства синхронизации потоков (ССЫЛКА)](#page11) | [11](#page11) |
| [15](#page11) | | [вопрос. Диспетчеризация процессора. Основная цель диспетчеризации процессора](#page11) | |
| [11](#page11) | |  |  |
| [16](#page11) | | [вопрос. Планировщик. Разновидности стратегий, с точки зрения прерывания или](#page11) |  |
| [избежания прерывания процессов, использует планировщик](#page11) | | | [11](#page11) |
| [17](#page12) | | [вопрос. Планировщик. Стратегия без прерывания процессов. Диаграмма Ганта](#page12) | [12](#page12) |
| [18](#page12) | | [вопрос. Планировщик. Стратегия с прерыванием процессов. Диаграмма Ганта](#page12) | [12](#page12) |
| [19](#page13) | | [вопрос. Планировщик. стратегии FCFS и каковы ее недостатки](#page13) | [13](#page13) |
| [20](#page14) | | [вопрос. Планировщик. стратегии SJF (и SRTF) и оптимальность по какому критерию](#page14) | |
| [она обеспечивает](#page14) | | | [14](#page14) |
| [21](#page15) | | [вопрос. Планировщик. Диспетчер. Диспетчеризации по приоритетам](#page15) | [15](#page15) |
| [22](#page16) | | [вопрос. Планировщик. Проблема голодания процессов и каково ее решение в ОС](#page16) | [16](#page16) |
| [23](#page16) | | [вопрос. Стратегии RR, оптимальность по какому критерию она обеспечивает и по](#page16) |  |
| [какому критерию она хуже, чем SJF](#page16) | | | [16](#page16) |
| [24](#page17) | | [вопрос. В чем заключается задача управления памятью? входная очередь заданий. 17](#page17) | |
| [25](#page17) | | [вопрос. Связывание адресов и на каких этапах обработки программы оно может](#page17) |  |
| [выполняться](#page17) | | | [17](#page17) |
| [26](#page18) | | [вопрос. Этапы обработки проходит программа на пути от исходного кода к](#page18) |  |
| [двоичному образу в памяти](#page18) | | | [18](#page18) |
| [27](#page19) | | [вопрос. Адресация с использованием регистра перемещения](#page19) | [19](#page19) |
| [29](#page20) | | [вопрос. Откачка и подкачка. Файл подкачки. организовано смежное распределение](#page20) | |
| [памяти. Схема откачки и подкачки](#page20) | | | [20](#page20) |

|  |  |  |
| --- | --- | --- |
| [30](#page21) | [вопрос. Схема адресации с аппаратной поддержкой регистров перемещения и](#page21) |  |
| [границы](#page21) | | [21](#page21) |
| [31](#page21) | [вопрос. Архитектура трансляции адресов при страничной организации](#page21) | [21](#page21) |
| [33](#page24) | [вопрос. Структура таблицы страниц. Хешированные таблицы страниц](#page24) | [24](#page24) |
| [34](#page25) | [вопрос. Откачка и подкачка. Файл подкачки. Разделяемые страницы](#page25) | [25](#page25) |
| [35](#page26) | [вопрос. Принципы сегментной организации памяти, использования разделяемых](#page26) |  |
| [сегментов](#page26) | | [26](#page26) |
| [36](#page27) | [вопрос. Схема адресации при сегментной организации памяти. Пример сегментной](#page27) | |
| [организации памяти](#page27) | | [27](#page27) |
| [37](#page27) | [вопрос. Понятие файла. Структура файла. Методы доступа к файлам](#page27) | [27](#page27) |
| [38](#page28) | [вопрос. Сетевые топологии. Стратегии маршрутизации](#page28) | [28](#page28) |

**1 вопрос. Определение операционные системы. Системное и прикладное программное обеспечение. Место операционной системы в общей структуре компьютера**

Операционная система – базовое системное программное обеспечение, управляющее работой компьютера и являющееся посредником между аппаратурой, прикладным программным обеспечением и пользователем компьютера.

Перед ОС стоят следующие цели:

* Обеспечение удобства, эффективности, надежности, безопасности выполнения пользовательских программ
* Обеспечение удобства, эффективности, надежности, безопасности использования компьютера
* Обеспечение удобства, эффективности, надежности, безопасности использования сетевых, дисковых и других внешних устройств, подключенных к компьютеру
* Обеспечение безопасности, надежности и защиты данных

Компоненты компьютерной системы:

* Аппаратура (процессор, память, устройства ввода-вывода)
* Операционная система (operating system) – системное программное обеспечение, управляющее использованием аппаратуры компьютера различными программами и пользователями.
* Прикладное программное обеспечение – программы, предназначенные для решения различных классов задач. К ним относятся, в частности, компиляторы, обеспечивающие трансляцию программ с языков программирования, например, C++, в машинный код (команды); системы управления базами данных (СУБД); графические библиотеки, игровые программы, офисные программы. Прикладное программное обеспечение образует следующий, более высокий уровень, по сравнению с операционной системой, и позволяет решать на компьютере различные прикладные и повседневные задачи.
* Пользователи.

Одна из важных функций ОС как раз и состоит в том, чтобы избавить пользователя от большей части такой рутинной работы (например, резервного копирования файлов) и позволить ему сосредоточиться на работе творческой. Другие компьютеры в сети также могут играть роль пользователей ( клиентов ) по отношению к данному компьютеру, выступающему в роли сервера, используемого, например, для хранения файлов или выполнения больших программ.

Пользователям компьютера доступны верхние уровни программного обеспечения – системные и прикладные программы (например, компиляторы, текстовые редакторы, системы управления базами данных). Эти программы взаимодействуют с операционной системой, которая, в свою очередь, управляет работой компьютера.

**2 вопрос. Функции ОС, Операционные среды. Операционные оболочки** Основные функции ОС:

* Исполнение запросов программ (ввод и вывод данных, запуск и остановка других программ, выделение и освобождение дополнительной памяти и др.).
* Загрузка программ в оперативную память и их выполнение.
* Стандартизованный доступ к периферийным устройствам (устройства ввода-вывода).
* Управление оперативной памятью (распределение между процессами, организация виртуальной памяти).
* Управление доступом к данным на энергонезависимых носителях (таких как жёсткий диск, оптические диски и др.), организованным в той или иной файловой системе.
* Обеспечение пользовательского интерфейса.
* Сохранение информации об ошибках системы.

Дополнительные функции ОС:

* Параллельное или псевдопараллельное выполнение задач (многозадачность).
* Эффективное распределение ресурсов вычислительной системы между процессами.
* Разграничение доступа различных процессов к ресурсам.
* Организация надёжных вычислений (невозможности одного вычислительного процесса намеренно или по ошибке повлиять на вычисления в другом процессе), основана на разграничении доступа к ресурсам.
* Взаимодействие между процессами: обмен данными, взаимная синхронизация.
* Защита самой системы, а также пользовательских данных и программ от действий пользователей (злонамеренных или по незнанию) или приложений.
* Многопользовательский режим работы и разграничение прав доступа (см.: аутентификация, авторизация). Операционная среда — это совокупность инструментов, методов их интеграции и приёмов работы с ними,

позволяющая пользователю решать любые задачи в инструментальной области и большинство задач в прикладных областях. Отличие операционной среды от специализированной (например, статистического пакета SPSS) в том, что, во-первых, в операционной среде есть средства решения задач во многих прикладных областях (а не в одной), а во-вторых, если инструмента решения какой-то задачи нет, средствами операционной среды его всегда можно изготовить.

Операционная оболочка - это самый внешний уровень операционной системы. Оболочки содержат в себе язык программирования для управления процессами и файлами, а также запуска и управления другими программами. Оболочка управляет взаимодействием между вами и операционной системой, показывая вам приглашения для ввода, интерпретируя ввод для операционной системы, а затем обрабатывая результирующий вывод операционной системы.

Оболочки операционных систем обеспечивают:

* создание, переименование, копирование, пересылку, удаление и быстрый поиск файла в текущем каталоге диска или на всех дисках компьютера;
* просмотр, создание и сравнение каталогов;
* просмотр, создание и редактирование текстовых файлов;
* архивацию, обновление и разархивацию архивных файлов и просмотр архивов;
* синхронизацию каталогов, расщепление и слияние файлов;
* поддержку связи двух компьютеров через последовательный или параллельный порты;
* форматирование и копирование дискет, смену метки дискеты и метки тома для жестких дисков, а также очистку дисков от ненужных файлов;
* запуск программ.

**3 вопрос. Классификация ОС. Однозадачные и многозадачные ОС**

Операционная система – базовое системное программное обеспечение, управляющее работой компьютера и являющееся посредником между аппаратурой, прикладным программным обеспечением и пользователем компьютера.

ОС различаются особенностями реализации алгоритмов управления ресурсами компьютера, областями использования. Программы, предназначенные для работы под управлением данной ОС, принято называть приложениями ОС.

Классификация ОС:

* В зависимости от количества одновременно выполняемых задач операционные системы делятся на однозадачные и многозадачные.
* В зависимости от размера ядра – на микроядерные и макроядерные.
* В зависимости от количества выполняемых функций – на специализированные и системы общего назначения.
* Операционные системы компьютеров, оснащенных несколькими процессорами, делятся на две категории: с асимметричной или симметричной обработками.

По числу одновременно выполняемых задач операционные системы делятся на два класса:

* однозадачные (MS DOS);
* многозадачные (OS/2, Unix, Windows).

Однозадачные ОС (например, MS DOS) передают все ресурсы вычислительной системы одному исполняемому приложению и не допускают запуска другого приложения.

Многозадачные ОС – управляют распределением ресурсов вычислительной системы между задачами и обеспечивают:

* возможность одновременной или поочередной работы нескольких приложений;
* возможность обмена данными между приложениями;
* возможность совместного использования программных, аппаратных, сетевых и прочих ресурсов вычислительной системы несколькими приложениями.

Смысл мультипрограммного режима работы заключается в том, что пока одна программа (один вычислительный процесс) ожидает завершения очередной операции ввода-вывода, другая программа может быть поставлена на решение. Это позволяет более полно использовать имеющиеся ресурсы и уменьшить общее время, необходимое для решения некоторого множества задач.

Однако, при этом время выполнения каждого процесса в общем случае больше, чем, если бы мы выполняли каждый из них как единственный. При мультипрограммировании повышается пропускная способность системы, но отдельный процесс никогда не может быть выполнен быстрее, чем, если бы он выполнялся в однопрограммном режиме. Всякое разделение ресурсов замедляет работу одного из участников за счет дополнительных затрат времени на ожидание освобождения ресурса.

**4 вопрос. Классификация ОС. Микроядерные и макроядерные ОС**

Операционная система – базовое системное программное обеспечение, управляющее работой компьютера и являющееся посредником между аппаратурой, прикладным программным обеспечением и пользователем компьютера.

Классификация ОС:

* В зависимости от количества одновременно выполняемых задач операционные системы делятся на однозадачные и многозадачные.

* В зависимости от размера ядра – на микроядерные и макроядерные.
* В зависимости от количества выполняемых функций – на специализированные и системы общего назначения.
* Операционные системы компьютеров, оснащенных несколькими процессорами, делятся на две категории: с асимметричной или симметричной обработками.

*По основному архитектурному принципу:*

* + микроядерные (ОСРВ QNX)
  + макроядерные (монолитные) — Linux, Windows 95/98.
* микроядерных системах ядро (главный модуль системы) имеет размер порядка десятков килобайтов. Остальные модули, образующие набор сервисных приложений, вызываются по мере необходимости. Этот подход вполне соответствует принципам структурного программирования.

Все первые ОС вынужденно являлись микроядерными, поскольку объем оперативной памяти компьютеров поначалу был небольшим. Например, первая версия операционной системы UNIX занимала всего около 12 Кбайт. Однако по мере увеличения объема оперативной памяти ядра операционных систем постепенно начали разрастаться.

* 90-е годы XX века было весьма распространенным убеждение, что большинство операционных систем следующих поколений будут строиться как микроядерные. Однако практика показывает, что это не совсем так. Разработчики желают иметь компактное микроядро, но при этом включить в него как можно больше функций, исполняемых непосредственно этим программным модулем. Это связано с тем, что выполнение затребованной функции другим модулем, вызываемым из микроядра, приводит и к дополнительным задержкам, и к дополнительным сложностям.
* макроядерных системах ядро получается монолитным, неделимым. Современные ОС общего назначения, такие как Windows, UNIX и Linux относятся к макроядерным системам. Ядра этих систем, представленные в виде программ, написанных на языке высокого уровня, содержат многие миллионы строк кода.

**5 вопрос. Классификация ОС. Специализированные операционные системы**

Операционная система – базовое системное программное обеспечение, управляющее работой компьютера и являющееся посредником между аппаратурой, прикладным программным обеспечением и пользователем компьютера.

Классификация ОС:

* В зависимости от количества одновременно выполняемых задач операционные системы делятся на однозадачные и многозадачные.
* В зависимости от размера ядра – на микроядерные и макроядерные.
* В зависимости от количества выполняемых функций – на специализированные и системы общего назначения.
* Операционные системы компьютеров, оснащенных несколькими процессорами, делятся на две категории: с асимметричной или симметричной обработками.

*Специализированные ОС.*

* этому классу относятся системы реального времени; системы мобильных вычислительных устройств; специализированные сетевые системы; а также системы, предназначенные для обучения студентов. В момент возникновения персональных компьютеров их операционные системы (ДОС) также относились к этому классу.

*ДОС (Дисковые Операционные Системы)*. Резидентный набор подпрограмм, который загружает пользовательскую программу в память и передает ей управление, после чего программа делает с системой все, что ей заблагорассудится. Считается желательным, чтобы после завершения программы машина оставалась в таком состоянии, чтобы ДОС могла продолжить работу. Существование систем такого класса обусловлено их простотой и тем, что они потребляют мало ресурсов. Еще одна причина, по которой такие системы могут использоваться даже на довольно мощных машинах – требование программной совместимости с ранними моделями того же семейства компьютеров.

*Системы реального времени* – системы, предназначенные для облегчения разработки так называемых приложений реального времени. Это программы, управляющие некомпьютерным по своей природе оборудованием, часто с очень жесткими ограничениями по времени реакции. Примером такого приложения может быть программа бортового компьютера крылатой ракеты, системы управления ускорителем элементарных частиц или промышленным оборудованием. Такие системы обязаны поддерживать многопроцессность, гарантированное время реакции на внешнее событие, простой доступ к таймеру и внешним устройствам. Примером такой системы может служить ОС QNX.

*Системы для обучения студентов*. Во времена молодости UNIX (версия 6) ее исходные коды были широко доступны по лицензии AT&T и активно изучались. Джон Лайонс (John Lions) из университета Нового Южного

Уэльса в Австралии даже написал небольшую брошюру, шаг за шагом описывающую работу UNIX. С разрешения AT&T эта брошюра использовалась во многих университетских курсах по операционным системам.

* выходом версии 7 система UNIX превратилась в дорогостоящий коммерческий продукт. Лицензия, под которой она распространялась, запрещала преподавание исходного кода на учебных курсах, чтобы не подвергать риску его статус коммерческого секрета. Поэтому многие университеты просто прекратили изучение UNIX, довольствуясь одной теорией. Чтобы исправить ситуацию Эндрив Таненбаум в 1987 году написал собственную операционную систему MINIX (mini-UNIX), предназначенную для обучения студентов, которая с точки зрения пользователя совместима с UNIX, но внутри совершенно самостоятельна. Ядро этой системы имело всего 4000 строк кода, в то время как в UNIX, или в Windows – это миллионы строк кода. Система настолько мала, что даже начинающий мог понять, как она работает. Одним из пользователей MINIX был финский студент по имени Линус Торвальдс. Он установил ее на свой компьютер и тщательно изучил исходный код. Опыты по усовершенствованию этой системы привели к созданию операционной системы LINUX в 1991 году.

*Сетевые системы*. Этот термин употребляют в двух различных смыслах:

* Системы, предназначенные только для предоставления сетевых услуг, аналогично тому, как ДОС предназначена для предоставления средств работы с диском. Под такое понимание подходят узкоспециализированные системы, такие как Novell Netware или, например, программное обеспечение маршрутизаторов Cisco.
* Системы, способные предоставлять сетевые услуги. Под такое определение подходят практически все современные ОС общего назначения.

**6 вопрос. Классификация ОС. Операционные системы общего назначения**

Операционная система – базовое системное программное обеспечение, управляющее работой компьютера и являющееся посредником между аппаратурой, прикладным программным обеспечением и пользователем компьютера.

Классификация ОС:

* В зависимости от количества одновременно выполняемых задач операционные системы делятся на однозадачные и многозадачные.
* В зависимости от размера ядра – на микроядерные и макроядерные.
* В зависимости от количества выполняемых функций – на специализированные и системы общего назначения.
* Операционные системы компьютеров, оснащенных несколькими процессорами, делятся на две категории: с асимметричной или симметричной обработками.

*Операционные системы общего назначения*

* этому классу относятся системы, берущие на себя выполнение большинства функций прочих видов ОС (за исключением, может быть, поддержки режима реального времени) и рассчитанные на интерактивную работу одного или нескольких пользователей в режиме разделения времени, при не очень жестких требованиях на время реакции системы на внешние события. Как правило, в таких системах уделяется большое внимание защите самой системы, программного обеспечения и пользовательских данных от ошибочных и злонамеренных действий программ и пользователей. Обычно такие системы используют встроенные в архитектуру процессора средства защиты и виртуализации памяти. К этому классу относятся такие широко распространенные системы, как различные семейства Unix, OS/2, семейство Windows NT, хотя некоторые из них не обеспечивают одновременной работы нескольких пользователей и защиты пользователей и программ друг от друга.

ОС общего назначения могут работать на компьютерах с одним или несколькими процессорами. ОС с мультипроцессорной обработкой делятся на две категории – с асимметричной либо симметричной обработкой.

**7 вопрос. Классификация ОС. Системы с асимметричной процессорной обработкой**

Операционная система – базовое системное программное обеспечение, управляющее работой компьютера и являющееся посредником между аппаратурой, прикладным программным обеспечением и пользователем компьютера.

Классификация ОС:

* В зависимости от количества одновременно выполняемых задач операционные системы делятся на однозадачные и многозадачные.
* В зависимости от размера ядра – на микроядерные и макроядерные.
* В зависимости от количества выполняемых функций – на специализированные и системы общего назначения.
* Операционные системы компьютеров, оснащенных несколькими процессорами, делятся на две категории: с асимметричной или симметричной обработками.

*Системы с асимметричной процессорной обработкой*

Операционные системы с асимметричной мультипроцессорной обработкой (asymmetric multiprocessing, ASMP) обычно выбирают для исполнения собственного кода один и тот же процессор, в то время как другие процессоры выполняют только пользовательские задачи. Так как код ОС исполняется на одном процессоре, то ОС ASMP довольно просто создать, усовершенствовав существующую однопроцессорную ОС. Особенно хорошо ОС ASMP подходят для работы на асимметричном оборудовании, например, процессоре, к которому подключен сопроцессор, или на двух процессорах, совместно использующих не всю доступную память. Однако такую ОС трудно сделать переносимой. Аппаратура разных производителей (и даже разные версии аппаратуры одного производителя) имеет тенденцию различаться по типу и степени асимметрии. Либо производители оборудования должны ориентироваться на одну ОС, либо ОС придется постоянно переписывать для каждой аппаратной платформы.

**8 вопрос. Классификация ОС. Системы с симметричной процессорной обработкой**

Операционная система – базовое системное программное обеспечение, управляющее работой компьютера и являющееся посредником между аппаратурой, прикладным программным обеспечением и пользователем компьютера.

Классификация ОС:

* В зависимости от количества одновременно выполняемых задач операционные системы делятся на однозадачные и многозадачные.
* В зависимости от размера ядра – на микроядерные и макроядерные.
* В зависимости от количества выполняемых функций – на специализированные и системы общего назначения.
* Операционные системы компьютеров, оснащенных несколькими процессорами, делятся на две категории: с асимметричной или симметричной обработками.

*Системы с симметричной процессорной обработкой*

Системы с симметричной мультипроцессорной обработкой (symmetric multiprocessing, SMP) позволяют коду ОС выполняться на любом свободном процессоре или на всех процессорах одновременно, причем каждому из процессоров доступна вся память. Такой подход полнее реализует возможности нескольких процессоров, так как дает возможность полнее использовать процессорное время, независимо от того, какие приложения исполняются. Исполнение ОС только на одном процессоре может сильно загружать его, в то время как остальные простаивают, что уменьшит производительность системы; при увеличении числа процессоров в системе возрастает вероятность того, что узким местом станут именно действия, выполняемые ОС. Помимо равномерного распределения системной загрузки, системы SMP сокращают время простоя из-за неисправностей, так как при сбое одного процессора код ОС может исполняться на других. Наконец, поскольку симметричная аппаратура реализуется разными производителями сходным образом, имеется возможность создания переносимой ОС SMP.

**9 вопрос. Процессы. Граф существования процесса**

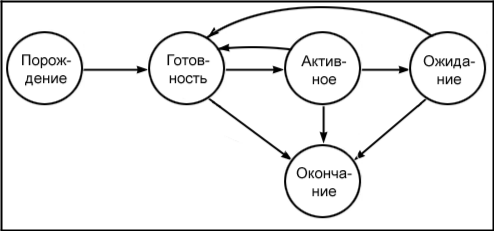
Процесс – система действия, реализующая определенную функцию в вычислительной системе. Это логическая единица работы ОС. ОС выполняет решение задач, связанных с процессами, таких как управление, планирование, взаимодействие, синхронизация, распределение ресурсов и др.

При исполнении программ на центральном процессоре чаще всего различают следующие характерные отдельные состояния:

* порождение — подготавливаются условия для первого исполнения на процессоре;
* активное состояние, или состояние “Счет” — программа исполняется на процессоре;
* ожидание — программа не исполняется на процессоре по причине занятости какого-либо требуемого ресурса;
* готовность — программа не исполняется, но для исполнения предоставлены все необходимые в текущий момент ресурсы, кроме центрального процессора;
* окончание — нормальное или аварийное окончание исполнения программы, после которого процессор и

другие ресурсы ей не предоставляются.

Процесс находится в каждом из своих допустимых состояний



* течение некоторого времени, после чего переходит в какое-то другое допустимое состояние. Состав допустимых состояний, а также допустимые переходы из состояния в состояние обычно задают в форме графа существования процесса.

Для ОС процесс в такой трактовке рассматривается как объект, в отношении которого требуется обеспечить реализацию каждого из допустимых состояний а также допустимые переходы из состояния в состояние в ответ на

события, которые могут явиться причиной таких переходов. Такие события могут инициироваться и самими

процессами, которые способны затребовать процессор или какой-либо другой ресурс, необходимый для исполнения программы.

**10 вопрос. Процессы. Классификация процессов**

Классификация процессов.

*По принадлежности к ЦП:*

* Внутренние - связаны с выполнение команд соответствующей программы на процессоре
* Внешние - процессы, которые развиваются на других устройствах вычислительной системы *По принадлежности к ОС:*
* Системные - исполняется программа из состава операционной системы
* Пользовательские - исполняется пользовательская (прикладная) программа

*По генеалогическому признаку:*

* Порождающие - задающий требование по порождению процессов
* Порожденные - создаваемый по требованию порождающего процесса *По результативности:*
* Эквивалентные - имеют одинаковый конечный результат обработки одних и тех же исходных данных по одной и той же или даже различным программам на одном и том же или на различных процессорах, трассы (порядок и длительность пребывания процесса в допустимых состояниях на интервале существования) эквивалентных процессов в общем случае не совпадают
* Тождественные - эквивалентные процессы, обработка данных которых происходит по одной и той же программе, но трассы при этом в общем случае не совпадают
* Равные - тождественные процессы, при совпадении трасс
* Различные - все остальные случаи

*По динамическому признаку:*

* Последовательные - интервалы двух процессов не пересекаются во времени
* Параллельные - на рассматриваемом интервале времени существуют одновременно два процесса
* Комбинированные - на рассматриваемом интервале найдется хотя бы одна точка, в которой существует один процесс, но не существует другой, и хотя бы одна точка, в которой оба процесса существуют одновременно

*По связности:*

* Конкурирующие - между ними поддерживаются с помощью системы управления процессами какого-либо рода связи: функциональные, пространственно-временные, управляющие, информационные
* Изолированные - не конкурирующие
* Информационно-независимые - два взаимосвязанных процесса при развитии используют совместно некоторые ресурсы, но информационно между собой не связаны, т. е. не обмениваются информацией
* Взаимодействующие - при наличии информационных связей между двумя процессами

*По временным характеристикам:*

* Процессы реального времени– это процессы, требующие своего полного выполнения к некоторому моменту времени
* Интерактивные процессы – это процессы, время выполнения которых должно укладываться в определенный интервал времени

**11 вопрос. Процессы. Программные потоки**

Процесс – система действия, реализующая определенную функцию в вычислительной системе. Это логическая единица работы ОС. ОС выполняет решение задач, связанных с процессами, таких как управление, планирование, взаимодействие, синхронизация, распределение ресурсов и др

Потоки - в зависимости от вида системы(link is external) имеют разную концепцию построения. В общем случае

* это процессы, имеющие по-умолчанию имеющие доступ к общим переменным, файловым дескрипторам и другим ресурсам памяти и передачи данных.

*Потоки*

Последовательный (однопоточный) процесс – это процесс, который имеет только один поток управления (control flow), характеризующийся изменением его счетчика команд. Поток (thread) – это запускаемый из некоторого процесса особого рода параллельный процесс, выполняемый в том же адресном пространстве, что и процесс-родитель. Схема организации однопоточного и многопоточного процессов изображена на рис. 10.1.

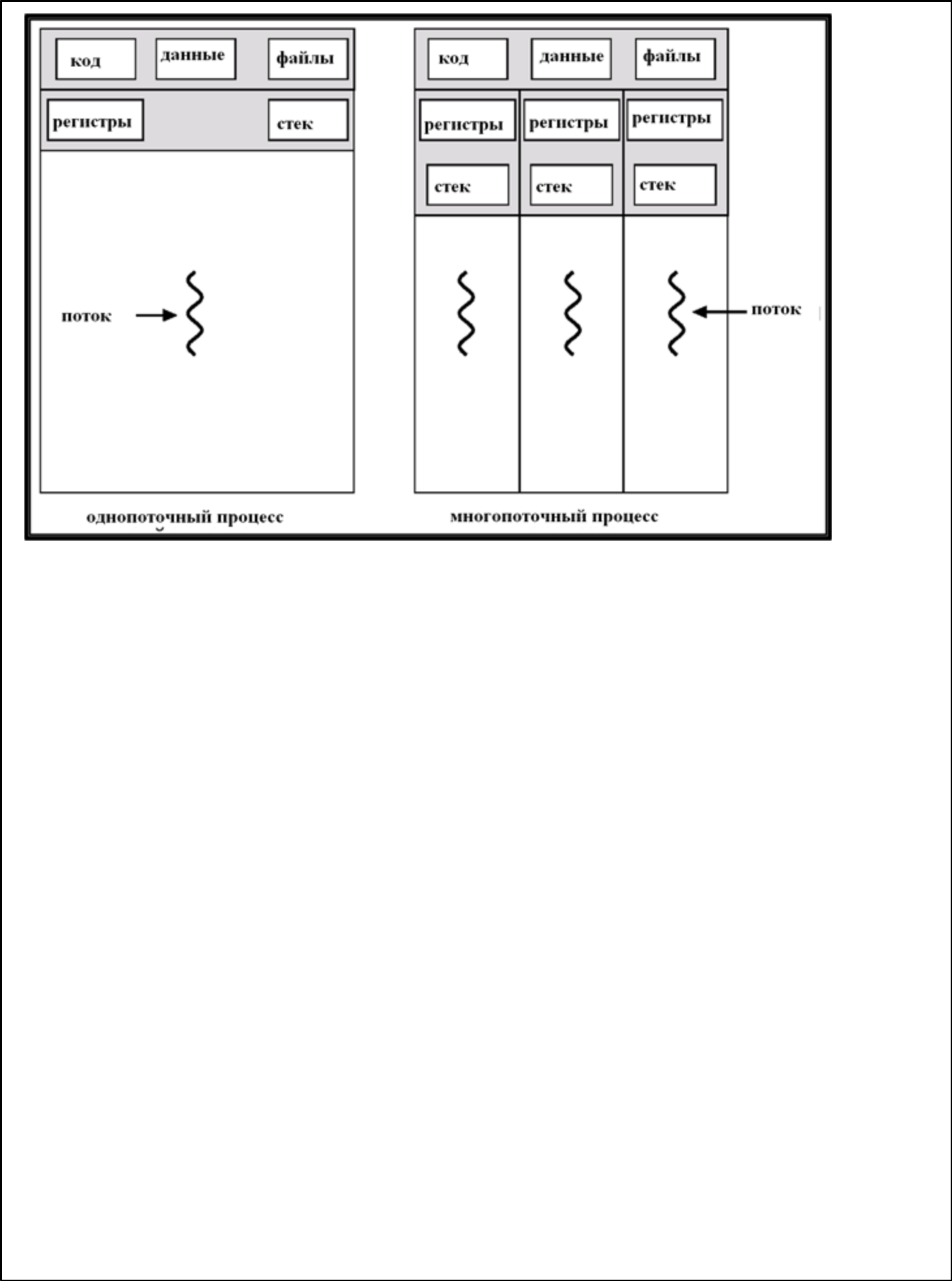


Рис. 10.1. Однопоточный и многопоточный процессы.

Как видно из схемы, однопоточный процесс использует, как обычно, код, данные в основной памяти и файлы, с которыми он работает. Процесс также использует определенные значения регистров и стек, на котором исполняются его процедуры. Многопоточный процесс организован несколько сложнее. Он имеет несколько параллельных потоков, для каждого из которых ОС создает свой стек и хранит свои собственные значения регистров. Потоки работают в общей основной памяти и используют то же адресное пространство, что и процесс-родитель, а также разделяют код процесса и файлы.

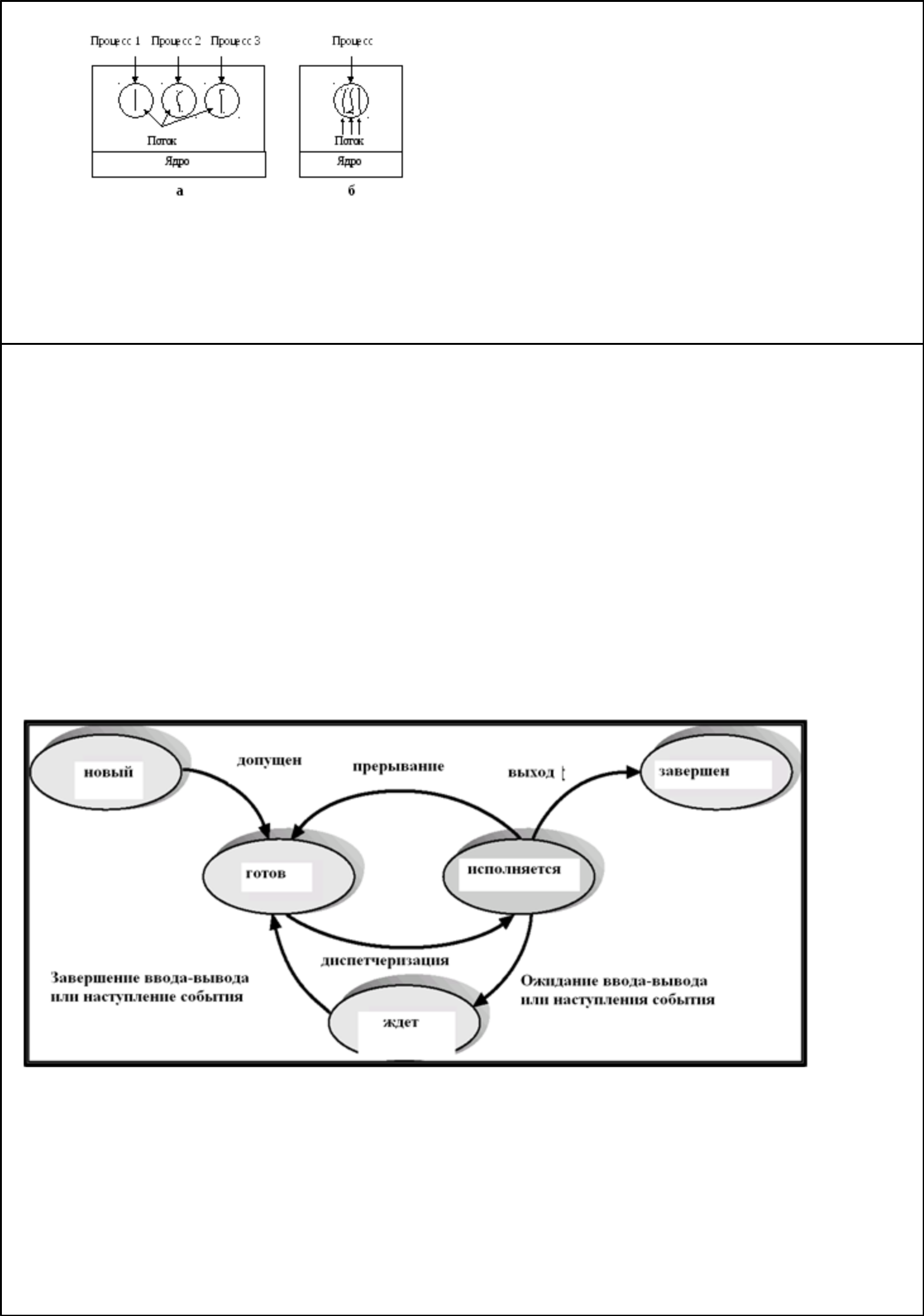
Многопоточность имеет большие преимущества:

* Увеличение скорости (по сравнению с использованием обычных процессов). Многопоточность основана на использовании облегченных процессов (lightweight processes),работающих в общем пространстве виртуальной памяти. Благодаря многопоточности, не возникает больше неэффективных ситуаций, типичных для классической системы UNIX, в которой каждая команда shell (даже команда вывода содержимого текущей директории ls исполнялась как отдельный процесс, причем в своем собственном адресном пространстве. В противоположность облегченным процессам, обычные процессы (имеющие собственное адресное пространство) часто называют тяжеловесными (heavyweight).
* Использование общих ресурсов. Потоки одного процесса используют общую память и файлы.
* Экономия. Благодаря многопоточности, достигается значительная экономия памяти, по причинам, объясненным выше. Также достигается и экономия времени, так как переключение контекста на облегченный процесс, для которого требуется только сменить стек и восстановить значения регистров, значительно быстрее, чем на обычный процесс

*Еще одна трактовка*

* традиционных операционных системах у каждого процесса есть адресное пространство и один поток управления. Тем не менее, нередко возникают ситуации, в которых желательно иметь несколько потоков управления, квазипараллельно выполняющихся в одном адресном пространстве так, как будто они представляют собой отдельные процессы (за исключением общности адресного пространства). Такие потоки управления называются программными потоками, или легковесными процессами.

На рис. 3.2.а представлены три обычных процесса. Каждый из них имеет собственное адресное пространство и единственный поток управления. На рис. 3.2.б представлен один процесс – с тремя потоками управления. В обоих случаях присутствуют три программных потока, но в первом случае каждый из них имеет собственное адресное пространство, а во втором – программные потоки имеют общую память.



* качестве примера приложения, рассчитанного на многопоточность, можно привести веб-браузер. Веб-страницы часто содержат множество картинок небольшого размера. Для загрузки каждой из них браузер должен установить отдельное соединение с сайтом. Установка и разрыв соединения отнимают много времени. При поддержке браузером многопоточности можно загружать несколько картинок одновременно, что значительно ускоряет загрузку страницы. Для небольших изображений установка соединений занимает гораздо больше времени, чем передача данных.

**12 вопрос. Управление ресурсами. Состояние процесса (НЕПОЛНЫЙ)**

ДОПОЛНИТЬ ПРО УПРАВЛЕНИЕ РЕСУРСАМИ

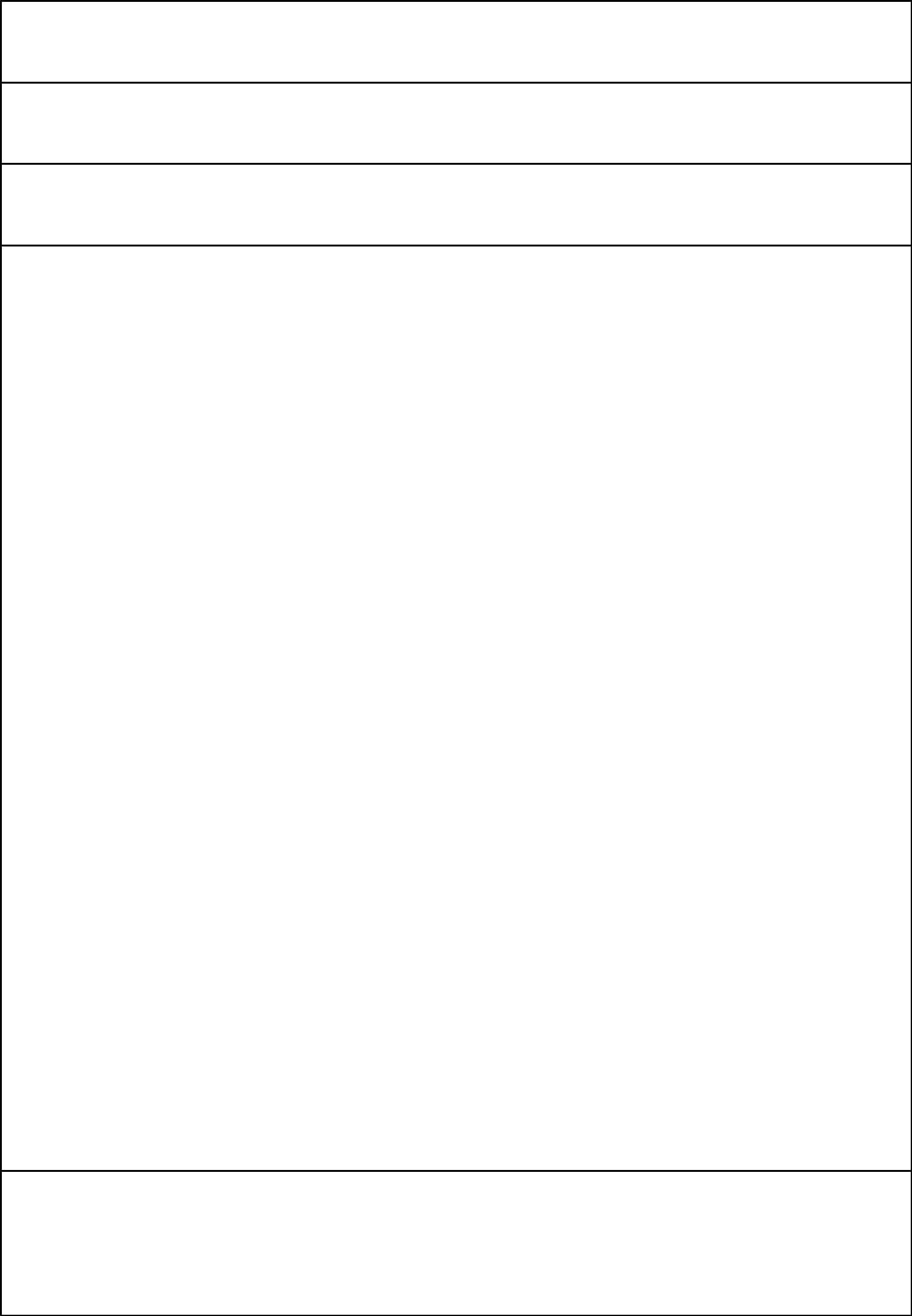
*Состояния процесса*

При исполнении процесс может изменять свое состояние следующим образом:

* Новый (new):Процесс создается операционной системой, но еще не начал выполняться.
* Исполняемый (running):Исполняются команды процесса на процессоре или процессорах компьютерной системы под управлением ОС.
* Ожидающий (waiting):Процесс ожидает наступления некоторого события, например, завершения ввода-вывода. В состоянии ожидания процесс не занимает процессор.
* Готовый к выполнению (ready):Процесс ожидает получения ресурсов процессора для его исполнения. В состояние готовности к выполнению процесс попадает обычно либо при его создании, либо после завершения ввода-вывода (из состояния ожидания).
* Завершенный (terminated):Исполнение процесса завершено.

Диаграмма состояний процесса представлена на рис. 8.1.

Как видно из схемы, новый процесс, созданный в системе, проходит стадию допущен (admitted) – включается операционной системой в очередь всех процессов в системе, после чего ОС переводит его в состояние готовности к выполнению. Отметим сразу, что очередь готовых к выполнению процессов – одна из наиболее часто используемых системных структур для управления процессами. Из состояния готовности в состояние выполнения процесс переводится планировщиком ОС в результате диспетчеризации – выделения кванта процессорного времени. При выполнении процесс может быть прерван (по таймеру, в результате ошибки и т.п.), а после обработки прерывания операционной системой переходит снова в состояние готовности к выполнению. Если в процессе выполняется синхронный ввод-вывод, либо процесс должен ожидать наступления некоторого события (например, определенного момента времени), процесс переходит в состояние ожидания. При завершении ввода-вывода или при наступлении ожидаемого события процесс не получает сразу же квант процессорного времени, а переходит в состояние готовности к выполнению. Процесс переходит в завершенное

состояние при завершении работы программы процесса - например, в результате системного вызова exit(c), где c

– код завершения. Если c = 0,процесс считается благополучно завершенным.

**13 вопрос. Сохранение состояния процесса (НЕ НАЙДЕН)**

Не нашел инфы

**14 вопрос. Потоки. Средства синхронизации потоков (ССЫЛКА)**

Если попадется, то по этой ссылке <https://intuit.ru/studies/courses/641/497/lecture/11288> Материал довольно сложно найти

**15 вопрос. Диспетчеризация процессора. Основная цель диспетчеризации процессора**

Диспетчеризация процессора – распределение его времени между процессами в системе. Цель диспетчеризации

– максимальная загрузка процессора, достигаемая с помощью мультипрограммирования.

Исполнение любого процесса можно рассматривать как цикл CPU / I-O – чередование периодов использования процессора и ожидания ввода-вывода.

Распределение периодов активности процессора ( bursts ) и ввода-вывода.

*Планировщик процессора*

Планировщик – компонента ОС, которая выбирает один из нескольких процессов, загруженных в память и готовых к выполнению, и выделяет процессор для одного из них.

Решения по диспетчеризации могут быть приняты в случаях, если процесс:

* Переключается из состояния выполнения в состояние ожидания.
* Переключается из состояния выполнения в состояние готовности к выполнению.
* Переключается из состояния ожидания в состояние готовности.
* Завершается.

Диспетчеризация типов 1 и 4 обозначается термином диспетчеризация без прерывания процесса (non-preemptive).

Диспетчеризация типов 2 и 3 обозначается термином диспетчеризация с прерыванием процесса (preemptive).

*Диспетчер процессора.*

Диспетчер процессора – компонента ОС, предоставляющая процессор тому процессу, который был выбран планировщиком. Диспетчер выполняет последовательность действий:

* Переключает контекст
* Переключает процессор в пользовательский режим
* Выполняет переход по соответствующему адресу в пользовательскую программу для ее рестарта. Скрытая активность (латентность) диспетчера (dispatch latency) – время, требуемое для диспетчера, чтобы

остановить один процесс и стартовать другой. Разумеется, система должна стремиться минимизировать это время, однако набор критериев диспетчеризации более сложен.

*Критерии диспетчеризации.* Имеется пять основных критериев диспетчеризации процессора, которые так или иначе должны учитываться системой.

* Использование процессора (CPU utilization) – поддержание его в режиме занятости максимально возможный период времени. Критерий оптимизации: максимизация данного показателя.
* Пропускная способность системы (throughput) – (среднее) число процессов, завершающих свое выполнение за единицу времени. Критерий оптимизации: максимизация.
* Время обработки процесса (turnaround time) – время, необходимое для исполнения какого-либо процесса. Критерий оптимизации: минимизация.
* Время ожидания (waiting time) – время, которое процесс ждет в очереди процессов, готовых к выполнению. Критерий оптимизации: минимизация.
* Время ответа (response time) – время, требуемое от момента первого запроса до первого ответа (данный показатель, как мы обсуждали ранее в лекции 1, наиболее важен для среды разделения времени). Критерий оптимизации: минимизация.

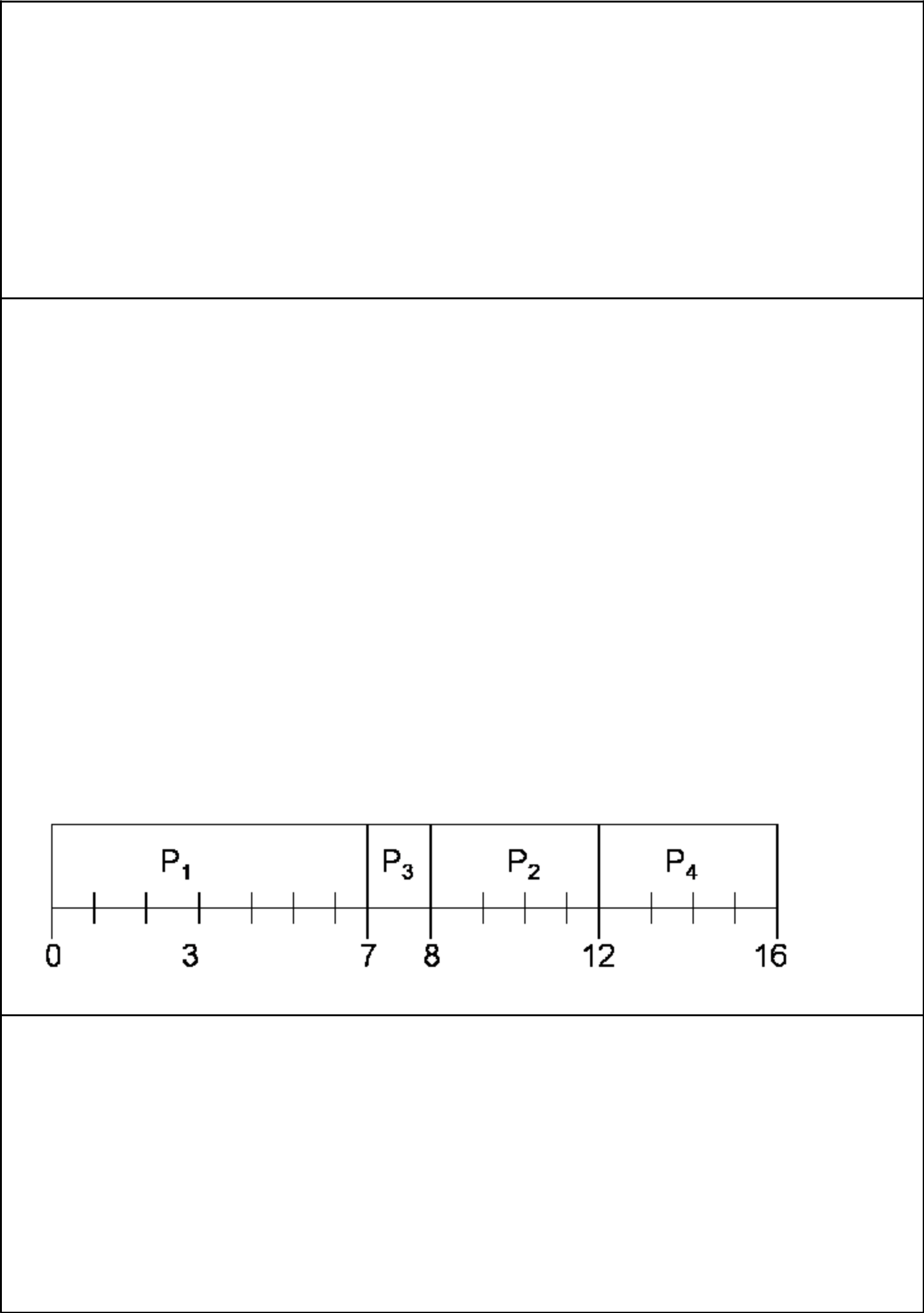
Как и при любой оптимизации, независимо от стратегии, удовлетворить всем критериям одновременно невозможно. Далее рассмотрим различные стратегии диспетчеризации и проанализируем их достоинства и недостатки, с точки зрения достижения оптимальности указанных критериев.

**16 вопрос. Планировщик. Разновидности стратегий, с точки зрения прерывания или избежания прерывания процессов, использует планировщик**

*Планировщик процессора*

Планировщик – компонента ОС, которая выбирает один из нескольких процессов, загруженных в память и готовых к выполнению, и выделяет процессор для одного из них.

Решения по диспетчеризации могут быть приняты в случаях, если процесс:

* Переключается из состояния выполнения в состояние ожидания.
* Переключается из состояния выполнения в состояние готовности к выполнению.
* Переключается из состояния ожидания в состояние готовности.
* Завершается.

Диспетчеризация типов 1 и 4 обозначается термином диспетчеризация без прерывания процесса (non-preemptive).

Диспетчеризация типов 2 и 3 обозначается термином диспетчеризация с прерыванием процесса (preemptive).

*Разновидности стратегий.*

Диспетчеризация без прерывания процессов (non-preemptive) – стратегии диспетчеризации, не использующие прерывания работы процессов при поступлении в систему более коротких или более приоритетных.

Диспетчеризация с прерыванием процессов (preemptive) – стратегии диспетчеризации, использующие прерывания работы процессов при поступлении в систему более коротких или более приоритетных.

**17 вопрос. Планировщик. Стратегия без прерывания процессов. Диаграмма Ганта**

*Планировщик процессора*

Планировщик – компонента ОС, которая выбирает один из нескольких процессов, загруженных в память и

готовых к выполнению, и выделяет процессор для одного из них.

Решения по диспетчеризации могут быть приняты в случаях, если процесс:

* Переключается из состояния выполнения в состояние ожидания.
* Переключается из состояния выполнения в состояние готовности к выполнению.
* Переключается из состояния ожидания в состояние готовности.
* Завершается.

Диспетчеризация типов 1 и 4 обозначается термином диспетчеризация без прерывания процесса (non-preemptive).

Диспетчеризация типов 2 и 3 обозначается термином диспетчеризация с прерыванием процесса (preemptive).

*Разновидности стратегий.*

Диспетчеризация без прерывания процессов (non-preemptive) – стратегии диспетчеризации, не использующие прерывания работы процессов при поступлении в систему более коротких или более приоритетных.

Диспетчеризация с прерыванием процессов (preemptive) – стратегии диспетчеризации, использующие прерывания работы процессов при поступлении в систему более коротких или более приоритетных.

*Диаграмма Ганта.*

При рассмотрении этой и других стратегий будем использовать диаграммы Ганта (Gantt charts) изображающие имена процессов и временные диапазоны их выполнения, выраженные в некоторых единицах времени.

Диаграмма Ганта (Gantt chart) – схема в виде "временной линейки", изображающая имена процессов и временные диапазоны их выполнения, выраженные в некоторых единицах времени.

*Схема диспетчеризации по стратегии SJF без прерывания процессов*

**18 вопрос. Планировщик. Стратегия с прерыванием процессов. Диаграмма Ганта**

*Планировщик процессора*

Планировщик – компонента ОС, которая выбирает один из нескольких процессов, загруженных в память и

готовых к выполнению, и выделяет процессор для одного из них.

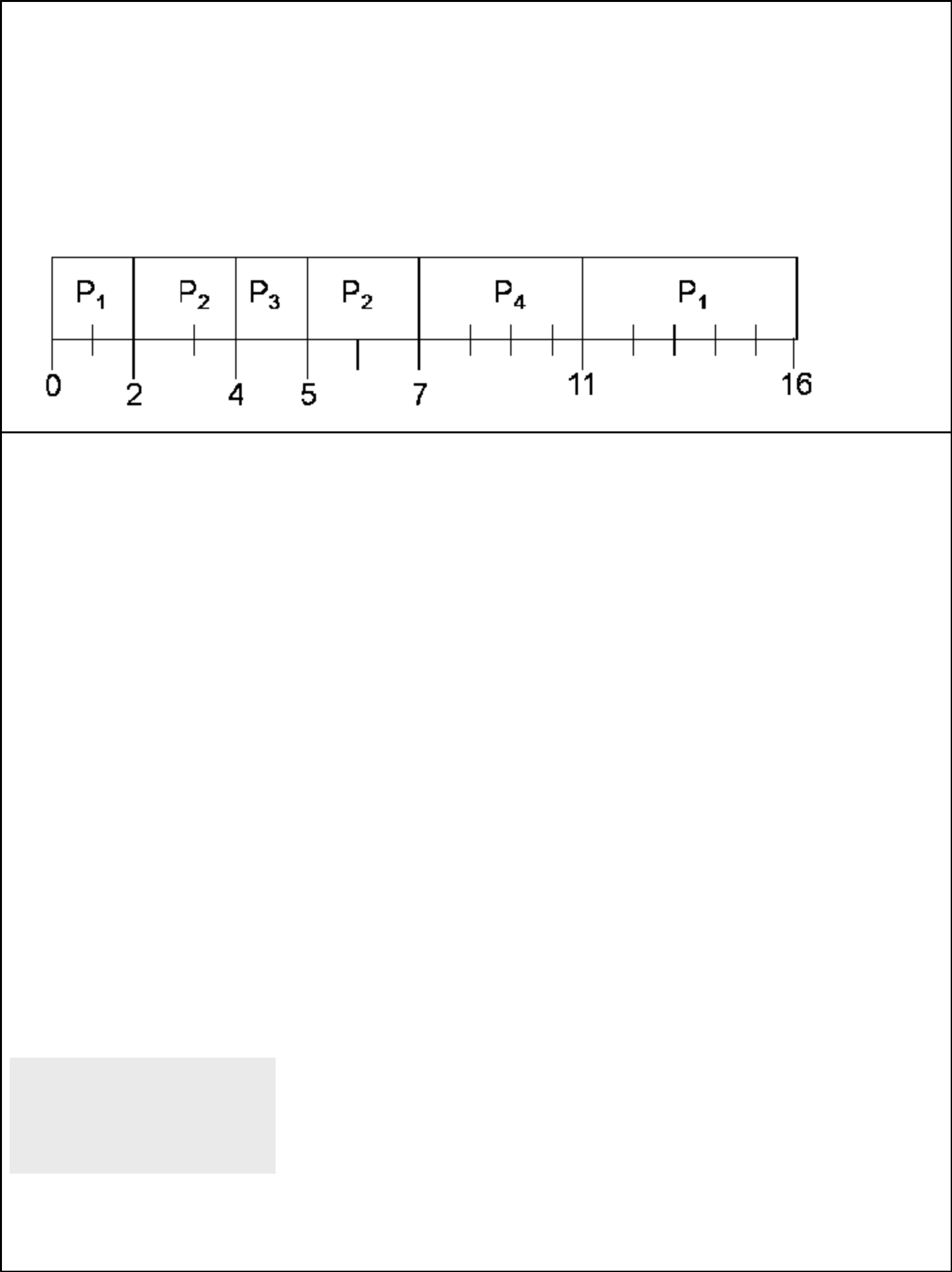
Решения по диспетчеризации могут быть приняты в случаях, если процесс:

* Переключается из состояния выполнения в состояние ожидания.
* Переключается из состояния выполнения в состояние готовности к выполнению.
* Переключается из состояния ожидания в состояние готовности.
* Завершается.

Диспетчеризация типов 1 и 4 обозначается термином диспетчеризация без прерывания процесса (non-preemptive).

Диспетчеризация типов 2 и 3 обозначается термином диспетчеризация с прерыванием процесса (preemptive).

*Разновидности стратегий.*

Диспетчеризация без прерывания процессов (non-preemptive) – стратегии диспетчеризации, не использующие прерывания работы процессов при поступлении в систему более коротких или более приоритетных.

Диспетчеризация с прерыванием процессов (preemptive) – стратегии диспетчеризации, использующие прерывания работы процессов при поступлении в систему более коротких или более приоритетных.

*Диаграмма Ганта.*

При рассмотрении этой и других стратегий будем использовать диаграммы Ганта (Gantt charts) изображающие имена процессов и временные диапазоны их выполнения, выраженные в некоторых единицах времени.

Диаграмма Ганта (Gantt chart) – схема в виде "временной линейки", изображающая имена процессов и временные диапазоны их выполнения, выраженные в некоторых единицах времени.

*Схема диспетчеризации по стратегии SJF с прерыванием процессов*

**19 вопрос. Планировщик. стратегии FCFS и каковы ее недостатки** *Планировщик процессора*

Планировщик – компонента ОС, которая выбирает один из нескольких процессов, загруженных в память и готовых к выполнению, и выделяет процессор для одного из них.

Решения по диспетчеризации могут быть приняты в случаях, если процесс:

* Переключается из состояния выполнения в состояние ожидания.
* Переключается из состояния выполнения в состояние готовности к выполнению.
* Переключается из состояния ожидания в состояние готовности.
* Завершается.

Диспетчеризация типов 1 и 4 обозначается термином диспетчеризация без прерывания процесса (non-preemptive).

Диспетчеризация типов 2 и 3 обозначается термином диспетчеризация с прерыванием процесса (preemptive).

*Стратегии диспетчеризации.*

* Стратегия First-Come-First-Served (FCFS)
* Стратегия Shortest Job First (SJF)
* Диспетчеризация по приоритетам
* Стратегия Round Robin (RR)

*Стратегия First-Come-First-Served (FCFS)*

Стратегия First-Come-First-Served (обслуживание в порядке поступления) – наиболее простая стратегия диспетчеризации, при которой ресурсы процессора предоставляются процессам в порядке их поступления (ввода) в систему, независимо от потребляемых ими ресурсов, в частности, от заявленного процессом времени, требуемого для его выполнения.

Рассмотрим следующий пример. Пусть процессы P1, P2 и P3 введены в систему в указанном порядке со следующими периодами активности:

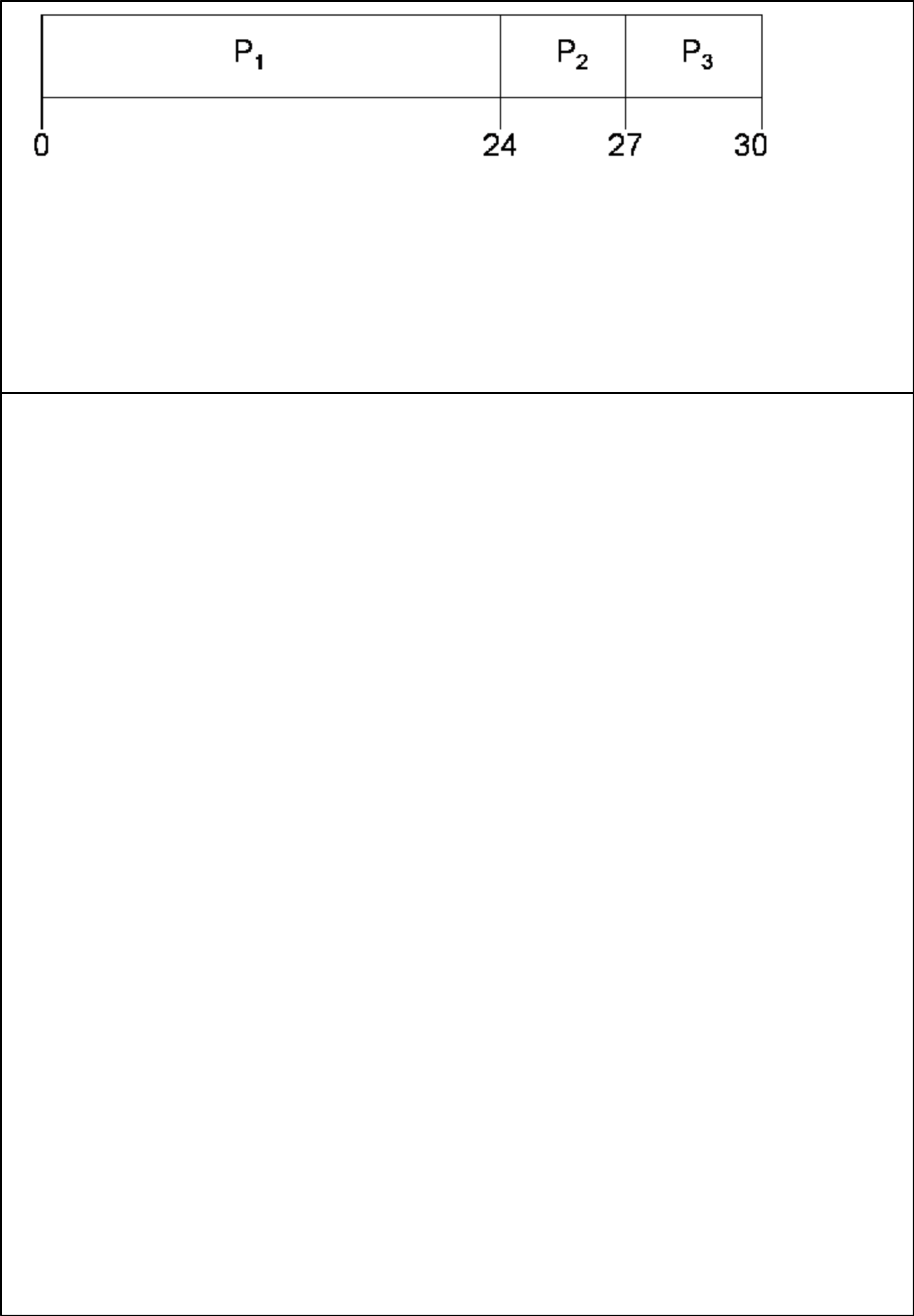
Процесс Период активности

P1 24

P2 3

P3 3

Тогда при использовании стратегии FCFS для их диспетчеризации первым получит процессор первый процесс, несмотря на то, что он – наиболее долгий. Распределение процессора между процессами в данном случае изображено ниже



Таким образом, время ожидания для P1 = 0; P2= 24; P3 = 27.

Среднее время ожидания: (0 + 24 + 27)/3 = 17.

Если порядок процессов иной: P2 , P3 , P1 (последний введенный в систему процесс – самый долгий), то результат их диспетчеризации будет совершенно иным

Время ожидания процессов в данном случае: P1 = 6; P2 = 0; P3 = 3.

Среднее время ожидания: (6 + 0 + 3)/3 = 3

Данный результат много лучше, чем в предыдущем случае.

Эффект, продемонстрированный первым примером, носит название эффекта сопровождения (convoy effect) – увеличение среднего времени ожидания процессов в случаях, если короткий процесс обслуживается после долгого процесса.

**20 вопрос. Планировщик. стратегии SJF (и SRTF) и оптимальность по какому критерию она обеспечивает**

*Планировщик процессора*

Планировщик – компонента ОС, которая выбирает один из нескольких процессов, загруженных в память и

готовых к выполнению, и выделяет процессор для одного из них.

Решения по диспетчеризации могут быть приняты в случаях, если процесс:

* Переключается из состояния выполнения в состояние ожидания.
* Переключается из состояния выполнения в состояние готовности к выполнению.
* Переключается из состояния ожидания в состояние готовности.
* Завершается.

Диспетчеризация типов 1 и 4 обозначается термином диспетчеризация без прерывания процесса (non-preemptive).

Диспетчеризация типов 2 и 3 обозначается термином диспетчеризация с прерыванием процесса (preemptive).

*Стратегии диспетчеризации.*

* Стратегия First-Come-First-Served (FCFS)
* Стратегия Shortest Job First (SJF)
* Диспетчеризация по приоритетам
* Стратегия Round Robin (RR)

Стратегия Shortest Job First (SJF)

Стратегия Shortest Job First (SJF, обслуживание самого короткого задания первым) – стратегия диспетчеризации процессора, при которой процессор предоставляется в первую очередь наиболее короткому процессу из имеющихся в системе.

* данном случае с каждым процессом связывается длина его очередного периода активности. Эта длина используется для того, чтобы первым обслужить самый короткий процесс .

Возможны две схемы применения данной стратегии:

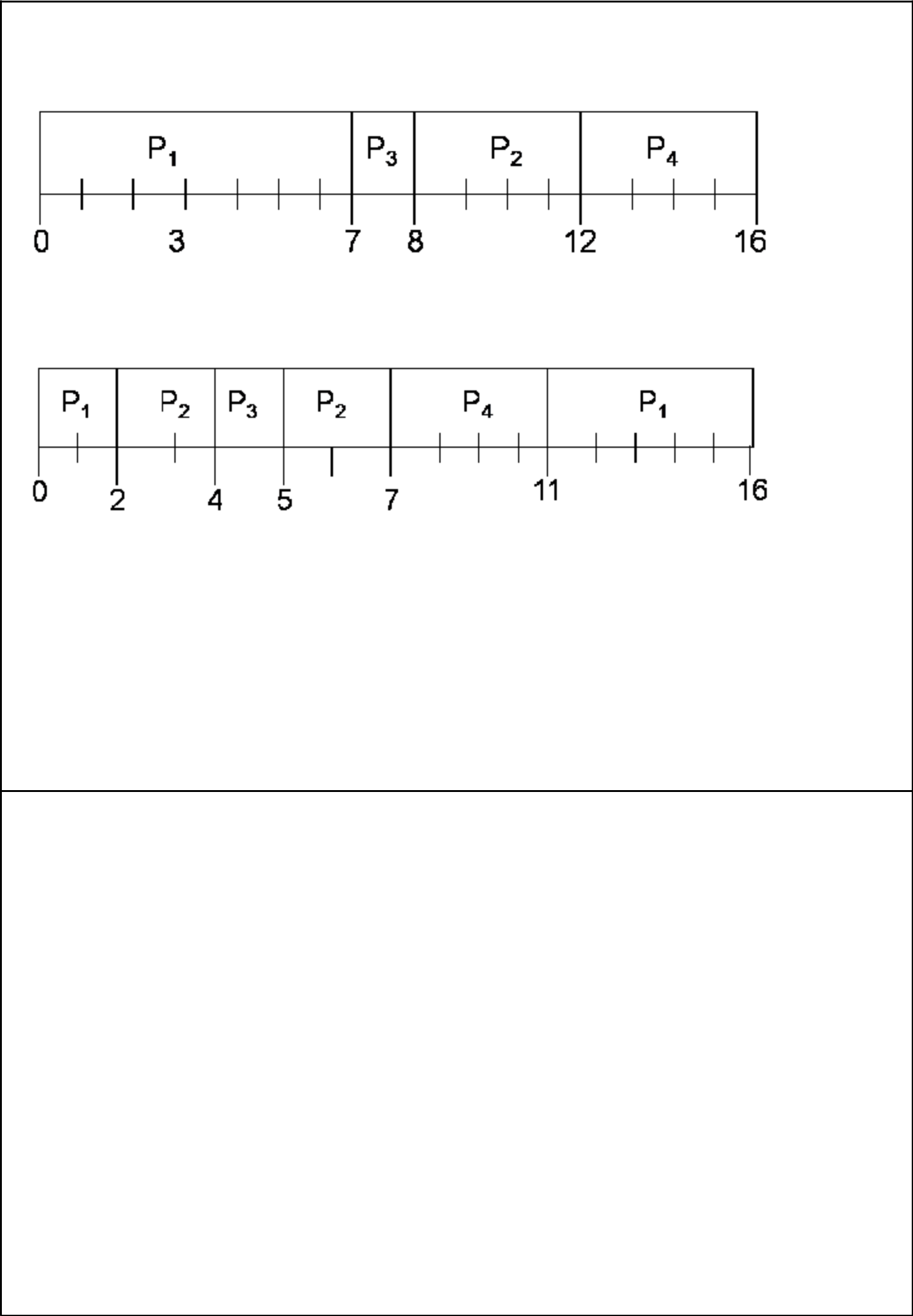
* **Без прерывания процессов** – пока процессу предоставляется процессор, он не может быть прерван, пока не истечет его квант времени.
* **С прерыванием процессов** – если приходит новый процесс, время активности которого меньше, чем оставшееся время активного процесса, - прервать активный процесс. Эта схема известна под названием

**Shortest-Remaining-Time-First (SRTF)**.

Нетрудно видеть, что стратегия SJF оптимальна, в том смысле, что она обеспечивает минимальное среднее время ожидания для заданного набора процессов.

Рассмотрим пример применения стратегии SJF без прерывания процессов. Пусть набор процессов, времен их появления в системе и времен их активности следующие:

|  |  |  |
| --- | --- | --- |
| Процесс | Время появления | Время активности |
| P1 | 0.0 | 7 |
| P2 | 2.0 | 4 |



|  |  |  |
| --- | --- | --- |
| P3 | 4.0 | 1 |
| P4 | 5.0 | 4 |

Схема их диспетчеризации по стратегии SJF без прерывания процессов приведена ниже

В данном случае среднее время ожидания = (0 + 6 + 3 + 7)/4 = 4.

Теперь применим к тем же процессам стратегию SJF с прерыванием и проанализируем, как изменится среднее время ожидания. Результат применения стратегии изображен ниже

* данном случае принцип прерывания процесса в момент поступления в систему более короткого процесса применяется несколько раз:
  + в момент 2 прерывается процесс 1 и начинает исполняться более короткий процесс 2;
  + в момент 4 прерывается процесс 2 и начинает исполняться более короткий процесс 3.

Из диаграммы видно, что, вследствие применения принципа прерывания процессов, периоды непрерывного выполнения процесса на процессоре могут быть не смежными и перемежаться с периодами выполнения других процессов.

* данном случае среднее время ожидания = (9 + 1 + 0 +2)/4 = 3, т.е. оно, как и следовало предполагать, оказалось меньше, чем без применения принципа прерывания процессов.

**21 вопрос. Планировщик. Диспетчер. Диспетчеризации по приоритетам** *Планировщик процессора*

Планировщик – компонента ОС, которая выбирает один из нескольких процессов, загруженных в память и готовых к выполнению, и выделяет процессор для одного из них.

Решения по диспетчеризации могут быть приняты в случаях, если процесс:

* Переключается из состояния выполнения в состояние ожидания.
* Переключается из состояния выполнения в состояние готовности к выполнению.
* Переключается из состояния ожидания в состояние готовности.
* Завершается.

Диспетчеризация типов 1 и 4 обозначается термином диспетчеризация без прерывания процесса (non-preemptive).

Диспетчеризация типов 2 и 3 обозначается термином диспетчеризация с прерыванием процесса (preemptive).

*Стратегии диспетчеризации.*

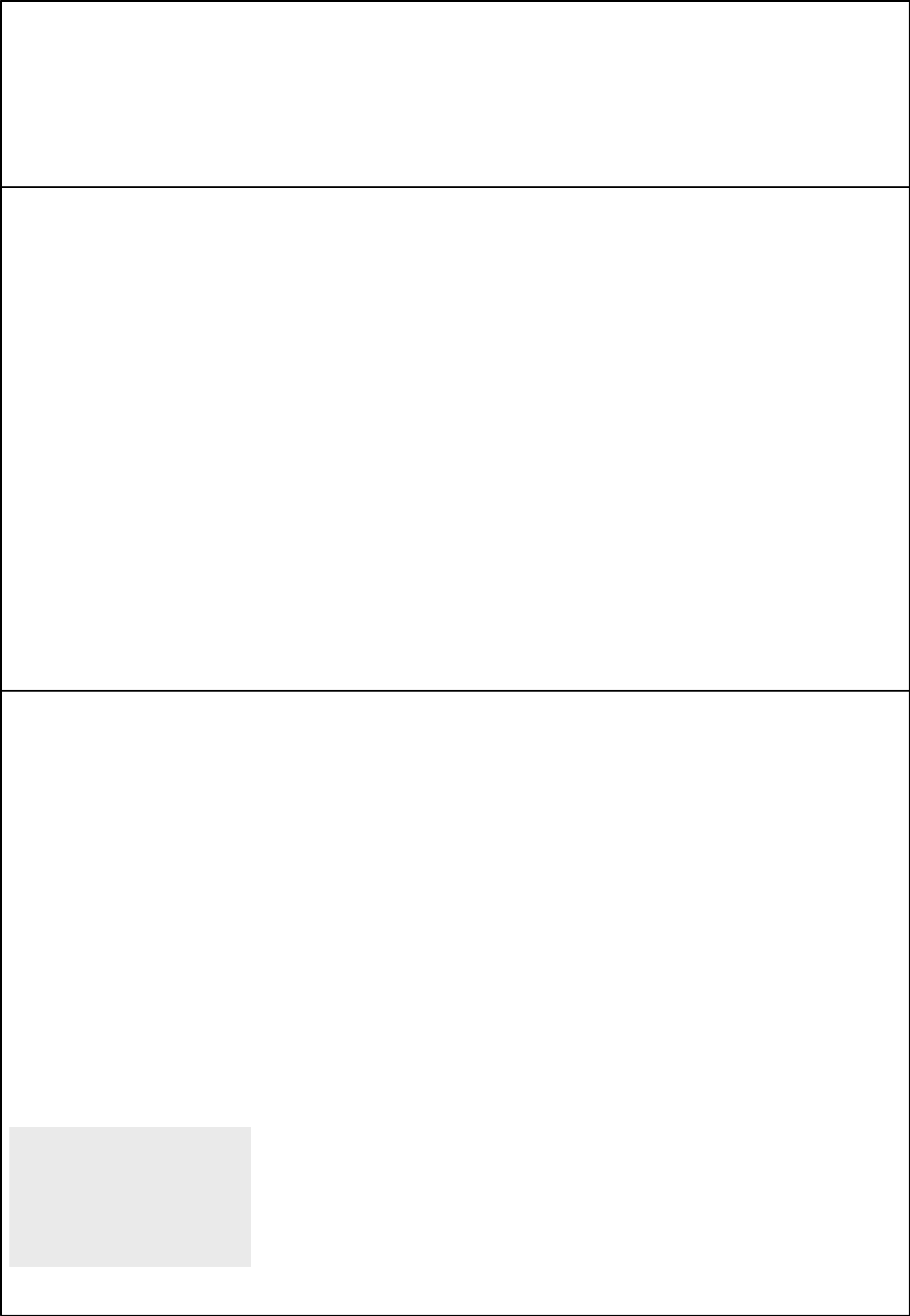
* Стратегия First-Come-First-Served (FCFS)
* Стратегия Shortest Job First (SJF)
* Диспетчеризация по приоритетам
* Стратегия Round Robin (RR)

*Диспетчеризация по приоритетам*

При данной стратегии с каждым процессом связывается его приоритет (целое число). Процессор выделяется

процессу с наивысшим приоритетом (будем считать, что меньшее число означает более высокий приоритет процесса, т.е. наиболее высокий приоритет процесса равен 1).

Данная стратегия имеет варианты с прерыванием и без прерывания.

Более того, стратегию SJF можно рассматривать как диспетчеризацию по приоритетам, в которой приоритетом является очередное время активности.

(следующий вопрос, но можно рассказать)

При диспетчеризации по приоритетам возникает проблема "голодания" (starvation) - ситуации, когда процессы с низким приоритетом могут никогда не исполниться и бесконечно ждать.

Традиционным способом решение данной проблемы в операционных системах является учет возраста процесса

* aging ): c течением времени приоритет процесса повышается системой.

**22 вопрос. Планировщик. Проблема голодания процессов и каково ее решение в ОС**

*Планировщик процессора*

Планировщик – компонента ОС, которая выбирает один из нескольких процессов, загруженных в память и

готовых к выполнению, и выделяет процессор для одного из них.

Решения по диспетчеризации могут быть приняты в случаях, если процесс:

* Переключается из состояния выполнения в состояние ожидания.
* Переключается из состояния выполнения в состояние готовности к выполнению.
* Переключается из состояния ожидания в состояние готовности.
* Завершается.

Диспетчеризация типов 1 и 4 обозначается термином диспетчеризация без прерывания процесса (non-preemptive).

Диспетчеризация типов 2 и 3 обозначается термином диспетчеризация с прерыванием процесса (preemptive).

*Стратегии диспетчеризации.*

* Стратегия First-Come-First-Served (FCFS)
* Стратегия Shortest Job First (SJF)
* Диспетчеризация по приоритетам
* Стратегия Round Robin (RR)

*Проблема голодания.*

При диспетчеризации по приоритетам возникает проблема "голодания" (starvation) - ситуации, когда процессы с низким приоритетом могут никогда не исполниться и бесконечно ждать.

Традиционным способом решение данной проблемы в операционных системах является учет возраста процесса

* aging ): c течением времени приоритет процесса повышается системой.

**23 вопрос. Стратегии RR, оптимальность по какому критерию она обеспечивает и по какому критерию она хуже, чем SJF**

*Стратегия Round Robin (RR, круговая система)* – это предоставление всем процессам по очереди одинаковых квантов времени.

При данной стратегии каждый процесс получает небольшой квант процессорного времени, обычно – 10-100 миллисекунд. После того, как это время закончено, процесс прерывается и помещается в конец очереди готовых процессов.

Если всего имеется n процессов в очереди готовых к выполнению, и квант времени равен q, то каждый процесс получает 1/ n процессорного времени порциями самое большее по q единиц за один раз. Ни один процесс не ждет больше, чем (n-1) q единиц времени.

Производительность данной стратегии зависит от коэффициента q:

* если q велико, то стратегия фактически эквивалентна стратегии FCFS;
* если q мало, то q должно быть большим, чем время контекстного переключения, иначе слишком велики окажутся накладные расходы на переключения с одного процесса на другой.

Рассмотрим пример применения стратегии RR. Пусть в системе имеются следующие процессы со следующими временами активности:

Процесc Время активности

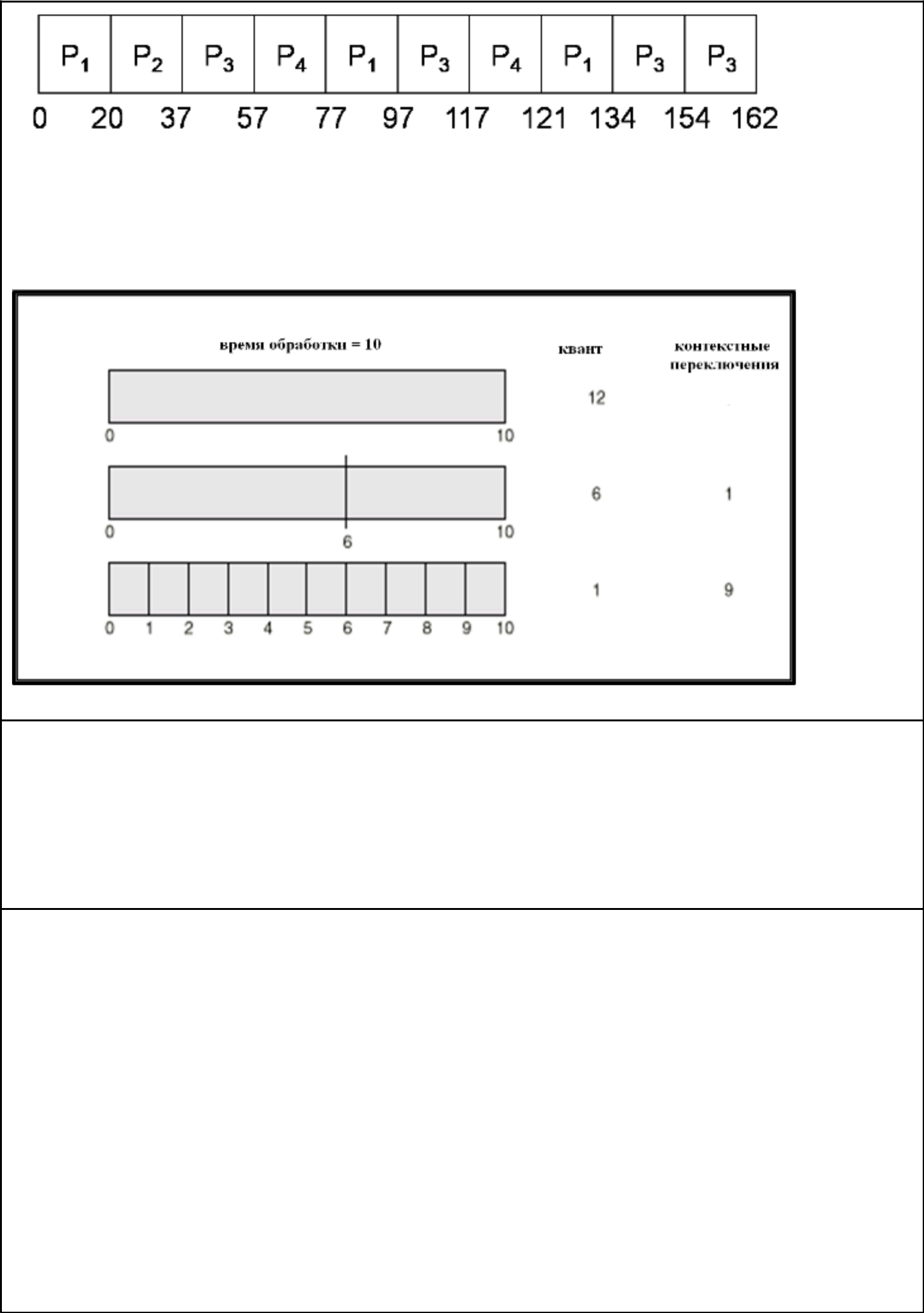
P1 53

P2 17

P3 68

P4 24

Схема диспетчеризации процессора по стратегии RR с квантом времени q = 20 приведена ниже



Обычно стратегия RR имеет худшее время оборота, чем SJF (так как каждый процесс должен ждать до предоставления следующего кванта времени, пока кванты времени будут предоставлены всем другим процессам), но лучшее время ответа.

Ниже иллюстрируется зависимость числа контекстных переключений от кванта времени: чем меньше квант, тем больше число переключений контекста.

**24 вопрос. В чем заключается задача управления памятью? входная очередь заданий.**

Управление памятью, наряду с управлением процессами и ресурсами, - одна из наиболее важных функций операционной системы. *Задача ОС* заключается в том, чтобы размещать в памяти пользовательские процессы, их данные, обслуживать запросы процессов на области памяти заданных размеров.

Любая программа, введенная в систему, должна быть размещена в памяти и оформлена в виде процесса для ее выполнения. Каждая программа при вводе в систему помещается во *входную очередь* – совокупность процессов на диске, ожидающих размещения в памяти для выполнения своих программ. До своего выполнения пользовательские программы проходят в системе несколько стадий.

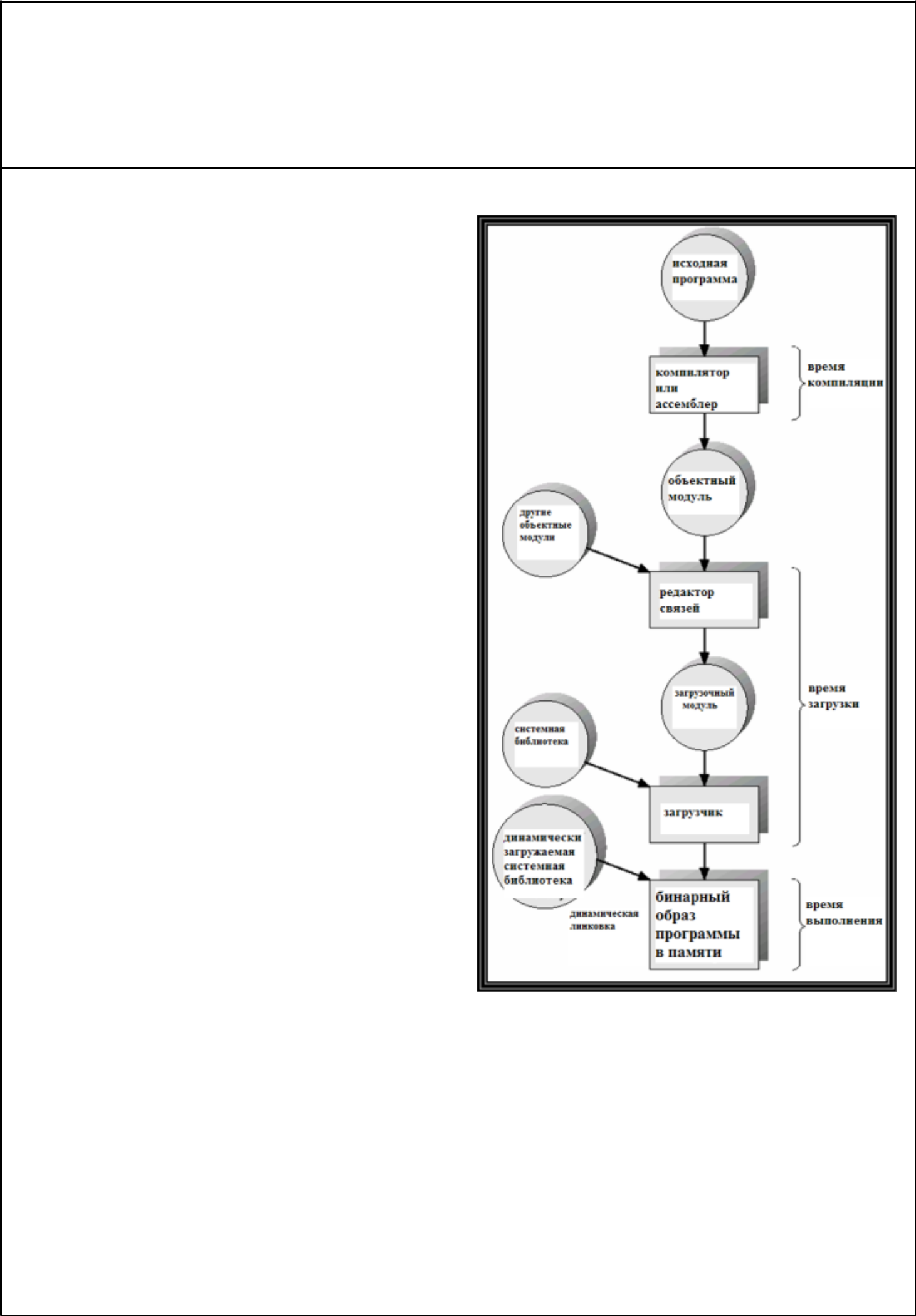
**25 вопрос. Связывание адресов и на каких этапах обработки программы оно может выполняться**

*Связывание программ и данных с адресами в памяти*

Перед загрузкой данных или кода в память они должны быть в какой-либо момент связаны с определенными

адресами в памяти. Связывание может выполняться на разных этапах:

* Связывание во время компиляции (compile-time).Если адрес в памяти априорно известен, компилятором может быть сгенерирован код с абсолютными адресами. При любом изменении размещения программы в памяти должна быть выполнена перекомпиляция. Данный подход более характерен для ранних компьютерных систем с небольшим объемом памяти, либо для обработки и выполнения системных модулей – частей ядра ОС, для которых характерно использование резидентных абсолютных адресов. Для пользовательских программ такой подход неудобен, так как не обеспечивает достаточной гибкости, в частности, возможности без изменений перезагрузить код в другую область памяти.
* Связывание во время загрузки (load-time).Загрузка программы в память – стадия ее обработки системой, предшествующая выполнению программы. Чтобы начальный адрес области памяти, куда загружается программа, можно было менять, и это не привело бы к необходимости изменения кода программы, применяется следующий метод. Генерируется перемещаемый код (relocatable code) – код, в котором адресация происходит относительно значения регистра перемещения (relocation register),и адрес в памяти равен сумме значения регистра перемещения и адреса, вычисляемого в команде. Таким образом, при необходимости загрузки кода на другое место в памяти требуется изменить только значение регистра

перемещения. Подобный подход широко используется для программ, написанных на традиционных языках программирования.

* Связывание во время исполнения (runtime),или динамическое (позднее) связывание.Используется, если процесс во время выполнения может быть перемещен из одного сегмента памяти в другой. Для реализации связывания во время исполнения требуется аппаратная поддержка отображения адресов – например, регистры базы и границы. В большинстве систем для пользовательских программ используется, главным образом, именно связывание во время исполнения.

**26 вопрос. Этапы обработки проходит программа на пути от исходного кода к двоичному образу в памяти** Общая схема многоэтапной обработки пользовательской программы, используемую в любой ОС

**Исходный код** программы (в форме текстового файла) на языке высокого уровня или на ассемблере преобразуется компилятором или ассемблером в

**объектный модуль**, содержащий бинарные выполняемые машинные команды и **таблицу символов**, определенных и использованных в данном модуле кода. Рассмотренная фаза называется **временем компиляции**.

Однако объектный модуль не может непосредственно исполняться, так как он содержит неразрешенные ссылки на внешние модули и их компоненты. Следующая фаза обработки программы – **редактирование связей. Редактор связей (linker)** – системная программа, которая получает на вход один или несколько объектных модулей, а на выходе выдает **загрузочный модуль** – двоичный код, образованный кодом нескольких объектных модулей, в котором разрешены все межмодульные ссылки - для каждого символа, внешнего для данного объектного модуля **A**, найден соответствующий символ (процедуры, переменной и т.д.) из другого модуля **B**, на который ссылается модуль **A**, и код соответственно откорректирован, т.е. он правильно адресует внешний символ.

Загрузочный модуль может быть загружен в память для исполнения с помощью еще одной системной программы

– **загрузчика (loader)**,который получает на вход загрузочный модуль и файлы с бинарными кодами **системных библиотек**,которые использует программа.Загрузчик, объединяя код программы с кодами системных библиотек, создает **бинарный образ программы в памяти**.

Фаза вызова редактора связей и загрузчика носит общее название **время загрузки**. Во многих ОС функции редактора связей и загрузчика, с целью экономии времени обработки программы в системе, объединены в одной системной программе – **редакторе связей и**

**загрузчике (linker and loader)**.Например, в системе UNIX редактор связей и загрузчик называется **ld (Linker and loaDer)**.Объединенному загрузчику и редактору связей на вход передается список объектных модулей и список библиотек, и в результате он генерирует исполняемый код. Фаза редактирования связей и загрузки часто на программистском слэнге называется **линковкой (linking)**.Будем далее использовать именно этот короткий и выразительный термин.

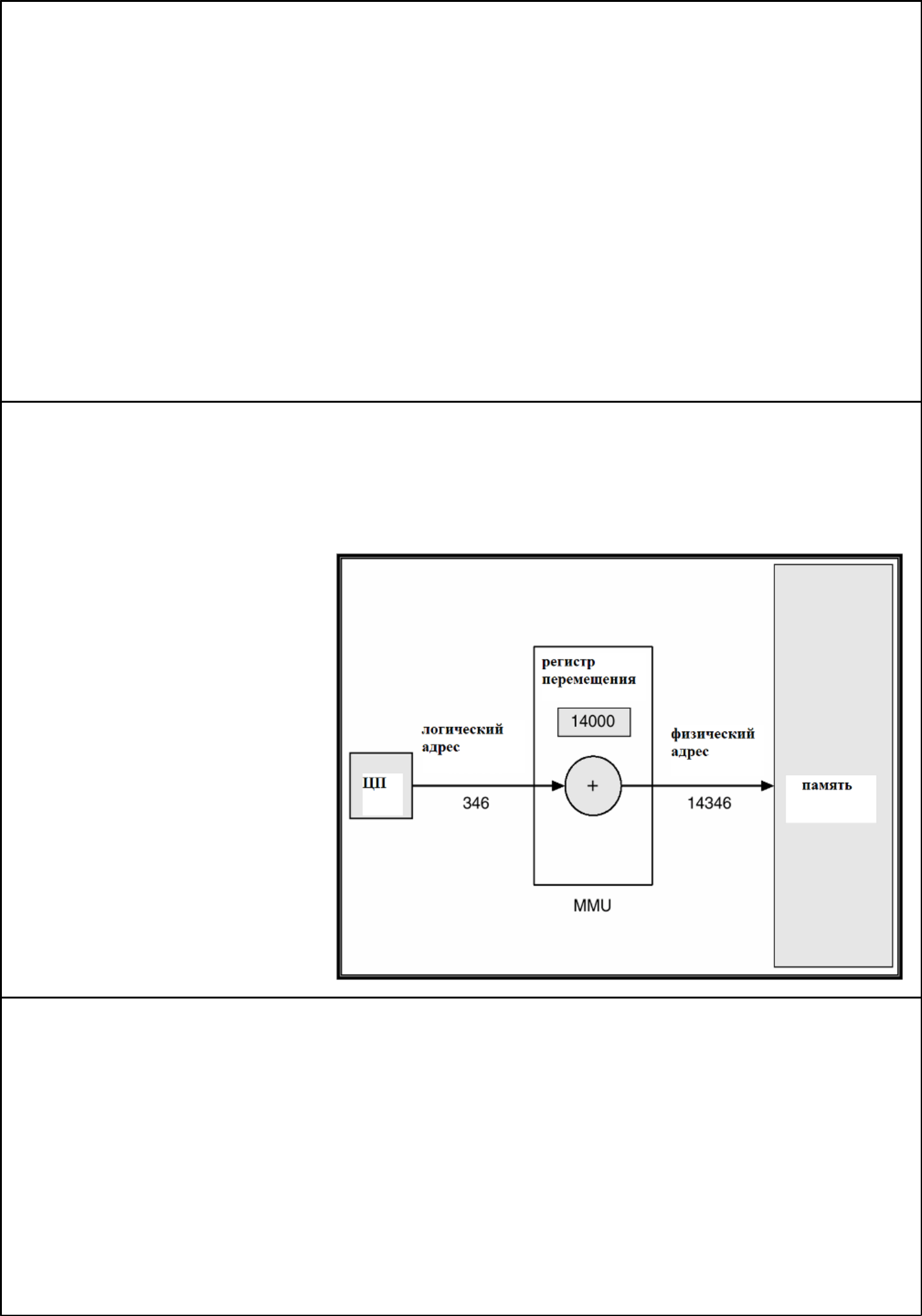
Вот пример последовательности фаз обработки программы в терминах команд системы UNIX:

сс –c program.c // Компиляция исходного кода на Си.

* В рабочей директории – объектный модуль program.o

ld program.o mylibrary.a // редактирование связей и загрузка

* В рабочей директории – исполняемый код с именем по умолчанию a.out a.out // Исполнение программы
* В стандартный вывод (по умолчанию – на консоль)
  + выдаются результаты программы

* примере предполагается, что в файле **program.c** хранится исходный код программы на Си, которая использует библиотечные функции из библиотеки **mylibrary.a**. Отметим соглашения в системе UNIX о расширениях имен файлов: **.c –** исходный код на Си, **.o** – объектный модуль, **.a** – бинарный файл **статически линкуемой библиотеки** (аббревиатура от термина **archive** ). Исполняемый код (executable) в UNIX не имеет стандартногорасширения имени, но имеет полное имя по умолчанию – несколько архаичное имя **a.out** (аббревиатура от **asembler output** ).
* Windows соглашения о расширениях имен файлов несколько иные: **.obj** – объектный модуль, **.exe** – исполняемый код, **.lib** – статически линкуемая библиотека.

Отличить объектный модуль от загрузочного очень просто: они сильно отличаются по своему размеру. Объясняется это тем, что в загрузочном модуле присутствует полностью или частично код статически линкуемых библиотек, а также гораздо больше, чем у объектного модуля, таблица символов – она содержит все символы библиотек и других объектных модулей, слинкованных в единую исполняемую программу.

На этапе выполнения, при первом обращении к ним из программы, в память загружаются **динамически линкуемые библиотеки (dymanically linked libraries)**.Данная разновидность библиотек, реализованная во всехсовременных ОС, позволяет сэкономить память, занимаемую образом исполняемого кода, который при статической линковке с библиотеками оказывается очень велик.

**27 вопрос. Адресация с использованием регистра перемещения** *Устройство управления памятью*

Устройство управления памятью ( Memory Management Unit – MMU ) – это один из модулей аппаратуры,

отвечающий за адресацию памяти и связанный с процессором и другими устройствами системной шиной. С

точки зрения поддержки описанных концепций адресации, устройство управления памятью – это аппаратура,

преобразующая логический адрес

(полученный по общей шине от

процессора) в физический (реальный

адрес в памяти, по которому и

происходит обращение).

Аппаратура MMU использует значение регистра перемещения, содержащего адрес начала области памяти, выделенной ОС для программы пользователя. MMU

добавляет значение регистра перемещения к (логическому) адресу, сгенерированному пользовательской программой, получая в результате физический адрес.

Программа пользователя работает только с логическими адресами и не "видит" физических адресов.

Схема адресации и преобразования логического адреса в физический с

использованием регистра перемещения изображена →

**28 вопрос. Оверлейная структура двухпросмотрового ассемблера** *Оверлейная структура программы*

* + ранних ОС, в особенности – для персональных компьютеров, для пользовательского процесса были вынужденно введены очень жесткие ограничения по памяти, - например, в MS DOS – не более 640 килобайт. При таком дефиците основной памяти, если программа оказывается настолько велика, что полностью не помещается
* память максимально разрешенного объема, необходимо предпринимать специальные меры при разработке программы, чтобы разбить ее на непересекающиеся группы модулей, такие. что в каждой группе модули логически взаимосвязаны и должны присутствовать в памяти одновременно, модули же разных групп не обязательно должны вместе загружаться в память. Во время исполнения такой программы должен использоваться специальный системный механизм, называемый оверлейная структура ( overlay ,дословно – наложение ), обеспечивающий поочередную загрузку в одну и ту же область памяти то одной, то другой исполняемой группы модулей. Простая программа, которая выполняет эти действия, называется драйвер оверлея ( overlay driver ).Интегрированная среда разработки Турбо Паскаль обеспечивала специальные опции компилятора, которые позволяли явно указывать модули, входящие в каждый оверлей.

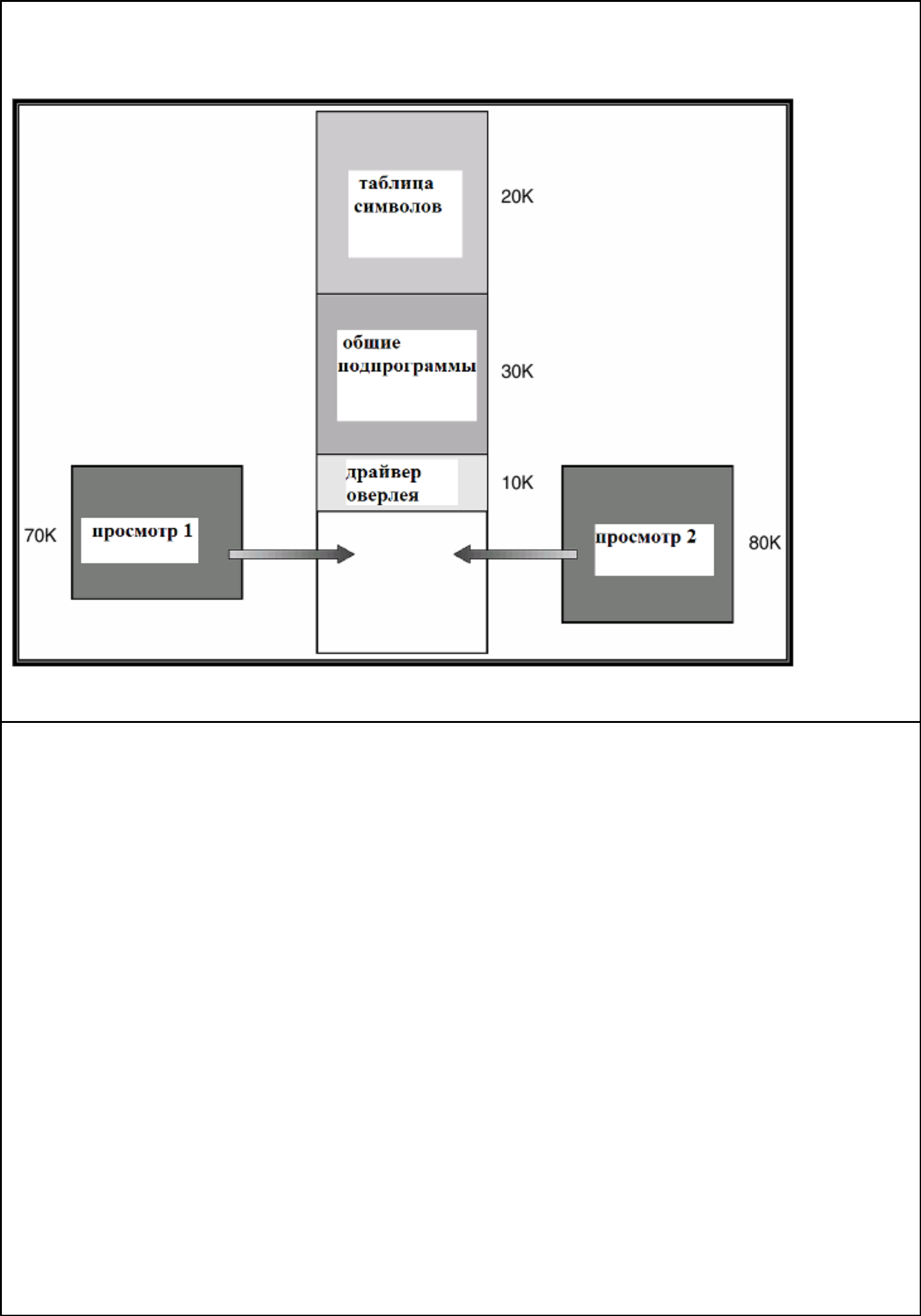
Типичный для ранних компьютеров и ОС пример программы с оверлейной структурой – двухпросмотровый ассемблер. На первом просмотре он преобразует исходный ассемблерный код в промежуточное представление, которое программа второго просмотра ассемблера получает на входе. Полностью весь ассемблер (оба просмотра) в память не помещался, и пришлось применить оверлейную структуру. Данный пример иллюстрируется …

Рис. 15.3. Оверлейная структура двухпросмотрового ассемблера.

**29 вопрос. Откачка и подкачка. Файл подкачки. организовано смежное распределение памяти. Схема**

**откачки и подкачки**

*Откачка и подкачка*

Пользовательский процесс может находиться в различных состояниях во время обработки системой. В частности, процесс может быть некоторое время неактивным, если, например, он исполняется в режиме разделения времени, и пользователь за терминалом обдумывает следующую команду или редактирует исходный код своей программы. В подобных случаях процесс может быть откачан операционной системой на диск, в связи

* тем, что занимаемая им память оказывается необходимой в данный момент для другого, активного, процесса. Откачка и подкачка (swapping) – это действия операционной системы по откачке (записи) образа неактивного процесса на диск или подкачке (считыванию) активного процесса в основную память. Необходимость выполнения подобных действий вызвана нехваткой основной памяти.

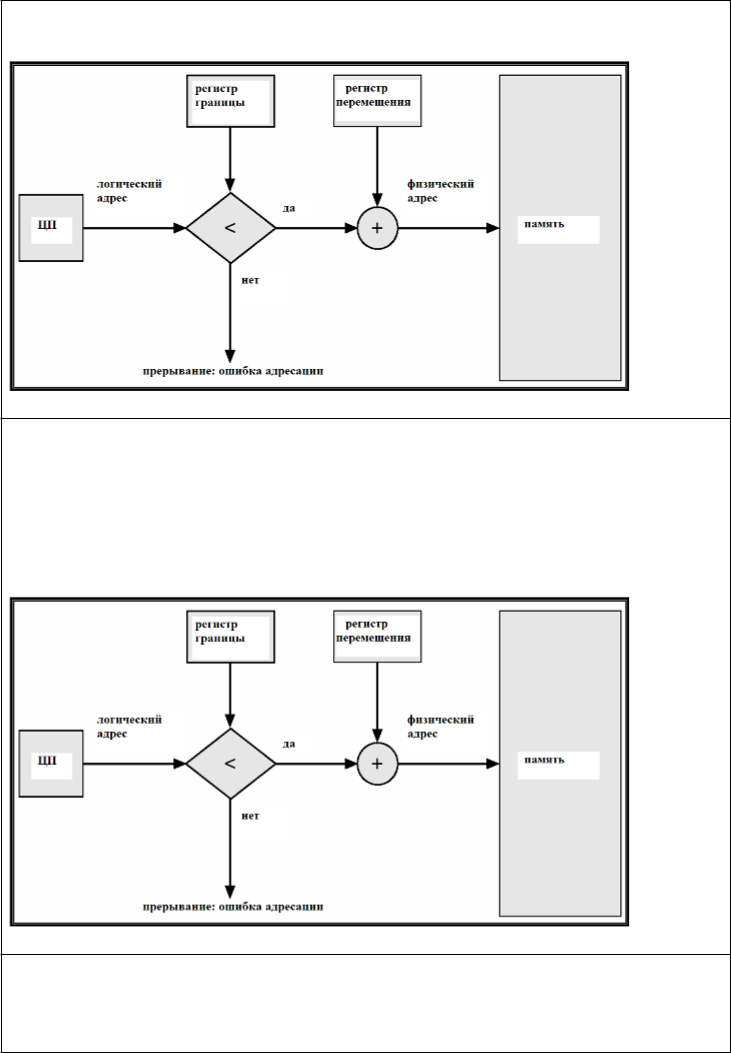
*Файл откачки.*

Файл откачки (backing store) - область дисковой памяти, используемая операционной системой для хранения образов откачанных процессов. Файл откачки организуется максимально эффективно: обеспечивается прямой доступ ко всем образам процессов в памяти (например, через таблицу по номеру процесса).

Популярная разновидность стратегии откачки и подкачки – roll out / roll in: откачка и подкачка на базе приоритетов; более приоритетные процессы исполняются, менее приоритетные – откачиваются на диск. Наибольшие временные затраты на откачку – это затраты на передачу данных: полный образ процесса может занимать большую область памяти. Общее время откачки пропорционально размеру откачиваемых данных.

*Смежное распределение памяти.*

Наиболее простая и распространенная стратегия распределения памяти – смежное распределение памяти – распределение памяти для пользовательских процессов в одной смежной области памяти. Основная память разбивается на две смежных части (partitions), которые "растут" навстречу друг другу: резидентная часть ОС и вектор прерываний – по меньшим адресам. Для пользовательских процессов память распределяется в одном и том же смежном участке памяти. Для каждого процесса регистр перемещения указывает на начало выделенной

ему области памяти, регистр границы содержит длину диапазона логических адресов. Каждый логический адрес должен быть меньше содержимого регистра границы. Физический адрес вычисляется аппаратно как сумма логического адреса и значения регистра перемещения.

**Рис. Адресация с аппаратной поддержкой регистров перемещения и границы.**

**30 вопрос. Схема адресации с аппаратной поддержкой регистров перемещения и границы** *Смежное распределение памяти.*

Наиболее простая и распространенная стратегия распределения памяти – смежное распределение памяти – распределение памяти для пользовательских процессов в одной смежной области памяти. Основная память разбивается на две смежных части (partitions), которые "растут" навстречу друг другу: резидентная часть ОС и вектор прерываний – по меньшим адресам. Для пользовательских процессов память распределяется в одном и том же смежном участке памяти. Для каждого процесса регистр перемещения указывает на начало выделенной ему области памяти, регистр границы содержит длину диапазона логических адресов. Каждый логический адрес должен быть меньше содержимого регистра границы. Физический адрес вычисляется аппаратно как сумма логического адреса и значения регистра перемещения.

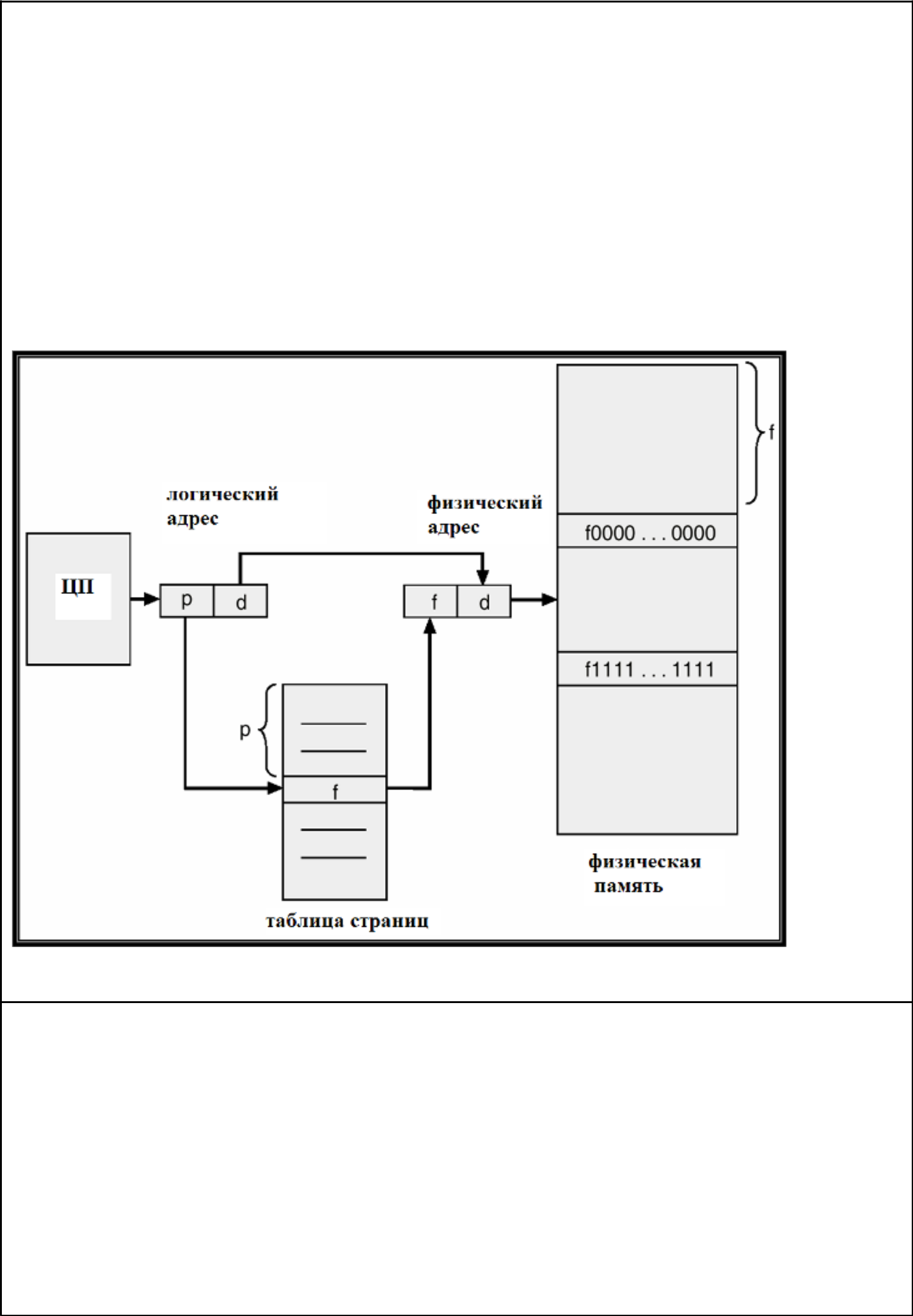
**Рис. Адресация с аппаратной поддержкой регистров перемещения и границы.**

**31 вопрос. Архитектура трансляции адресов при страничной организации** *Страничная организация*

Страничная организация (paging) – стратегия управления памятью, при которой:

логическая память делится на страницы – смежные области одинаковой длины, обычно – степень 2 (например,

1. слов);

физическая память, соответственно, делится на фреймы такого же размера; распределение логической памяти происходит с точностью до страницы; физическая память процесса может не быть непрерывной;

связь между логической и физической памятью процесса осуществляется с помощью таблицы страниц – системной структуры, выделяемой процессу для трансляции его логических адресов в физические.

При страничной организации ОС хранит информацию обо всех свободных фреймах. Поскольку память выделяется с точностью до страницы, возможна внутренняя фрагментация (см. п. 16.5).

Цели страничной организации – обеспечить возможность не смежного распределения физической памяти для процессов, а также расширить пространство логической памяти.

При страничной организации логический адрес обрабатывается системой особым образом – как структура (p, d):его старшие разряды обозначают номер страницы, младшие – смещение внутри страницы. Номер страницы (p) трактуется как индекс в таблице страниц, соответствующий элемент которой содержит базовый адрес начала страницы в физической памяти. Смещение внутри страницы (d) добавляется к ее базовому адресу. В результате формируется физический адрес, передаваемый в устройство управления памятью.

Архитектура трансляции адресов при страничной организации изображена на рис. 16.3.

Рис. 16.3. Архитектура трансляции адресов при страничной организации.

**32 вопрос. Таблицы страниц. Защита памяти**

*Реализация таблицы страниц*

Использование ассоциативной памяти.Таблица страниц – непрерывная область физической памяти. В системе имеется базовый регистр таблицы страниц (page table base register – PTBR),указывающий на таблицу страниц и хранящий ее длину.

Таким образом, при страничной организации любой доступ к памяти требует фактически не одного, а двух обращений в память – одно в таблицу страниц, другое – непосредственно к данным или команде. В этом – некоторый недостаток и неэффективность страничной организации, по сравнению с более простыми методами управления памятью.

* системах с теговой архитектурой, например, "Эльбрус", регистр таблицы страниц ( регистр таблицы страниц пользователя – РТСП) содержит дескриптор таблицы страниц, который, кроме ее адреса, содержит также ее длину.

Проблема двух обращений решается введением ассоциативной памяти (cache) страниц,называемой также буфер трансляции адресов (translation lookaside buffer – TLB).Ассоциативная память, по существу, является

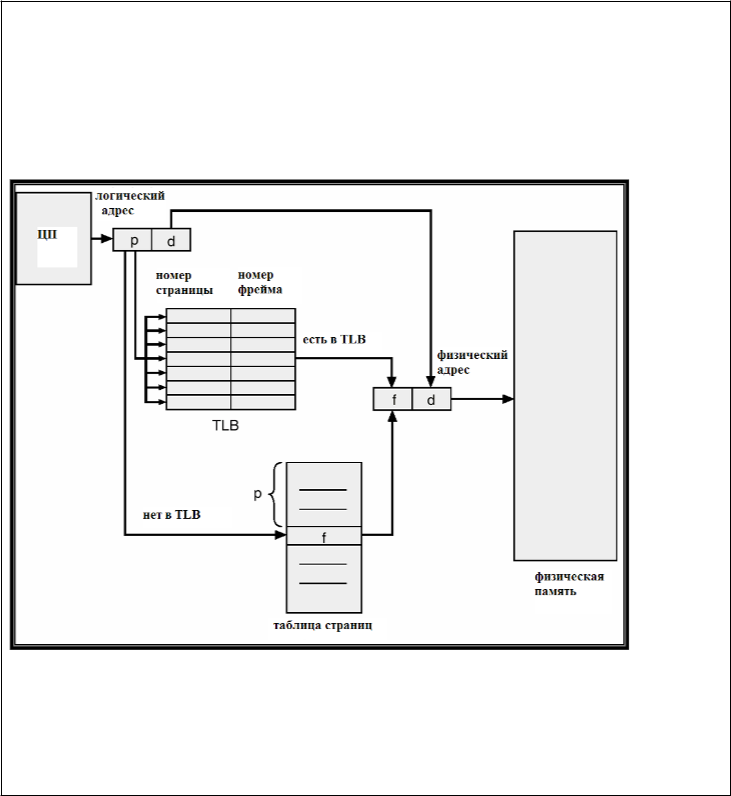
ассоциативным списком пар вида: (номер страницы, номер фрейма).Ее быстродействие значительно выше, чем у основной памяти и у регистров.

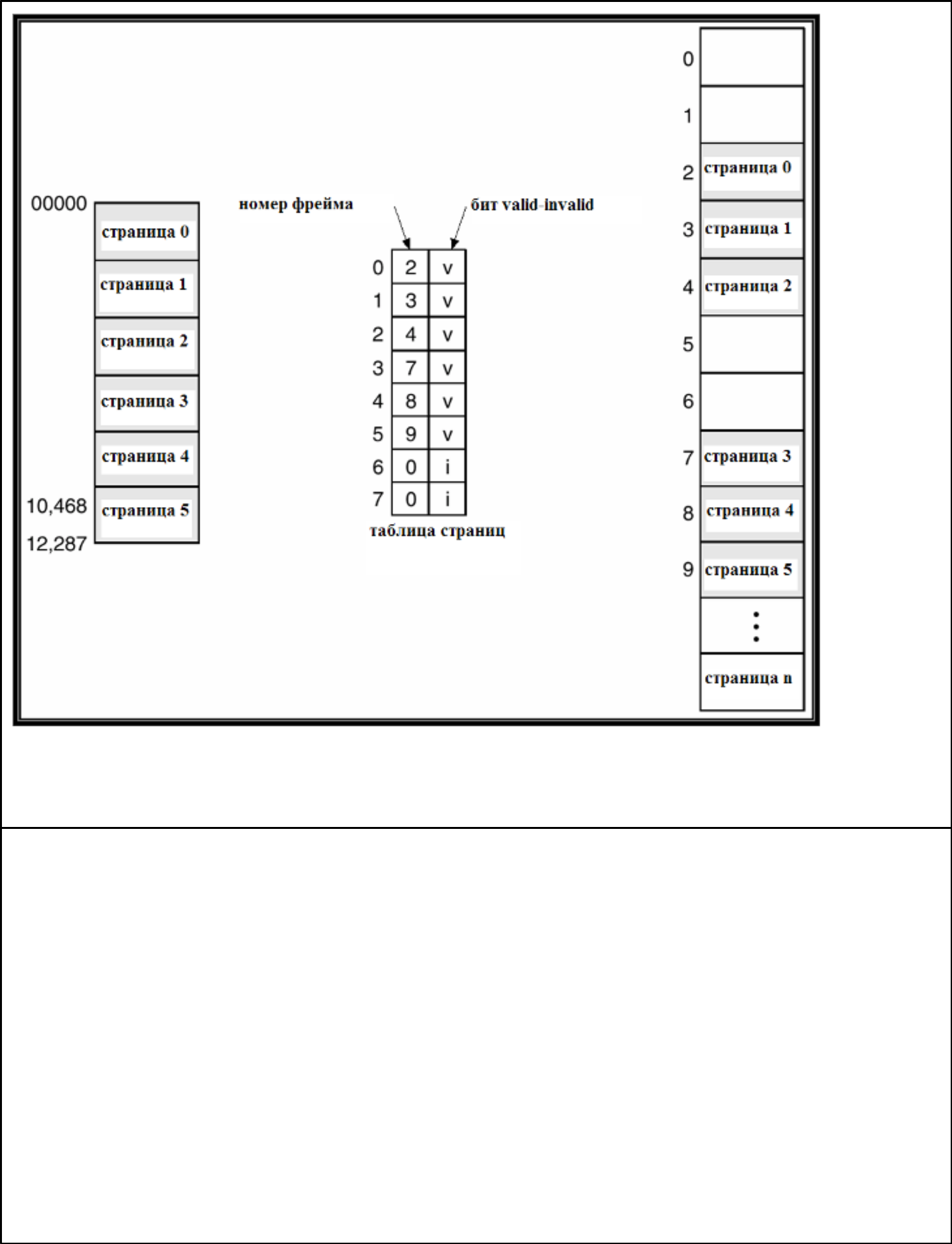
Схема трансляции адресов с использованием ассоциативной памяти изменяется: если номер страницы из логического адреса найден в ассоциативной памяти, то из ее элемента извлекается соответствующий номер фрейма.Если же номер страницы отсутствует в ассоциативной памяти, он выбирается обычным образом из таблицы страниц, но заносится в ассоциативную память. Таким образом, в ассоциативной памяти накапливается информация о наиболее часто используемых страницах.

Модифицированная схема трансляции адресов с использованием TLB иллюстрируется рис. 16.7.

*Защита памяти*

При адресации с помощью страничной организации возможно, что логический адрес сформирован неверно, и его номер страницы выходит за пределы логической памяти процесса. Защита от неверной адресации может быть реализована хранением и проверкой дополнительного бита valid-invalid в каждом элементе таблицы страниц. Значение valid указывает, что страница с данным номером принадлежит логической памяти процесса, значение invalid – что это не так.

Организация защиты памяти процесса с помощью бита valid-invalid иллюстрируется на рис. 16.8



* примере процесс имеет 6 логических страниц с номерами от 0 до 5. Таблица страниц имеет длину 8 (с элементами от 0 до 7). Элементы 6 и 7 не соответствуют логическим страницам процесса, поэтому в них биты valid-invalid установлены в значение invalid. Поэтому при попытке обращения по логическому адресу с номером страницы 6 или 7 произойдет прерывание по неверной адресации.

**33 вопрос. Структура таблицы страниц. Хешированные таблицы страниц** *Структура таблицы страниц*

Иерархические таблицы страниц.Таблицы страниц в операционных системах могут быть по-разному организованы, при сохранении общих принципов их использования, описанных ранее. Рассмотрим далее три основных способа организации таблиц страниц – иерархические таблицы страниц, хешированные таблицы страниц и инвертированные таблицы страниц.

Чаще всего используются иерархические таблицы страниц.При их использовании логическое адресное пространство разбивается на несколько таблиц страниц (иначе говоря, используются таблицы таблиц страниц ). Наиболее простой и распространенный метод – двухуровневая таблица страниц.

При обычной организации таблицы страниц, логический адрес (для 32-разрядной архитектуры, при размере страницы 4 килобайта = 4096 байтов) разбивается на номер страницы (20 битов) и смещение внутри страницы (12 битов).

При двухуровневой организации таблицы страниц, таблица страниц верхнего уровня сама делится на страницы, поэтому логический адрес будет иметь вид: (p1, p2, d),где p1 – индекс во внешней таблице страниц, p2 – смещение внутри страницы для внешней таблицы страниц, d – смещение внутри страницы (адресуемой по внутренней таблице страниц). При тех же предположениях об архитектуре и размере страницы, p1 и p2 будут занимать по 10 битов.

Организация двухуровневых таблиц страниц изображена на рис. 16.9.

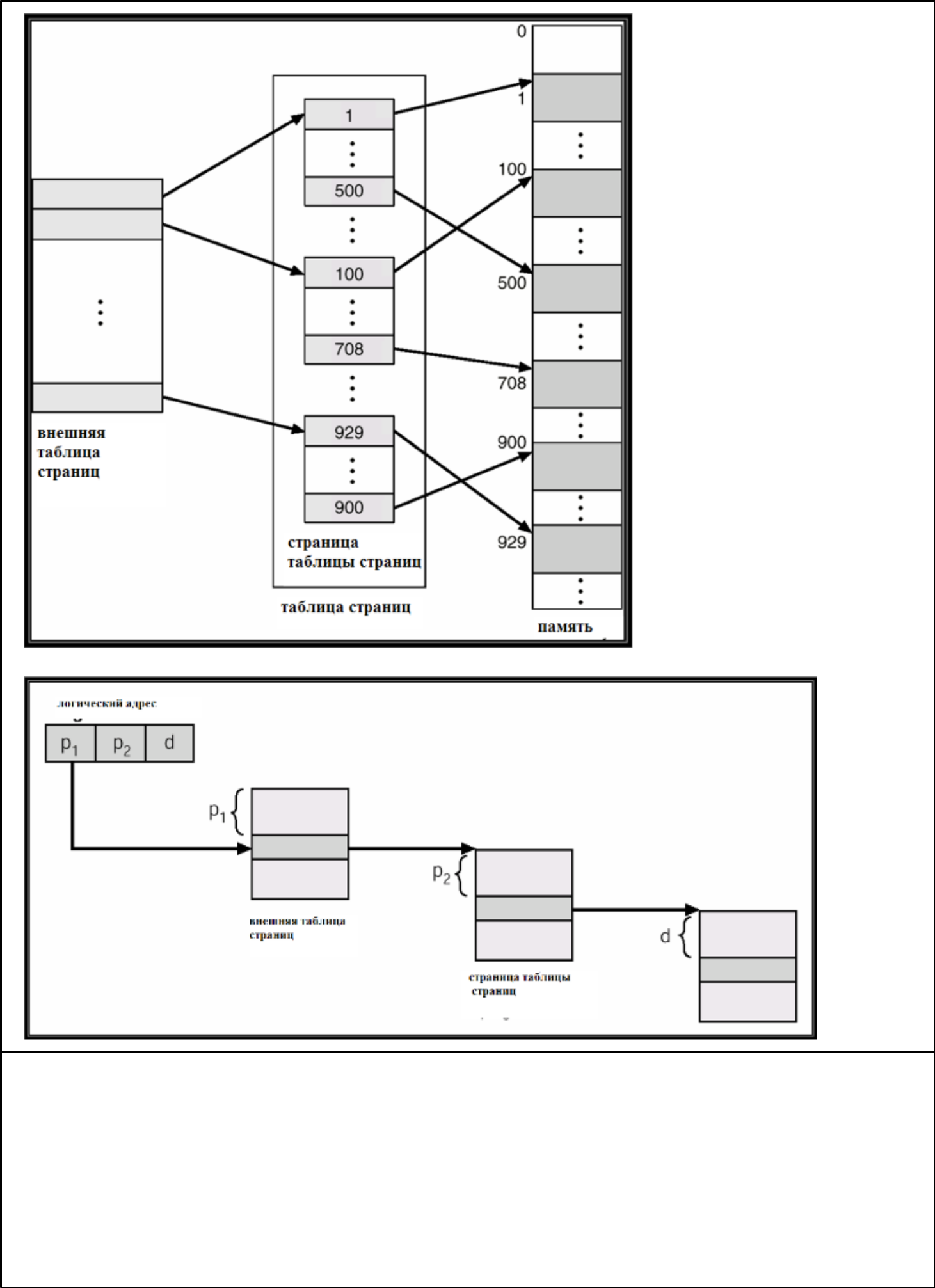
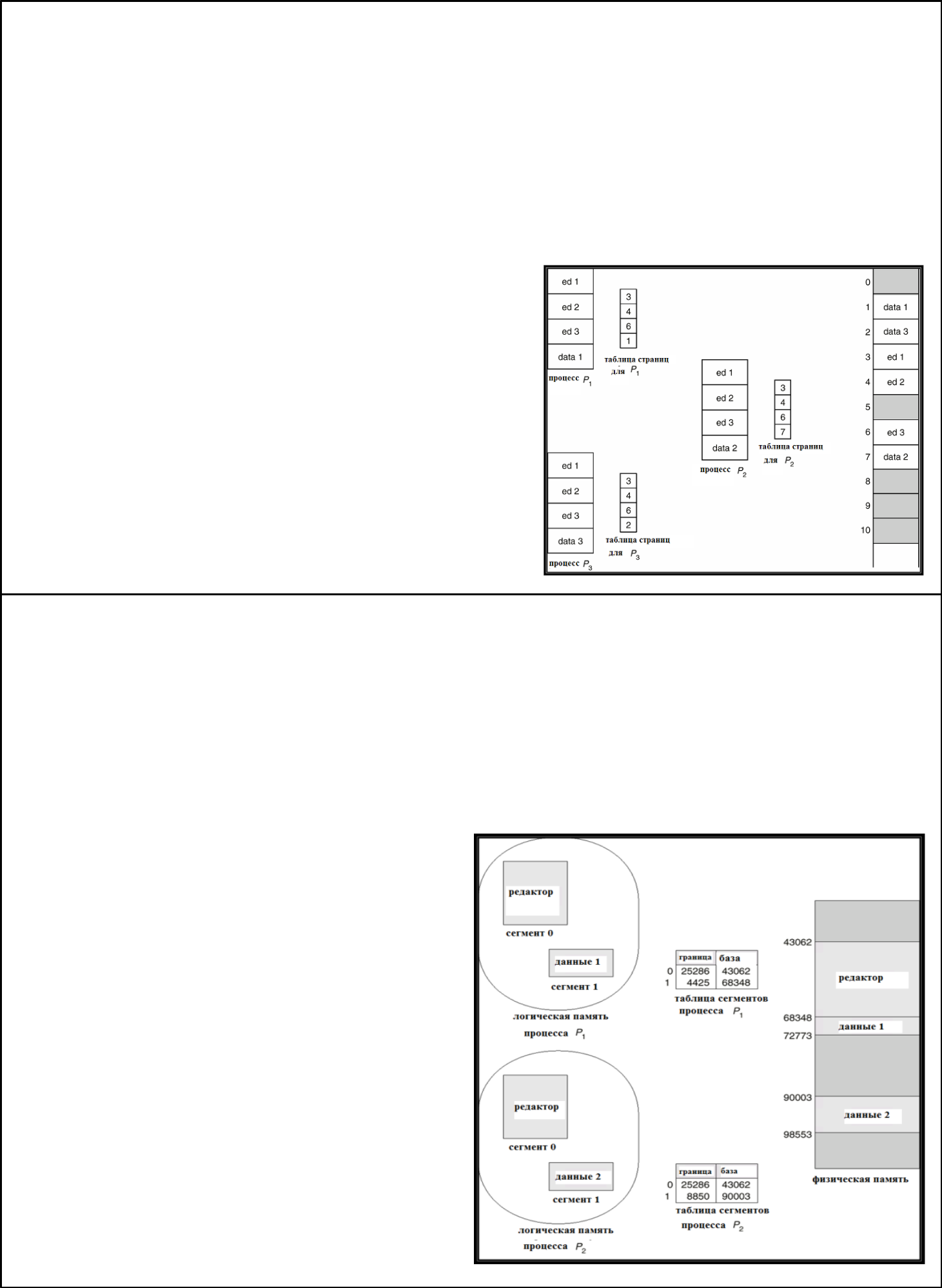


Схема адресной трансляции по двухуровневой таблице страниц иллюстрируется рис. 16.10.

**34 вопрос. Откачка и подкачка. Файл подкачки. Разделяемые страницы**

*Откачка и подкачка.* Пользовательский процесс может находиться в различных состояниях во время обработки системой. В частности, процесс может быть некоторое время неактивным, если, например, он исполняется в режиме разделения времени, и пользователь за терминалом обдумывает следующую команду или редактирует исходный код своей программы. В подобных случаях процесс может быть откачан операционной системой на диск, в связи с тем, что занимаемая им память оказывается необходимой в данный момент для другого, активного, процесса.

Откачка и подкачка (swapping) – это действия операционной системы по откачке (записи) образа неактивного процесса на диск или подкачке (считыванию) активного процесса в основную память. Необходимость выполнения подобных действий вызвана нехваткой основной памяти.

*Файл откачки (backing store)* - область дисковой памяти, используемая операционной системой для хранения образов откачанных процессов. Файл откачки организуется максимально эффективно: обеспечивается прямой доступ ко всем образам процессов в памяти (например, через таблицу по номеру процесса).

*Разделяемые страницы.* Разделяемые (совместно используемые) страницы – логические страницы, используемые несколькими процессами и имеющие один и тот же номер в таблице страниц каждого из процессов. Данный механизм полезен для экономии памяти, так как позволяет загружать в память в единственном экземпляре не изменяемую информацию, необходимую нескольким процессам – например, код и массивы констант.

Пример использования разделяемых страниц тремя процессами приведен на рисунке →. Три процесса используют одни и те же коды трех редакторов ed1, ed2, ed3, логические страницы которых имеют для обоих процессов номера 0, 1 и 2. Кроме того, каждый процесс использует свои индивидуальные данные data1, data2, data3.

**35 вопрос. Принципы сегментной организации памяти, использования разделяемых сегментов**

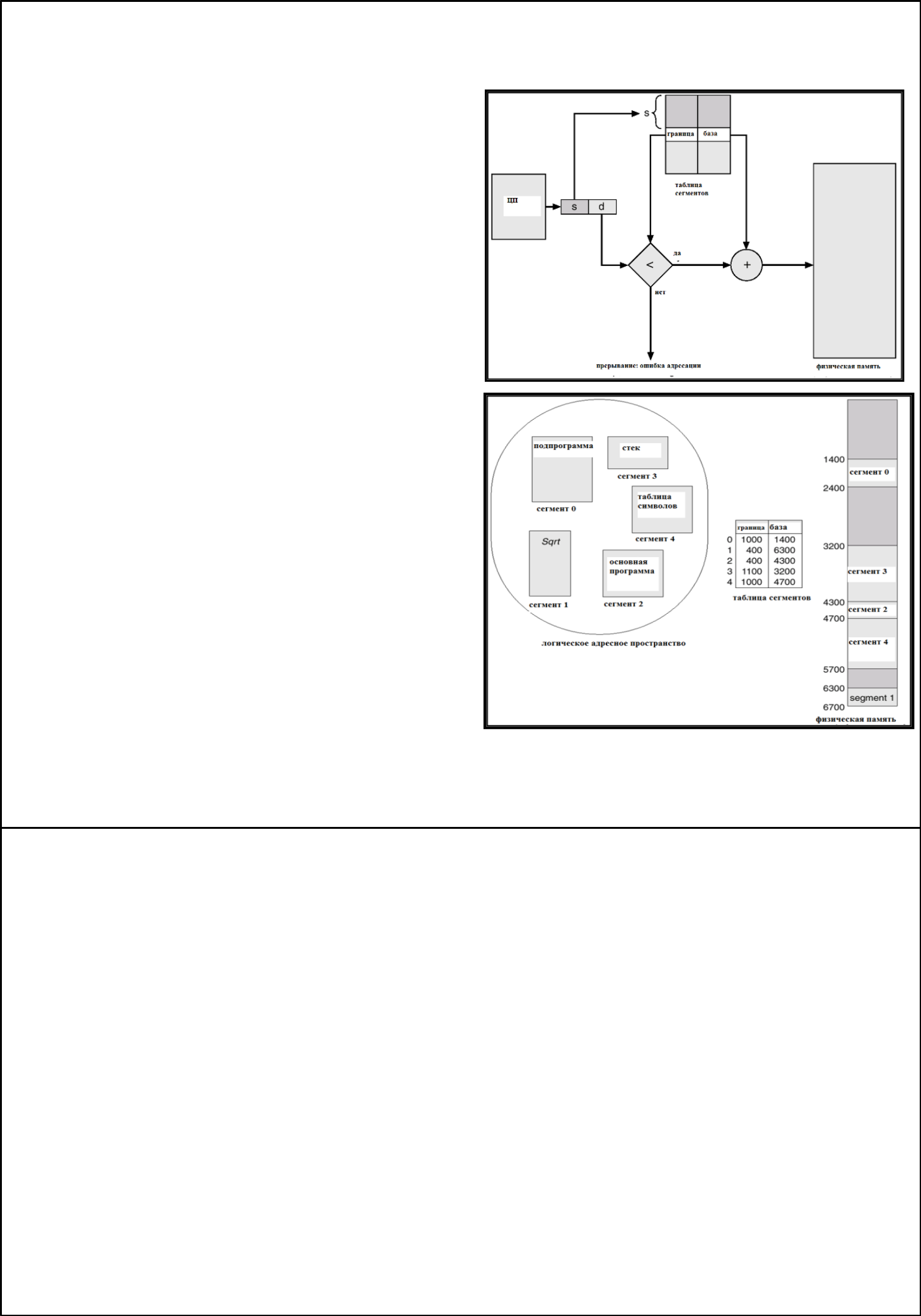
*Принципы сегментной организации памяти.* Сегментная организация памяти (segmentation) - схема распределения памяти в виде сегментов переменной длины, соответствующая пользовательской трактовке распределения памяти, т.е. логической структуре программ и данных. С точки зрения пользователя (разработчика программы), программа – это набор модулей кода и данных, каждому из которых должен соответствовать свой сегмент в памяти. Сегмент – логическая единица распределения памяти, предназначенная для размещения в памяти одного модуля программного кода или данных. Например, в виде сегментов памяти могут быть представлены: основная программа, процедура, функция, метод, объект, набор локальных переменных, набор глобальных переменных, общий блок данных (например, COMMON-блок в языке FORTRAN), стек, таблица символов, массив.

Возможен общий доступ (sharing) нескольких процессов к одному и тому же сегменту, т.е. поддерживается концепция разделяемых сегментов. При этом логический номер общего сегмента для разных процессов будет одним и тем же.

*Архитектура сегментной организации памяти.* Многие принципы архитектуры сегментной организации схожи с принципами страничной организации, однако во всех случаях приходится учитывать, что длина сегмента переменна, и хранить ее в явном виде в таблицах.

Стратегии распределения памяти при сегментной организации: метод первого подходящего или метод наиболее подходящего.

*Пример использования разделяемых сегментов.* В примере два процесса используют общий сегмент – код редактора, который имеет в таблицах сегментов обоих процессов один и тот же номер – 0. Кроме того, каждый из процессов имеет собственный сегмент данных с номером 1, однако эти сегменты, естественно, у обоих процессов разные.

**36 вопрос. Схема адресации при сегментной организации памяти. Пример сегментной организации памяти**

Логический адрес при сегментной организации памяти - пара:

*<segment-number, offset>*, где segment-number – номер

сегмента, offset – смещение в сегменте.

Таблица сегментов – служит для отображения логических адресов в физические при сегментной организации памяти.

Каждый ее элемент содержит следующую информацию:

* base – начальный адрес сегмента в оперативной (физической) памяти;
* limit – длину сегмента.

Базовый регистр таблицы сегментов - segment-table base register (STBR) содержит адрес таблицы сегментов в памяти.

Регистр длины таблицы сегментов - segment-table length register (STLR) содержит число сегментов, используемое программой.

Номер сегмента s корректен, если s < STLR.

*Схема.* На рисунке приведена схема адресации при сегментном распределении памяти. Логический адрес (s, d),где s – номер сегмента, d – смещение внутри сегмента, обрабатывается следующим образом. По номеру сегмента s происходит обращение в таблицу сегментов, и определяется база сегмента – его начальный адрес в основной памяти. Смещение d сравнивается с длиной (границей) сегмента. Если оно меньше, то оно складывается с базой, и в результате получается физический адрес, по которому и происходит обращение

* память. Если смещение больше или равно базе, происходит прерывание – ошибка адресации. На схеме не показана еще одна проверка – для номера сегмента s проверяется, что он не превосходит значения регистра длины таблицы сегментов, иначе – прерывание.

*Пример сегментной организации памяти*.

* + примере программа использует пять сегментов с номерами от 0 до 4: сегмент 0 – подпрограмма, сегмент 1 – стандартные подпрограммы ( sqrt и др.), сегмент 2 – основная программа, сегмент 3 – стек, сегмент 4 – таблица символов. Показано размещение сегментов в памяти.

**37 вопрос. Понятие файла. Структура файла. Методы доступа к файлам**

*Понятие файла.* Файл (file) – это смежная область логического адресного пространства; это данные или программа, имеющие имя и хранящиеся в долговременной памяти как единое целое.

Каждый файл имеет свой тип, определяющий, какая информация хранится в файле. Основные типы файлов – программа (код) или данные. Данные подразделяются на числовые, символьные (текстовые) и двоичные ( произвольная информация).

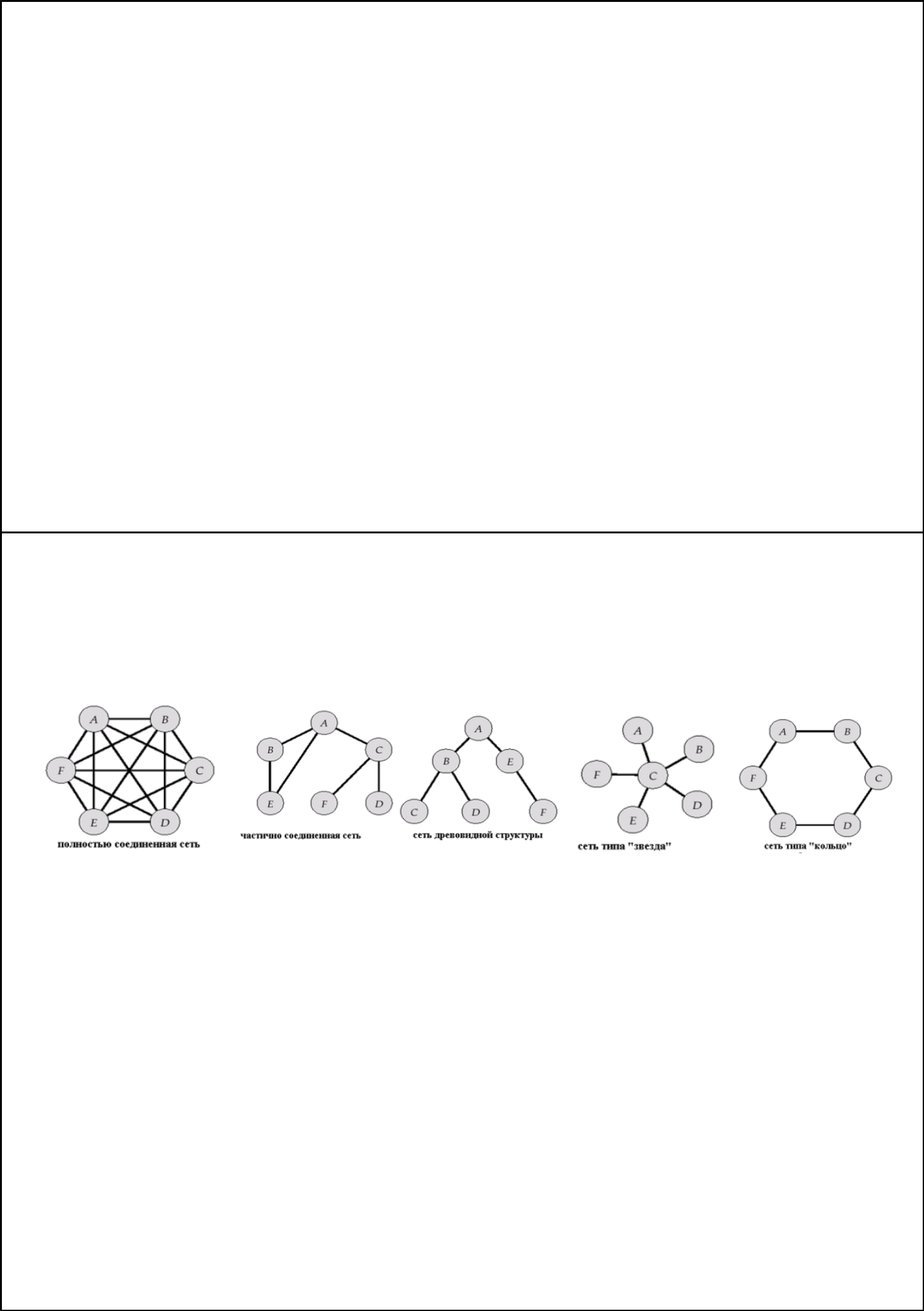
*Структура файла.* В различных системах приняты различные точки зрения на структуру файлов. Наиболее простую и унифицированную точку зрения из них предложили авторы системы UNIX: файл – это последовательность слов или байтов. Файлы можно условно подразделять на файлы простой и сложной структуры (хотя точка зрения на структуру файла зависит от той программы, которая его обрабатывает).

Файлы простой структуры состоят из последовательности записей (records) – элементарных единиц, в терминах которых выполняются операции обмена с файлом. Записи могут быть:

* строками, если это текстовый файл ;
* двоичными данными фиксированной длины ;
* двоичными данными переменной длины.

Файлы сложной структуры могут быть самого разного вида, например:

* отформатированным документом Microsoft Office (такой файл, кроме собственно текста, содержит управляющие символы переключения шрифтов, цветов и т.д.);
* загрузочным модулем реального или виртуального двоичного кода, например, portable executable (PE) -файлом для платформы .NET;
* class-файлом для платформы Java; подобные файлы состоят из нескольких секций, содержат внутренние ссылки и таблицы и т.д.

Сложная структура файла может быть смоделирована записями путем добавления соответствующих управляющих символов.

Файлы интерпретируются операционной системой или программами их обработки.

*Методы доступа к файлам*

Файл последовательного доступа – это файл, доступ к которому возможен только позиционированием на начало и конец и затем операциями обмена вида считать или обновить следующую ( предыдущую ) запись.

Файл прямого доступа – это файл, для которого возможен непосредственный доступ по номеру записи и операция обмена с явным указанием номера записи.

* любом случае, при выполнении обмена с файлом всегда существует некоторая текущая позиция по файлу, указывающая на некоторую запись, на позицию перед началом или после конца файла. В операциях над файлом последовательного доступа произвольная установка позиции не допускается, а разрешены только операции, автоматически передвигающие текущую позицию на следующую (предыдущую) запись.

Типичные операции последовательного доступа: read next

write next

reset – установка на начало файла для чтения rewrite – установка на начало файла для записи. Типичные операции прямого доступа:

read n write n

position to n – позиционирование на запись с номером n read next

write next

rewrite n, где n - относительный номер блока (записи).

**38 вопрос. Сетевые топологии. Стратегии маршрутизации**

Машины в системе могут быть физически соединены разнообразными способами – например, проводной связью (кабелями "витая пара" или коаксиальными, волоконно-оптическими кабелями, телефонными кабелями), беспроводной связью – Wi-Fi / Wi-MAX, Bluetooth, инфракрасной связью и др.

Различные топологии представляются в виде графов, вершины которых соответствуют машинам. Дуга из вершины A в вершину B соответствует непосредственному соединению двух машин.

Основные топологии сетей, изображенные на схемах, - полностью соединенная сеть (любая машина соединена с любой другой), частично соединенная сеть, сеть древовидной структуры, сеть типа звезда, сеть типа кольцо.

*Стратегии маршрутизации.* Маршрутизация (routing) – поиск маршрута для каждого сетевого пакета и направление его по найденному маршруту. Для маршрутизации в сетях используется специальное сетевое оборудование – маршрутизаторы (routers),которые обязательно должны использоваться в больших локальных сетях. Наиболее известные компании, выпускающие маршрутизаторы, - Cisco и 3COM. Рассмотрим возможные стратегии маршрутизации.

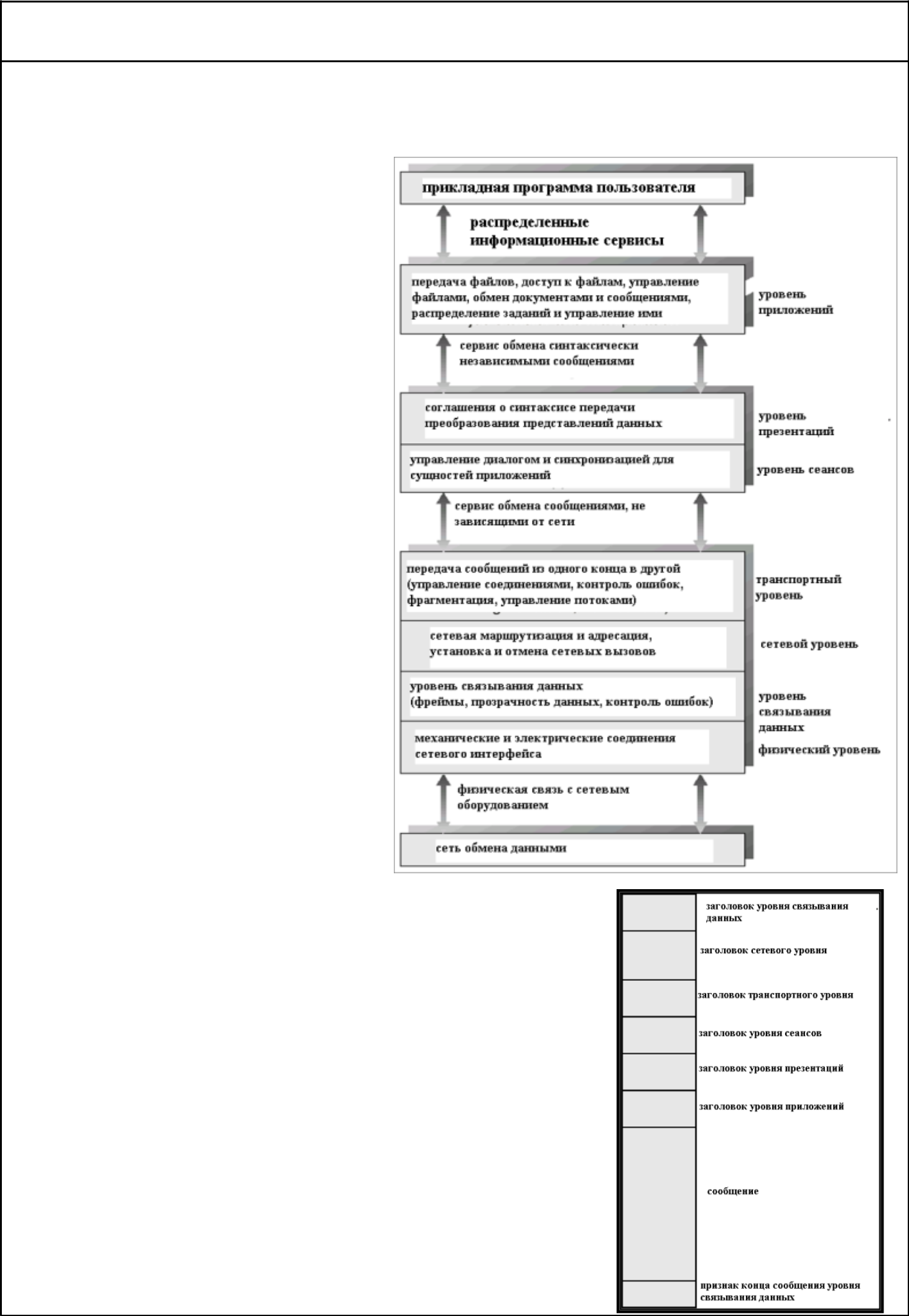
Фиксированная маршрутизация. Путь от A к B задан заранее; он изменяется, только если им невозможно воспользоваться из-за отказов аппаратуры. При этом:

* Поскольку выбирается кратчайший путь, затраты на коммуникацию минимизированы.
* Фиксированная маршрутизация не может быть адаптирована к изменению загрузки.
* Обеспечивается получение сообщений в том же порядке, в каком они были посланы.

Виртуальная цепочка. Путь от A к B фиксируется на время одного сеанса. Различные сеансы, включающие сообщения от A в B, могут иметь различную маршрутизацию. Особенности данного способа маршрутизации:

Это частичное средство адаптации к изменениям загрузки. Обеспечивается получение сообщений в том же порядке, в каком они были посланы.

Динамическая маршрутизация. Путь для отправки сообщения от A к B определяется только в момент отправки данного сообщения. Обычно система посылает сообщение другой системе через соединение, наименее используемое в данный момент времени. Метод адаптирован к изменениям загрузки, так как избегает отправки сообщений через интенсивно используемые соединения. Сообщения при данном методе могут приходить в

другом порядке. Эта проблема может быть решена путем присваивания номера последовательности каждому сообщению (что и реализовано в сетях TCP/IP).

**39 вопрос. Уровни сетевых протоколов (ISO). Структура сетевого сообщения, согласно модели ISO. Уровни протокола TCP/IP.**

Уровни сетевых протоколов (ISO). Сетевой протокол – это набор команд (операций) для взаимодействия узлов в

сети. Ввиду сложности организации сетей,

используемая модель сетевых протоколов

содержит большое число уровней абстракции.

Согласно стандарту коммуникационной модели ISO, коммуникационная сеть подразделяется на следующие основные уровни (layers):

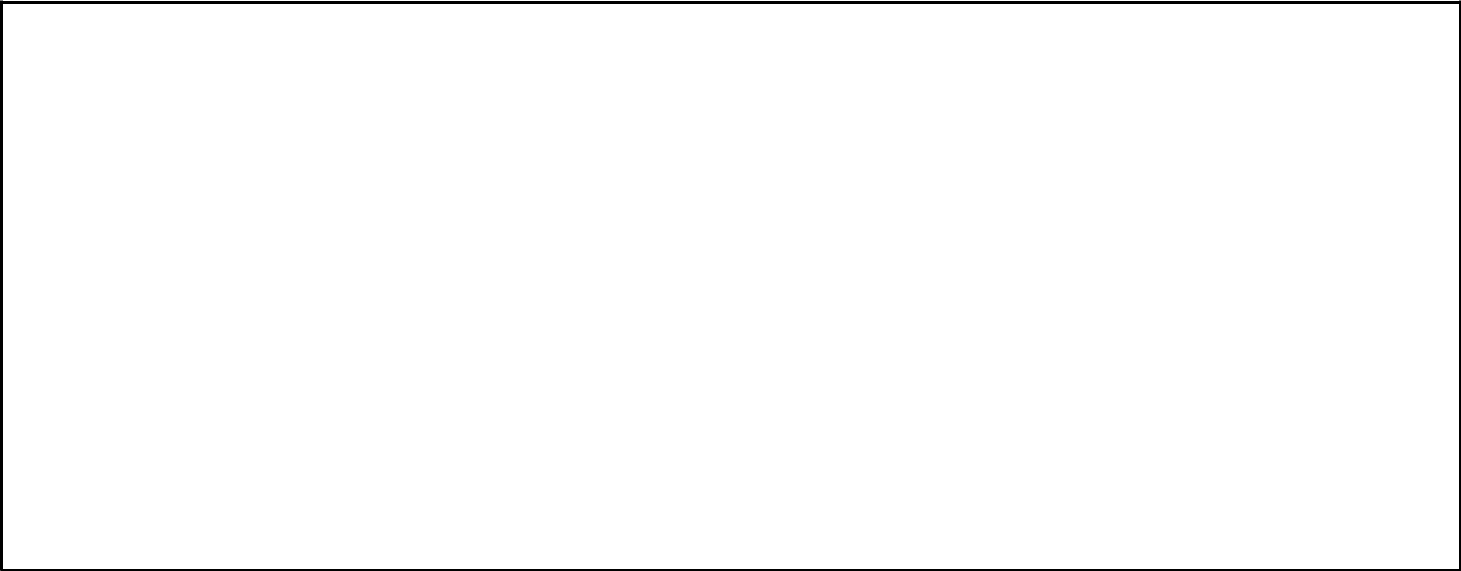
* Физический уровень (physical layer) – механические и электрические устройства для передачи сигналов. Самый нижний уровень сетевой коммуникации. Включает сетевое оборудование - сетевые кабели, разъемы, концентраторы и т.д.
* Уровень (связывания) данных (Data link layer) – обрабатывает фреймы ( frames),или части пакетов фиксированной длины, включая обнаружение ошибок и восстановление после ошибок на физическом уровне.
* Сетевой уровень (network layer) –

обеспечивает соединение и маршрутизацию пакетов в коммуникационной сети, включая обработку адресов исходящих пакетов, декодирование адресов входящих пакетов и

поддержку информации для маршрутизации для соответствующего ответа для изменения уровней загрузки.

* Транспортный уровень (transport layer) – отвечает за сетевой доступ нижнего уровня и за передачу сообщений между клиентами, включая разделение сообщений на пакеты, сопровождение порядка пакетов, поток управления и генерацию физических адресов.
* Уровень сеанса (session layer) – реализует сеансы (sessions), или протоколы коммуникации между процессами.
* Уровень презентаций (presentation layer) – инкапсулирует различие в форматах между различными системами в сети, включая преобразования символов и полудуплексную (дуплексную) связь (эхо-вывод).
* Уровень приложений (application layer) – самый высокий уровень модели сетевых протоколов. Взаимодействует непосредственно с запросами на передачу файлов пользовательского уровня, протоколами удаленных входов и передачи электронной почты, а также со схемами распределенных баз данных.

*Структура сетевого сообщения*. Сообщение имеет заголовки каждого уровня, начиная от уровня связывания данных, затем следует тело сообщения и признак конца сообщения (уровня связывания данных, отвечающего за целостность передачи сообщения). Таким образом, реализация каждого уровня абстракции обрабатывает сообщение, используя заголовок сообщения соответствующего уровня.

*Протокол TCP/IP.* TCP/IP (Transmission Control Protocol / Internet Protocol) – наиболее распространенное семейство протоколов сетевого и транспортного уровня, используемых в Интернете.

Протокол TCP/IP основан на использовании IP-адресов каждого хоста (компьютера), имеющих вид: a.b.c.d (все четыре числа – в диапазоне от 0 до 255) и обеспечивает пересылку по сети пакетов (packets) фиксированного размера, содержащих адрес получателя и номер пакета в сообщении. TCP/IP -протокол обеспечивает транспорт сетевых пакетов, деление сообщения на пакеты отправителем и сборку сообщения из пакетов получателем. IP-адрес может быть закреплен за компьютером постоянно Интернет-провайдером пользователя компьютера, либо присваивается компьютеру динамически (каждый раз – разный) при выходе в Интернет.

Более общее современное название TCP/IP - Internet Protocol Suite.Различаются более новая версия – IPv6 и более старая – IPv4. В версии IPv6 используется IP-адрес не из четырех, а из 6 чисел.

Протокол TCP/IP – синхронный, т.е. получатель ожидает получения каждого пакета и посылает отправителю подтверждение об этом.

Скорость TCP/IP не всегда удовлетворительна, ввиду огромного числа IP-узлов в Интернете. Для оптимизации связи между узлами сети применяются Distributed Hash Tables (DHT) – распределенные хеш-таблицы. В них собственная система имен узлов сети и более быстрого их поиска, чем с использованием TCP/IP протоколов, работающая "поверх" TCP/IP.