Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

Кафедра физики и прикладной математики

КОНСПЕКТ ЛЕКЦИЙ ПО ДИСЦИПЛИНЕ «СИСТЕМНОЕ И ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ»

Составители:
С.И. АБРАХИН
А.В. ДУХАНОВ

Владимир 2010

УДК 004.92
ББК 31а1
К65

Рецензент
Кандидат физико-математических наук,
генеральный директор ООО «Фирма “Инрэко ЛАН”»
К.В. Демидов

Печатается по решению редакционного совета
Владимирского государственного университета

К65 Конспект лекций по дисциплине «Системное и прикладное программное обеспечение» / Владим. гос. ун-т ; сост. : С. И. Абрахин, А. В. Духанов. – Владимир : Изд-во Владим. гос. ун-та, 2010. – 39 с.

Кратко рассматриваются основы архитектуры ЭВМ, устройство и ресурсы, которые используются в работе операционных систем. Подробно описываются подходы к управлению ресурсами ЭВМ на уровне операционных систем. Даётся краткая характеристика прикладному программному обеспечению.

Предназначен для студентов второго курса дневной формы обучения специальностей 010501 – прикладная математика и информатика, 230401 – прикладная математика; бакалавров по направлению 010500 – прикладная математика и информатика.

Ил. 12. Библиогр.: 8 назв.

УДК 004.92
ББК 31а1

Введение

В современном обществе развитие компьютерных и информационных технологий происходит бурными темпами. В настоящее время человеку сложно сориентироваться в огромном многообразии различных программных решений, методов и технологий. Дисциплина «Системное и прикладное программное обеспечение» призвана упорядочить знания в области информационных технологий и рассмотреть «изнутри» работу ЭВМ и системного программного обеспечения. Данный конспект лекций содержит краткие сведения о самом компьютере, являющимся аппаратным средством для обслуживания программного обеспечения, работе операционных систем, прикладном программном обеспечении различного назначения, решающие задачи управления ресурсами.

Дисциплина представлена четырьмя разделами.

- Ресурсы ЭВМ.
- Операционные системы.
- Системы программирования.
- Прикладное программное обеспечение.

В первом разделе рассматриваются основы архитектуры ЭВМ и виды их ресурсов. В нём приводятся принципы Джеймса-фон-Неймана и подробно описывается устройство современных ЭВМ.

Во втором – понятие «Операционная система» и её функции. Также приводятся классификация и назначение операционных систем.

В третьем – приводятся основные подходы управления ресурсами. Рассматриваются вопросы и проблемы управления процессорным временем (управление процессами), описываются методы управления оперативной памятью ЭВМ, подходы управления вводом/выводом.

Последний раздел содержит обзор видов прикладного программного обеспечения и их назначение.

Лекция 1. ОСНОВЫ АРХИТЕКТУРЫ ЭВМ. РЕСУРСЫ ЭВМ

1.1. Основы архитектуры ЭВМ

Архитектура ЭВМ – наиболее общие принципы построения ЭВМ, реализующие программное управление работой и взаимодействием основных ее функциональных узлов.

Общие принципы построения ЭВМ, которые относятся к архитектуре.

1. Структура памяти ЭВМ.
2. Способы доступа к памяти и внешним устройствам.
3. Возможность изменения конфигурации.
4. Система команд.
5. Форматы данных.
6. Организация интерфейса.

Классические принципы построения архитектуры ЭВМ были предложены в работе Дж. фон Неймана, Г. Голдстейга и А. Беркса в 1946 г. и известны как «принципы фон Неймана».

Они сводятся к четырем положениям [1].

1. Использование двоичной системы представления данных.
2. Принцип хранимой программы.
3. Принцип последовательного выполнения операций.
4. Принцип произвольного доступа к ячейкам оперативной памяти.

Рассмотрим их по отдельности.

Первое положение – использование двоичной системы представления данных. Любая информация на ЭВМ хранится в виде последовательности нулей и единиц. Такой способ представления информации для машин наиболее удобен, поскольку она легко обрабатывается («нуль» – нет сигнала, «единица» – сигнал есть). Для двоичной системы существует соответствующий аппарат вычислений, который описан практически в любом учебнике по информатике.

Второе положение – принцип хранимой программы. Как данные программа хранится в виде нулей и единиц, то есть принципиальная разница между программой (исполняемым кодом и

данными) отсутствует. Фон Нейманом была предложена следующая структура логического устройства ЭВМ (рис. 1).

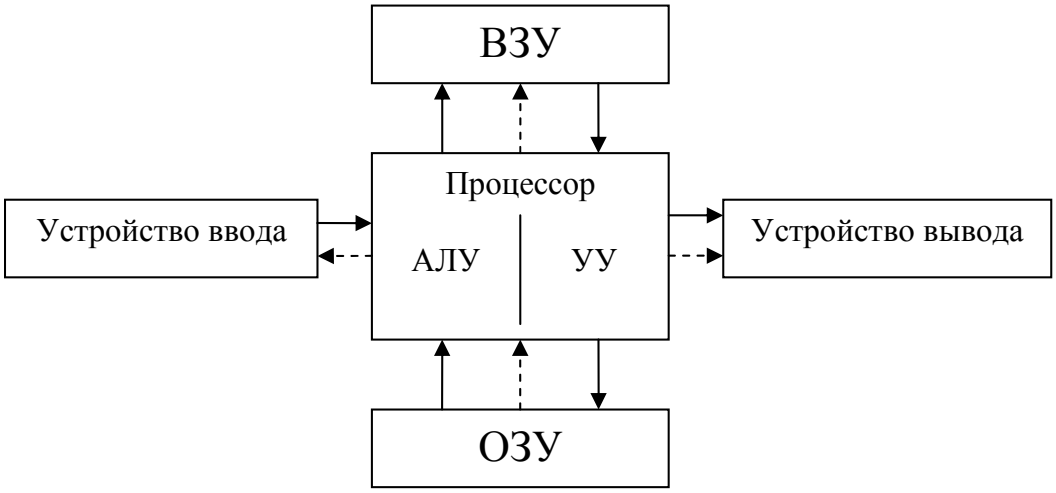

Рис. 1. Схема логического устройства ЭВМ

В современных ЭВМ процессор состоит из арифметико-логического устройства (АЛУ) и устройства управления (УУ). Он осуществляет управление устройствами ввода/вывода, внешними запоминающими устройствами (ВЗУ) и оперативным запоминающим устройством (ОЗУ). Также процессор осуществляет обработку информации, поступающей из устройств. Все виды запоминающих устройств (ЗУ) хранят информацию (данные) и коды программ. В современных ЭВМ ЗУ «многоярусны». ОЗУ хранит информацию, с которой ЭВМ работает непосредственно в данное время. ВЗУ обеспечивает хранение информации больших объемов, чем ОЗУ, но при этом работает значительно медленнее.

Третье положение – принцип последовательного выполнения операций. Одним и тем же устройством, например, процессором операции выполняются последовательно друг за другом. Например, если процессор осуществляет вычисления, а пользователь в это же время вводит данные с клавиатуры, то вычисления прерываются, осуществляется обработка введенных символов и лишь затем возобновляются вычисления.

Четвёртое положение – принцип произвольного доступа к ячейкам оперативной памяти. Структурно основная память состоит из пронумерованных ячеек. Процессору в произвольный мо-

мент времени доступна любая ячейка. Отсюда следует возможность давать имена областям памяти так, чтобы к запомненным в них значениям можно было бы впоследствии обращаться или менять их в процессе выполнения программы с использованием присвоенных имён.

1.2. Устройство современного персонального компьютера

Рассмотрев общую концепцию устройства компьютера, перейдем к изучению устройства современных персональных компьютеров (ПК).

Практически каждый компьютер состоит из системного блока, монитора, клавиатуры, манипуляторов и периферии.

Системный блок основной, если не главный блок ЭВМ [2]. В нем содержатся центральный процессор, материнская плата, контроллеры, накопители и т.д.

Для обеспечения информационного взаимодействия между процессором и накопителями, внешними устройствами и так далее существуют соответствующие контроллеры. Управление монитором процессор осуществляет через видеоконтроллер или видеокарту. Жесткие диски и другие носители информации «общаются» с процессором через контроллер накопителей. И так далее. Фактически контроллер можно рассматривать как специализированный процессор, управляющий работой «вверенного ему» внешнего устройства по специальным встроенным программам обмена. Такой процессор имеет собственную систему команд. Например, контроллер накопителя на гибких магнитных дисках (дисковода) умеет позиционировать головку на нужную дорожку диска, читать или записывать сектор, форматировать дорожку и другое. Результаты выполнения каждой операции заносятся во внутренние регистры памяти контроллера и могут быть в дальнейшем прочитаны центральным процессором.

Таким образом, наличие интеллектуальных внешних устройств может существенно изменять идеологию обмена. Центральный процессор при необходимости произвести обмен выдаёт задание на его осуществление контроллеру. Дальнейший обмен ин-

формацией может протекать под руководством контроллера без участия центрального процессора. Последний получает возможность «заниматься своим делом», то есть продолжать выполнение командных кодов программы.

В материнской плате, а также в системном блоке (в торце) находятся разъёмы, которые обеспечивают физическое присоединение устройств. Логически разъёмы характеризуются портами. Последние выполняют две функции.

1. Служат посредником при передаче данных между компьютером и устройствами ввода/вывода.
2. Выдают процессору сигнал прерывания, по которому начинается процесс прерывания.

Для связи между отдельными функциональными узлами ЭВМ используется **общая шина**, или **магистраль**. Шина состоит из трёх частей.

1. Шина данных, по которой передаётся информация.
2. Шина адреса, определяющая, куда передаются данные.
3. Шина управления, регулирующая обмен информацией.

Вышеприведённые тезисы можно привести к следующей схеме устройства современного ПК и потоков данных (рис. 2).

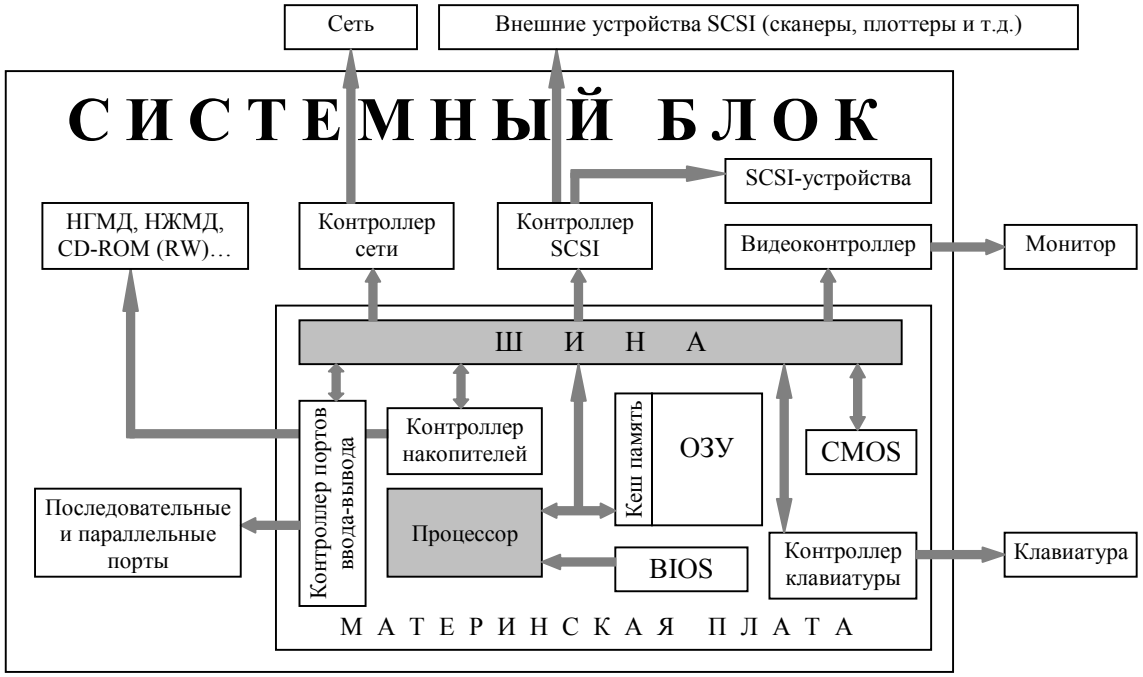

Рис. 2. Схема устройства современного ПК

Поясним, что такое CMOS и BIOS.

CMOS (Complementary Metal-Oxide Semiconductor) – специальная память, обладающая низким энергопотреблением, и предназначенная для хранения параметров конфигурации компьютера.

BIOS (Basic Input-Output System) – постоянная память (ПЗУ), предназначенная для хранения специальной программы, обеспечивающей проверку оборудования ЭВМ, инициализации загрузки операционной системы и выполнения базовых функций по обслуживанию устройств компьютера. В BIOS содержится также программа настройки конфигурации компьютера (SETUP). Она позволяет установить некоторые характеристики устройств компьютера.

Ресурсы ЭВМ: тактовая частота процессора, кеш-память первого и второго уровней, размер оперативной памяти, ёмкость жёсткого диска, аппаратные прерывания, частота системной шины и так далее. Идентифицируются они двумя способами: перед загрузкой операционной системы (нажатием клавиши «Pause»), в операционной системе с помощью специальных диагностирующих программ. Также ресурсы компьютера можно определить по строке спецификаций. Подобные строки чаще всего приводятся в прейскурантах цен на вычислительную технику. При формировании таких строк определёнными стандартами не пользуются. Тем не менее, из содержимого строки спецификации можно легко определить все интересующие ресурсы ЭВМ.

Лекция 2. ОПЕРАЦИОННЫЕ СИСТЕМЫ

Понятие «Операционная система»

Работа на ЭВМ пользователями различных уровней обеспечивается операционной системой (ОС). Она – связующее звено между ЭВМ как набором электронных плат и пользователем.

Все машинные операции, обеспечивающие выполнение работы с файлом, берет на себя операционная система. Таким образом, можно сказать, что ОС берет на себя практически все «низкоуровневые» проблемы: обработка прерываний, управление таймерами

и оперативной памятью и другие. Благодаря ей пользователь предстает не перед реальной аппаратурой со всеми вытекающими последствиями, а перед её абстракцией (упрощённым отображением).

Кроме интерфейсной функции ОС выполняет функцию распределения ресурсов. В соответствии с данным подходом функцией ОС является распределение процессоров, памяти, устройств и данных между процессами, претендующими на эти ресурсы. ОС должна управлять всеми ресурсами вычислительной машины таким образом, чтобы обеспечить максимальную эффективность ее функционирования по заданным критериям: пропускная способность, реактивность системы и так далее. Управление ресурсами включает решение двух общих, не зависящих от типа ресурса задач [3]:

1) планирование ресурса – то есть определение, кому, когда, а для делимых ресурсов и в каком количестве, необходимо выделить данный ресурс;

2) отслеживание состояния ресурса – то есть поддержание оперативной информации о том, занят или не занят ресурс, а для делимых ресурсов – какое количество ресурса уже распределено, а какое свободно.

Используя вышесказанное, можно дать следующее определение операционной системе.

Операционной системой называют комплекс программ, обеспечивающих автоматизацию управления аппаратными ресурсами ЭВМ и выполнение команд пользователя [4].

Для решения этих общих задач управления ресурсами разные ОС используют различные алгоритмы, которые определяют облик системы, включая характеристики производительности, область применения и даже пользовательский интерфейс. Так, например, алгоритм управления процессором в значительной степени определяет, является ли ОС системой разделения времени, системой пакетной обработки или системой реального времени.

Классификация операционных систем

Операционные системы можно классифицировать по многим признакам: способы реализации внутренних алгоритмов управле-

ния ресурсами, методы проектирования, типы аппаратных платформ и другим. На рис. 3 представлена схема классификации ОС.

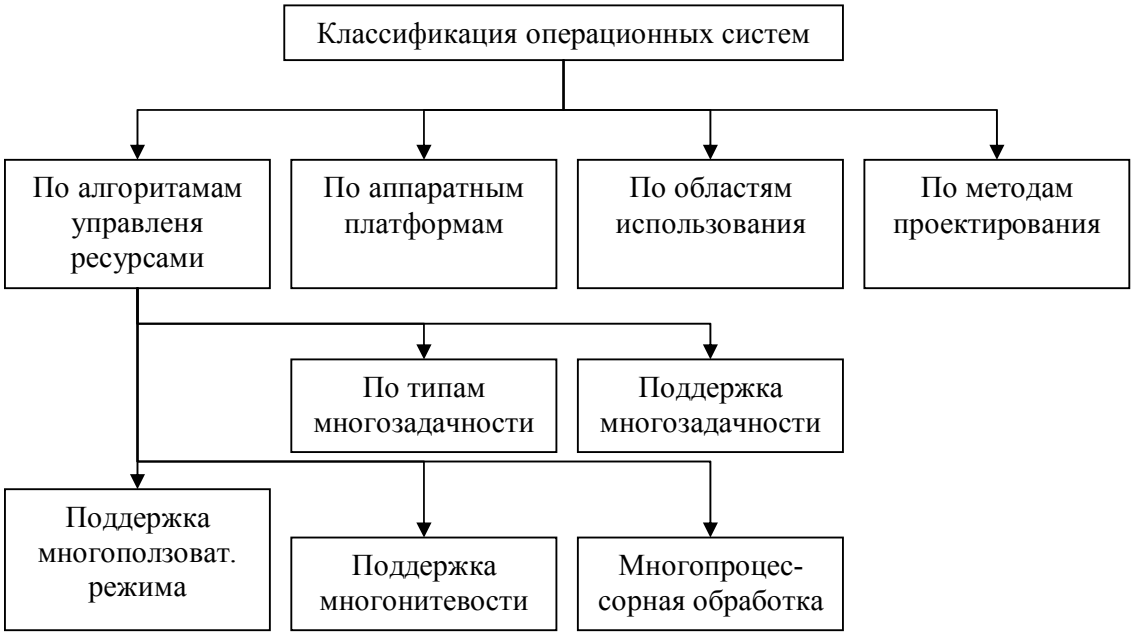

Рис. 3. Классификация операционных систем

По алгоритмам управления ресурсами ОС можно распределить на следующие подклассы:

- по поддержке многозадачности;
- по типам многозадачности;
- по поддержке многопользовательского режима;
- по поддержке многонитевости;
- по алгоритмам многопроцессорной обработки.

По поддержке многозадачности ОС являются однозадачными и многозадачными.

Однозадачные ОС обеспечивают удобное взаимодействие пользователя с ЭВМ, управление периферийными устройствами. В её состав входит файловая система и командный интерпретатор.

Многозадачные ОС, помимо перечисленных выше функций для однозадачных ОС, управляют разделением совместно используемых ресурсов, например, процессором, оперативной памятью, дисковым накопителем и т.д.

По типам многозадачности ОС делятся на системы с *невытесняющей* многозадачностью и *вытесняющей* многозадачностью.

Виды многозадачности определяются способом распределения процессорного времени. Когда речь идет о невытесняющей многозадачности, то механизм планирования процессов сосредоточен в операционной системе. В этом случае активный процесс выполняется до тех пор, пока он не передаст управление операционной системе. При вытесняющей многозадачности механизм планирования распределён между системой и прикладными программами. В этом случае операционная система принимает решение о переключении процессора с одного процесса на другой, а не сами процессы, включая активный.

По поддержке многопользовательского режима ОС подразделяются на *однопользовательские* и *многопользовательские*.

Основное отличие многопользовательских систем от однопользовательских – наличие средств защиты информации каждого пользователя от несанкционированного доступа других пользователей.

По поддержке многонитевости ОС различают с *распараллеливанием вычислений* в рамках одной задачи и без него. Операционные системы с распараллеливанием вычислений разделяют процессорное время на отдельные ветви (нити) процесса.

По алгоритмам многопроцессорной обработки системы характеризуются наличием или отсутствием средств мультипроцессорной обработки – *мультипроцессирования*. Системы с наличием средств мультипроцессирования разделяются на *ассиметричные* и *симметричные*.

Ассиметричные ОС выполняются только на одном процессоре. Ресурсы остальных процессоров распределяются по прикладным программам.

Симметричные ОС полностью децентрализированы и используют ресурсы всех процессоров для выполнения системных задач и прикладных программ.

По аппаратным платформам различают операционные системы для персональных компьютеров, миникомпьютеров, мейнфреймов, кластеров и сетей ЭВМ. Здесь специфика аппаратных средств (конфигурации) определяет типы операционных систем.

Существуют специальные операционные системы, которые могут переносится между ЭВМ различного класса. Такие системы на-

зывают *мобильными*. В них аппаратно-зависимые блоки чётко локализованы. Поэтому при переносе системы переписываются только они.

По областям использования системы классифицируют на следующие:

- системы пакетной обработки;
- системы разделения времени;
- системы реального времени.

Критерием эффективности для *систем пакетной обработки* является пропускная возможность – объём решаемых задач в единицу времени.

Такие системы работают следующим образом. В начале отведённого периода времени (например, рабочего дня) в системе формируется пакет заданий. Для каждого задания известны требования к системным ресурсам. Из пакета формируется набор одновременно выполняемых процессов. Как правило, ОС отбирает задания таким образом, чтобы сбалансировать работу всех устройств ЭВМ. Например, эффективным считается наличие задач с большим объёмом вычислений и задач с интенсивным вводом/выводом данных. Таким образом, система из невыполненных задач выбирает ту, которая наиболее «выгодна» в сложившейся ситуации. Следовательно, пользователь не может интерактивно взаимодействовать с машиной, на которой установлена систем пакетной обработки. Процесс работы сводится к тому, что в начале периода пользователь предоставляет машине задачу и забирает результат в конце периода.

В *системах разделения времени* каждому пользователю предоставляется терминал и некоторая часть процессорного ресурса (квант). При этом пропускная способность машины снижается, но пользователь в это время не изолирован от процесса выполнения его задач. Критериями эффективности систем разделения времени являются удобство и эффективность работы пользователя.

Критерием эффективности *систем реального времени* является реактивность, иными словами, способность в заданный интервал времени отреагировать на некоторое событие. Пример такой системы – система управления экспериментальной установкой. В

определённом интервале времени установка передает данные, которые система предварительно обрабатывает и записывает на диск. Если какой-то фрагмент данных не будет обработан в заданный промежуток времени, а новые данные начнут поступать в ЭВМ, то возникнет аварийная ситуация: потеря данных, контроля над установкой и т.д.

Для этих систем мультипрограммная смесь представляет собой фиксированный набор заранее разработанных программ. Выбор программы на выполнение осуществляется исходя из текущего состояния управляемого объекта или в соответствии с расписанием плановых работ.

Существуют операционные системы, которые способны совмещать в себе системы разных типов. Например, система может выполнять пакетную обработку и осуществлять диалог с пользователем. В этом случае пакетная обработка осуществляется в так называемом *фоновом* режиме.

По методам проектирования ОС классифицируются по следующим базовым концепциям:

- способы построения ядра системы;
- объектно-ориентированный подход построения ОС;
- множественность прикладных сред;
- распределенная организация операционной системы.

По способам построения ядра операционные системы разделяются на системы *с монолитным ядром* и *системы с микроядром*.

Монолитное ядро представляет собой единую программу, работающую в привилегированном режиме. В данном случае не требуется переключения из привилегированного режима в пользовательский и наоборот при переходе с одной процедуры на другую.

Микроядро также работает в привилегированном режиме, но при этом выполняет минимум функций по управлению аппаратурой. Функции ОС более высокого уровня выполняют специализированные компоненты системы – службы (сервисы), работающие в пользовательском режиме. ОС с микроядром более медленные по сравнению с ОС, построенными на базе микроядра, за счет того, что в них осуществляется большое количество переходов из привилегированного режима в пользовательский и наоборот. Но такие

системы более гибки – их можно наращивать, модифицировать, то есть адаптировать для решения задач различного класса и уровня сложности.

Построение ОС на базе объектно-ориентированного подхода (ООП) даёт возможность использовать все его достоинства внутри операционной системы: накопление работающих блоков в виде стандартных объектов, возможность создания новых объектов на базе имеющихся с помощью механизма наследования. Инкапсуляция обеспечивает хорошую защиту данных. ООП позволяет структуризировать системы, состоящие из набора хорошо определённых объектов.

Наличие множественности прикладных сред в рамках одной ОС позволяет выполнять приложения, разработанные для нескольких ОС. Многие современные операционные системы поддерживают одновременно прикладные среды MS-DOS, Windows, UNIX (POSIX), OS/2 или хотя бы некоторого подмножества из этого популярного набора. Такой подход обеспечивается на базе микроядра и соответствующих сервисов (серверов), реализующих прикладную среду для той или иной операционной системы.

Распределенная организация операционной системы позволяет упростить работу пользователей и программистов в сетевых средах. В распределенной ОС реализованы механизмы, которые дают возможность пользователю представлять и воспринимать сеть в виде традиционного однопроцессорного компьютера. Характерные признаки распределенной организации ОС: наличие единой справочной службы разделяемых ресурсов, единой службы времени. Для эффективного распределения программных процедур по машинам в таких ОС реализован механизм вызова удалённых процедур (RPC – Remote Procedure Call). Многонитевая обработка позволяет распараллеливать вычисления в рамках одной задачи и выполнять эту задачу сразу на нескольких компьютерах сети. Существуют другие распределённые службы.

Лекция 3. УПРАВЛЕНИЕ ЛОКАЛЬНЫМИ РЕСУРСАМИ

В рамках данной лекции мы познакомимся с принципами и алгоритмами управления локальными ресурсами ЭВМ, которые

берут на себя операционные системы. В число основных задач, решаемых операционными системами, входят:

- управление процессами (распределение процессорного ресурса);
- управление памятью;
- управление вводом/выводом;
- управление прерываниями и т.д.

Рассмотрим первые три задачи.

3.1. Управление процессами

Подсистема управления процессами непосредственно влияет на функционирование операционной системы. Здесь *процессом* называют абстракцию, определяющую и описывающую выполняемую программу. В рамках ОС процесс представляет собой единицу работы и заявку на использование системных ресурсов, например, процессорного времени. Подсистема управления процессом осуществляет планирование выполнения процессов, например создание или уничтожение процессов.

Понятия и определения в планировании процессов

Чтобы говорить об управлении процессами, необходимо ввести понятия состояний в многозадачной ОС:

1) *выполнение* – активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;

2) *ожидание* – пассивное состояние процесса, когда он не может выполняться по своим внутренним причинам, он ждет осуществления некоторого события, например, завершения операции ввода/вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса (процесс заблокирован);

3) *готовность* – пассивное состояние процесса, когда он заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

Жизненный цикл процесса характеризуется его пребыванием в одном из состояний и переходом между ними.

Теперь рассмотрим еще два понятия, которые характеризуют состояние операционной среды и ход работы процесса – контекст и дескриптор [5].

Контекст содержит информацию о состоянии регистров и программного счетчика, состоянии ввода/вывода, об ошибках и другую.

Дескриптор содержит идентификатор процесса, адрес расположения сегмента кода, степень привилегированности процесса и т.д.

Создание и планирование процессов

Чтобы активировать процесс, операционной системе необходимо в первую очередь знать его идентификатор, адрес сегмента, то есть прочитать дескриптор. Только затем ОС должна восстановить контекст процесса. Таким образом, дескриптор хранит более актуальную (оперативную) информацию о процессе по отношению к контексту.

Создание процесса происходит в три этапа:

1) создать контекст и дескриптор;

2) включить дескриптор нового процесса в очередь готовых процессов;

3) загрузить кодовый сегмент процесса в оперативную память.

Планирование процессов делится на решение следующих задач [8]:

1) определение момента времени для смены выполняемого процесса;

2) выбор процесса на выполнение из очереди готовых процессов;

3) переключение контекстов «старого» и «нового» процессов.

Алгоритмы планирования процессов делятся на две большие группы: алгоритмы, основанные на *квантовании,* и алгоритмы, основанные на *приоритетах.*

В алгоритмах, основанных на квантовании процессорного времени, смена активного процесса осуществляется в следующих случаях:

- процесс завершил свою работу и вышел из системы;
- при выполнении процесса произошла ошибка;
- процесс перешел в состояние «Ожидание»;
- исчерпан квант процессорного времени, отведённого процессу.

На рис. 4 приведена хронологическая схема выполнения процесса, который управляется алгоритмом, основанным на квантовании.

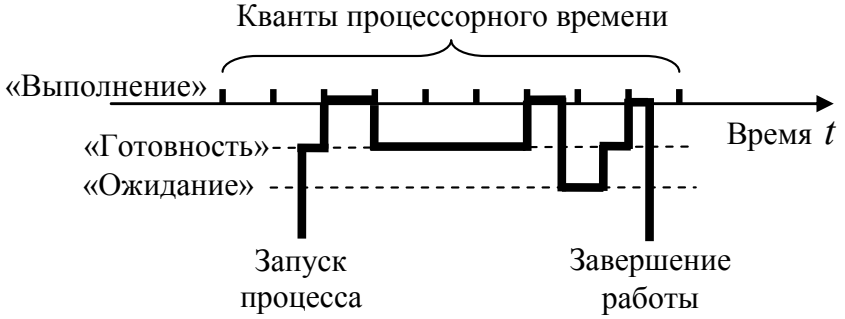

Рис. 4. Выполнение процесса, управляемого алгоритмом, основанным на квантовании

Исчерпавший квант процесс переходит в состоянии «Готовность» и ждёт нового кванта. Освободившийся ресурс ОС предоставляет другому процессу, который по определённым правилам выбирается из соответствующей очереди готовых процессов. Таким образом, процесс не занимает процессор в течение длительного времени. Отсюда алгоритмы квантования получили широкое распространение в ОС с разделением времени.

Выделяемые для процессов кванты процессорного времени могут быть как одинаковыми, так и различными в зависимости от алгоритма распределения процессорного времени. Более того, в течение жизненного цикла процесса выделяемые для него кванты могут различаться по размеру.

По-разному может быть организована очередь процессов в состоянии «Готовность»: по принципу «первый вошёл – последний вышел» (принцип стека), по принципу «первый вошёл – первый вышел» (принцип очереди).

Состояние процессов, планируемых с помощью алгоритмов, основанных на квантовании, можно представить в виде ориентированного графа, изображённого на рис. 5.

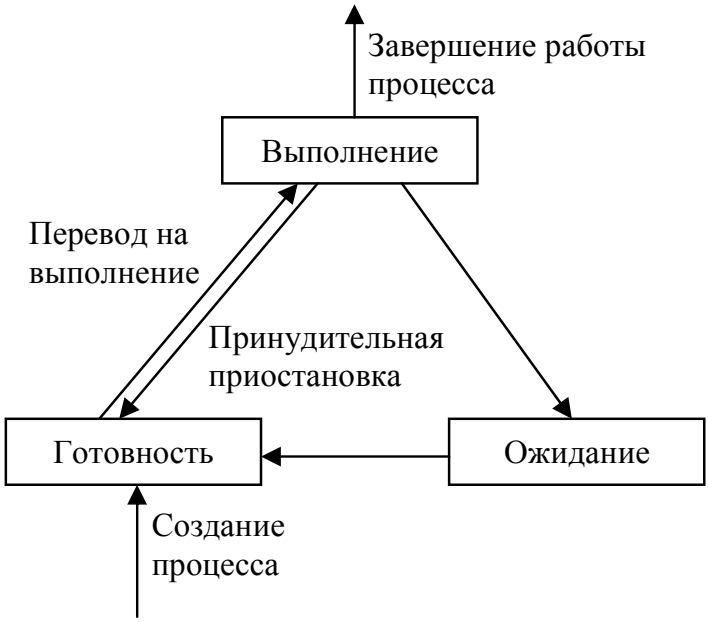

Рис. 5. Граф состояний процесса, планируемого
с помощью алгоритма, основанного на квантовании

Алгоритмы, основанные на приоритетах, делят на две разновидности:

1) использующие относительные приоритеты;

2) использующие абсолютные приоритеты.

Обе разновидности основываются на приоритетах и выбирают из очереди готовых процессов тот, у которого наивысший приоритет. Различия алгоритмов заключаются в определении момента времени смены активного процесса. В алгоритме с относительными приоритетами активный процесс выполняется до тех пор, пока он сам не покинет процессор (перейдёт в состояние «Ожидание», завершит работу или произойдёт ошибка). В алгоритме с абсолютными приоритетами выполнение активного процесса прервётся также в том случае, если в очереди появиться процесс с большим приоритетом. В этом случае прерванный процесс перейдёт в состояние «Готовность». Граф состояний процесса в ОС с алгоритмами планирования, основанных на абсолютном приоритете, в точности повторяет граф, изображенный на рис. 5. Граф состояний процесса в ОС с алгоритмами планирования, основанных на относительном приоритете, отличается одним ребром, следующим из состояния «Выполнение» в состояние «Готовность» (рис. 6).

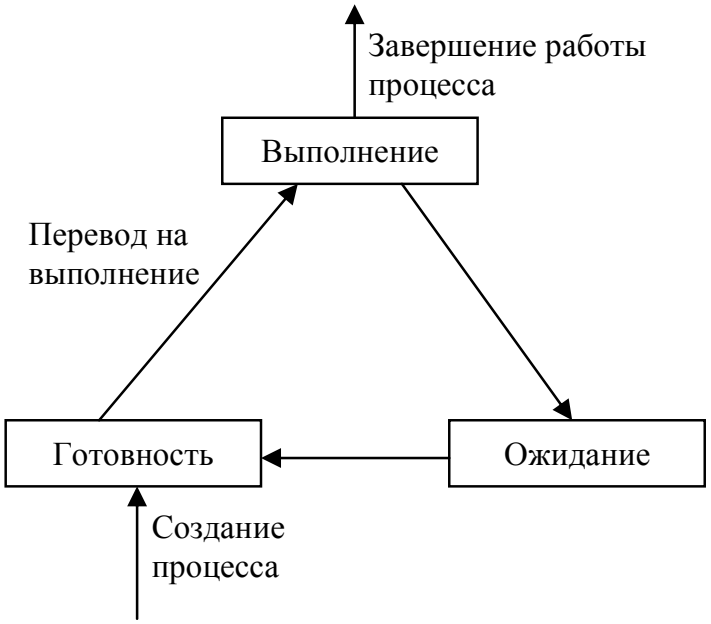

Рис. 6. Граф состояний процесса, планируемого
с помощью алгоритма, основанного на приоритетах

Вышеописанные группы алгоритмов также можно подразделить на вытесняющие и невытесняющие.

Non-preemptive multitasking – невытесняющая многозадачность – это способ планирования процессов, при котором активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление планировщику операционной системы для того, чтобы тот выбрал из очереди другой, готовый к выполнению процесс.

Preemptive multitasking – вытесняющая многозадачность – это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается планировщиком операционной системы, а не самой активной задачей.

Средства синхронизации и взаимодействия процессов

Процессам часто нужно взаимодействовать друг с другом, например, один процесс может передавать данные другому процессу, или несколько процессов могут обрабатывать данные из общего файла. Во всех этих случаях возникает проблема синхронизации процессов, которая может решаться приостановкой и акти-

визацией процессов, организацией очередей, блокированием и освобождением ресурсов.

Пренебрежение вопросами синхронизации процессов, выполняющихся в режиме мультипрограммирования, может привести к их неправильной работе или даже к краху системы. Рассмотрим, например, программу печати файлов (принт-сервер; рис. 7). Эта программа печатает по очереди все файлы, имена которых последовательно в порядке поступления записывают в специальный общедоступный файл «заказов» другие программы. Особая переменная NEXT, также доступная всем процессам-клиентам, содержит номер первой свободной для записи имени файла позиции файла «заказов». Процессы-клиенты читают эту переменную, записывают в соответствующую позицию файла «заказов» имя своего файла и наращивают значение NEXT на единицу. Особая переменная NEXT, также доступная всем процессам-клиентам, содержит номер первой свободной для записи имени файла позиции файла «заказов». Процессы-клиенты читают эту переменную, записывают в соответствующую позицию файла «заказов» имя своего файла и наращивают значение NEXT на единицу.

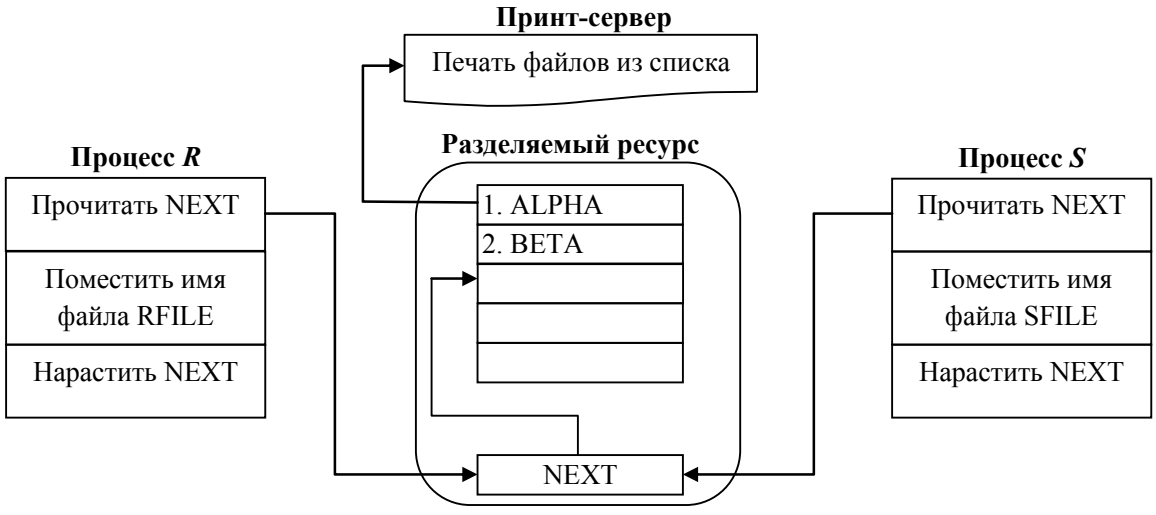

Рис. 7. Схема взаимодействия процессов *R* и *S* с разделяемым ресурсом

Предположим, что в некоторый момент процесс *R* решил распечатать свой файл, для этого он прочитал значение переменной NEXT, значение которой для определённости предположим равным 4. Процесс запомнил это значение, но поместить имя файла не

успел, так как его выполнение было прервано (например, вследствие исчерпания кванта). Очередной процесс *S*, желающий распечатать файл, прочитал то же самое значение переменной NEXT, поместил в четвертую позицию имя своего файла и нарастил значение переменной на единицу. Когда в очередной раз управление будет передано процессу *R*, то он, продолжая своё выполнение, в полном соответствии со значением текущей свободной позиции, полученным во время предыдущей итерации, запишет имя файла также в позицию 4, поверх имени файла процесса *S*.

На рис. 8 представлена диаграмма переключения между процессами.

Таким образом, процесс *S* никогда не увидит свой файл распечатанным. Сложность проблемы синхронизации состоит в нерегулярности возникающих ситуаций. В предыдущем примере можно представить и другое развитие событий: были потеряны файлы нескольких процессов или, напротив, не был потерян ни один файл. В данном случае всё определяется взаимными скоростями процессов и моментами их прерывания. Поэтому отладка взаимодействующих процессов – сложная задача. Ситуации подобные той, когда два или более процессов обрабатывают разделяемые данные и конечный результат зависит от соотношения скоростей процессов, называются гонками.

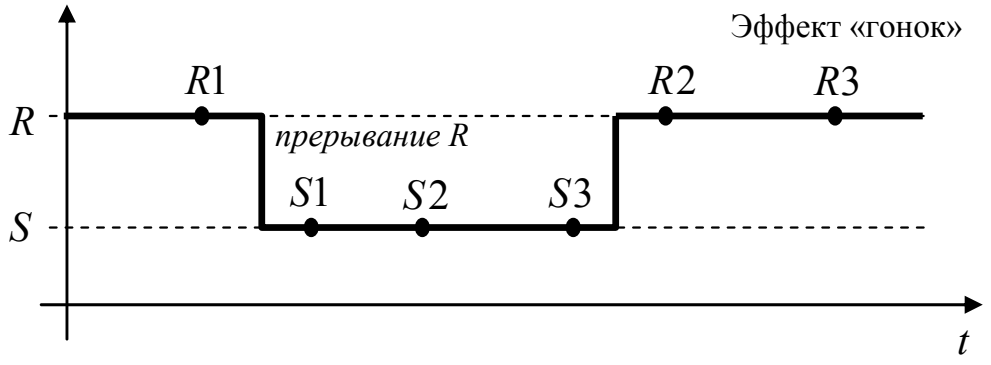

Рис. 8. Диаграмма переключения между процессами *R* и *S*

Важным понятием синхронизации процессов является понятие «*критическая секция*» программы. Критическая секция – это часть программы, в которой осуществляется доступ к разделяемым данным. Для исключения эффекта гонок по отношению к некото-

рому ресурсу необходимо обеспечить, чтобы в каждый момент в критической секции, связанной с этим ресурсом, находился максимум один процесс. Этот приём называют взаимным исключением.

Взаимное исключение реализуется следующими способами:

- запрет всех прерываний;
- метод блокирующих переменных;
- использование аппарата событий;
- использование семафоров;
- применение мониторов.

Вышеперечисленные способы подробно рассмотрены в [8].

Тупики

Приведенный на рис. 7 пример поможет нам проиллюстрировать ещё одну проблему синхронизации – *взаимные блокировки*, называемые также *дедлоками (deadlocks), клинчами (clinch)* или *тупиками*. Рассмотрим пример тупика. Пусть двум процессам, выполняющимся в режиме мультипрограммирования, для выполнения их работы нужно два ресурса, например принтер и диск. На рис. 9, *а* показаны фрагменты соответствующих программ. И пусть после того, как процесс А занял принтер (установил блокирующую переменную), он был прерван [8]. Управление получил процесс В, который сначала занял диск, но при выполнении следующей команды был заблокирован, так как принтер оказался уже занятым процессом А. Управление снова получил процесс А, который в соответствии со своей программой сделал попытку занять диск и был заблокирован: диск уже распределён процессу В. В таком положении процессы А и В могут находиться сколь угодно долго.

В зависимости от соотношения скоростей процессов, они могут либо взаимно блокировать друг друга (9, *б*), либо образовывать очереди к разделяемым ресурсам (9, *в*), либо независимо использовать разделяемые ресурсы (9, *г*). Тупиковые ситуации надо отличать от простых очередей, хотя и те и другие возникают при совместном использовании ресурсов и внешне выглядят похоже: процесс приостанавливается и ждет освобождения ресурса. Однако очередь – это нормальное явление, неотъемлемый признак высокого коэф-

фициента использования ресурсов при случайном поступлении запросов. Она возникает тогда, когда ресурс недоступен в данный момент, но через некоторое время он освобождается, и процесс продолжает своё выполнение. Тупик же, что видно из его названия, является в некотором роде неразрешимой ситуацией.

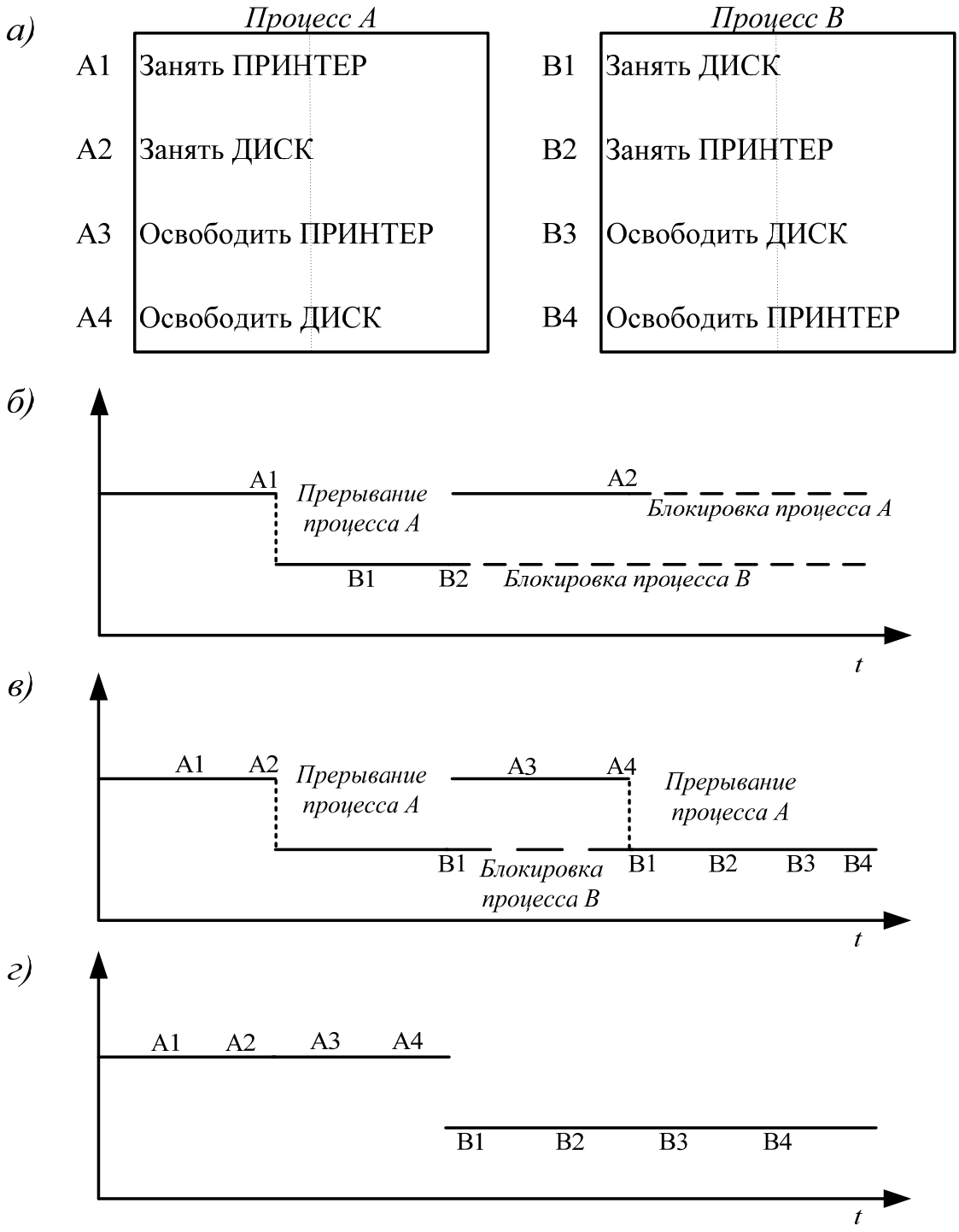

Рис. 9. Пример тупика в работе двух процессов

В рассмотренных примерах тупик был образован двумя процессами, но взаимно блокировать друг друга могут и большее число процессов.

Проблема тупиков включает в себя следующие задачи:

- предотвращение тупиков,
- распознавание тупиков,
- восстановление системы после тупиков.

Тупики могут быть предотвращены на стадии написания программ, то есть программы должны быть написаны таким образом, чтобы тупик не мог возникнуть ни при каком соотношении взаимных скоростей процессов. Так, если бы в предыдущем примере процесс А и процесс В запрашивали ресурсы в одинаковой последовательности, то тупик был бы в принципе невозможен. Другой подход к предотвращению тупиков называется динамическим и заключается в использовании определенных правил при назначении ресурсов процессам, например, ресурсы могут выделяться в определенной последовательности, общей для всех процессов.

В некоторых случаях, когда тупиковая ситуация образована многими процессами, использующими много ресурсов, распознавание тупика – нетривиальная задача. Существуют формальные, программно-реализованные методы распознавания тупиков, основанные на ведении таблиц распределения ресурсов и таблиц запросов к занятым ресурсам. Анализ этих таблиц позволяет обнаружить взаимные блокировки.

Если же тупиковая ситуация возникла, то не обязательно снимать с выполнения все заблокированные процессы. Можно снять только часть из них, при этом освобождаются ресурсы, ожидаемые остальными процессами, можно вернуть некоторые процессы в область свопинга, можно совершить «*откат*» некоторых процессов до так называемой контрольной точки, в которой запоминается вся информация, необходимая для восстановления выполнения программы с данного места. Контрольные точки расставляются в программе в местах, после которых возможно возникновение тупика.

3.2. Управление памятью

Память – важнейший ресурс, требующий тщательного управления со стороны мультипрограммной операционной системы. Рас-

пределению подлежит вся оперативная память, не занятая операционной системой. Обычно ОС располагается в самых младших адресах, однако может занимать и самые старшие адреса. Функции ОС по управлению памятью [8]:

- отслеживание свободной и занятой памяти;
- выделение памяти процессам и освобождение памяти при завершении процессов;
- вытеснение процессов из оперативной памяти на диск;
- распределение данных процессов между оперативной и внешней памятью в случае недостаточности оперативной памяти для всех процессов;
- настройка адресов программы на конкретную область физической памяти.

Типы адресов

Для идентификации переменных и команд используются символьные имена (метки), виртуальные адреса и физические адреса (рис. 10).

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

Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык. Так как во время трансляции в общем случае не известно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, обычно считая по умолчанию, что программа будет размещена, начиная с нулевого адреса. Совокупность виртуальных адресов процесса называется *виртуальным адресным пространством*. Каждый процесс имеет собственное виртуальное адресное пространство.

Физические адреса соответствуют номерам ячеек оперативной памяти, где в действительности расположены или будут расположены переменные и команды. Переход от виртуальных адресов к физическим может осуществляться двумя способами. В первом случае замену виртуальных адресов на физические делает специальная системная программа – перемещающий загрузчик. Перемещающий загрузчик на основании имеющихся у него исходных

данных о начальном адресе физической памяти, в которую предстоит загружать программу, и информации, предоставленной транслятором об адресно-зависимых константах программы, выполняет загрузку программы, совмещая её с заменой виртуальных адресов физическими.

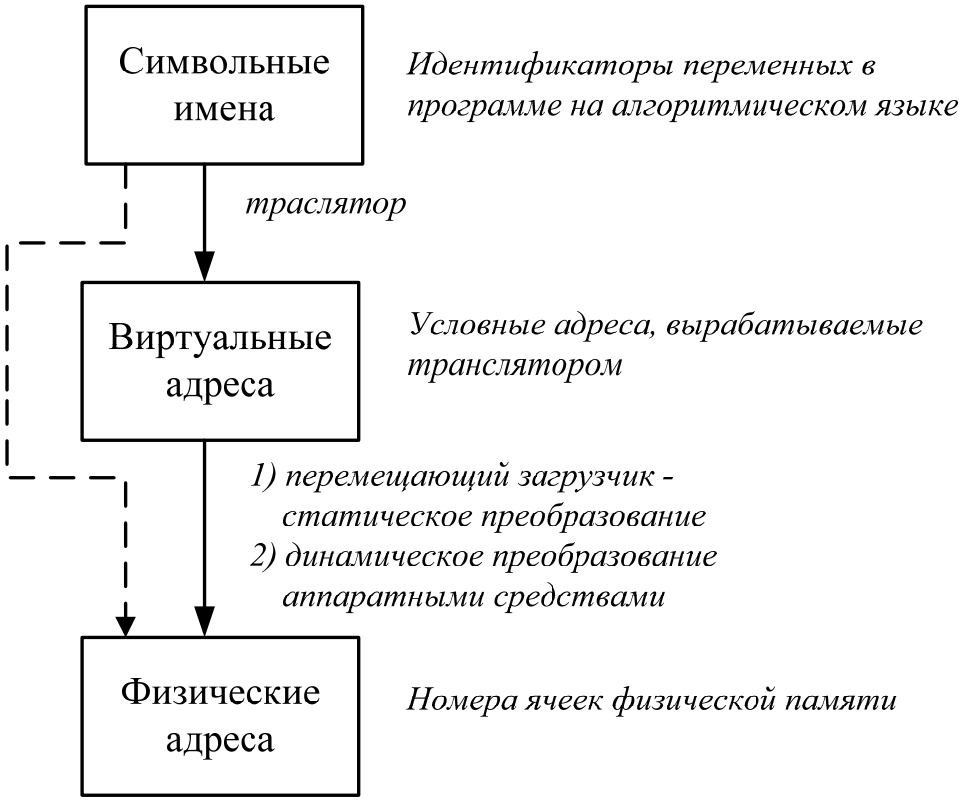

Рис. 10. Типы адресов

Второй способ заключается в том, что программа загружается в память в неизмененном виде в виртуальных адресах, при этом операционная система фиксирует смещение действительного расположения программного кода относительно виртуального адресного пространства. Во время выполнения программы при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. Второй способ более гибкий, он допускает перемещение программы во время ее выполнения, в то время как перемещающий загрузчик жёстко привязывает программу к первоначально выделенному ей участку памяти. Вместе с тем использование перемещающего загрузчика уменьшает накладные расходы, так как преобразование каждого виртуального адреса

происходит только один раз во время загрузки, а во втором случае – каждый раз при обращении по данному адресу.

В некоторых случаях (обычно в специализированных системах), когда заранее точно известно, в какой области оперативной памяти будет выполняться программа, транслятор выдаёт исполняемый код сразу в физических адресах.

Методы распределения памяти без использования дискового пространства

Данная группа методов характеризуется тем, что при распределении памяти дисковое пространства (как правило, ресурс жёсткого диска) не привлекается.

Различают следующие методы распределения памяти без использования дискового пространства [8]:

- *распределение памяти фиксированными разделами* – память заранее разбивается на стационарные разделы фиксированной длины, для запускаемого приложения отводится подходящий раздел;
- *распределение памяти разделами переменной длины* – под каждое приложение выделяется стационарный раздел требуемого размера;
- *распределение памяти перемещаемыми разделами* – под каждое приложение выделяется перемещаемый раздел требуемого размера.

Методы распределения памяти с использованием дискового пространства

Любое приложение на этапе его выполнения требует ресурсов оперативной памяти. Часто потребности приложений превышают максимально доступный объём свободной физической памяти. В первое время данную проблему «обходили» путём привлечения оверлеев, которые позволяли часть кода приложения и его данных хранить на диске, а не в ОЗУ. Существенным недостатком данного подхода является то, что вся ответственность за распределение памяти лежит на разработчике (программисте) со всеми вытекающими последствиями.

Развитие методов организации вычислительного процесса в этом направлении привело к появлению метода, известного под названием *виртуальная память*. Виртуальным называется ресурс, который пользователю или пользовательской программе представляется обладающим свойствами, которыми он в действительности не обладает. Так, например, пользователю может быть предоставлена виртуальная оперативная память, размер которой превосходит всю имеющуюся в системе реальную оперативную память. Пользователь пишет программы так, как будто в его распоряжении имеется однородная оперативная память большого объёма, но в действительности все данные, используемые программой, хранятся на одном или нескольких разнородных запоминающих устройствах, обычно на дисках, и при необходимости частями отображаются в реальную память.

Таким образом, виртуальная память – это совокупность программно-аппаратных средств, позволяющих пользователям писать программы, размер которых превосходит имеющуюся оперативную память; для этого виртуальная память решает следующие задачи:

- размещает данные в запоминающих устройствах разного типа, например, часть программы в оперативной памяти, а часть на диске;
- перемещает по мере необходимости данные между запоминающими устройствами разного типа, например, подгружает нужную часть программы с диска в оперативную память;
- преобразует виртуальные адреса в физические.

Все эти действия выполняются автоматически, без участия программиста, то есть механизм виртуальной памяти прозрачен по отношению к пользователю.

Наиболее распространёнными реализациями виртуальной памяти являются страничное, сегментное и сегментно-страничное распределение памяти, а также свопинг.

На рис. 11 показана схема *страничного распределения памяти*. Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы разме-

ра, называемые виртуальными страницами. В общем случае размер виртуального адресного пространства не является кратным размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

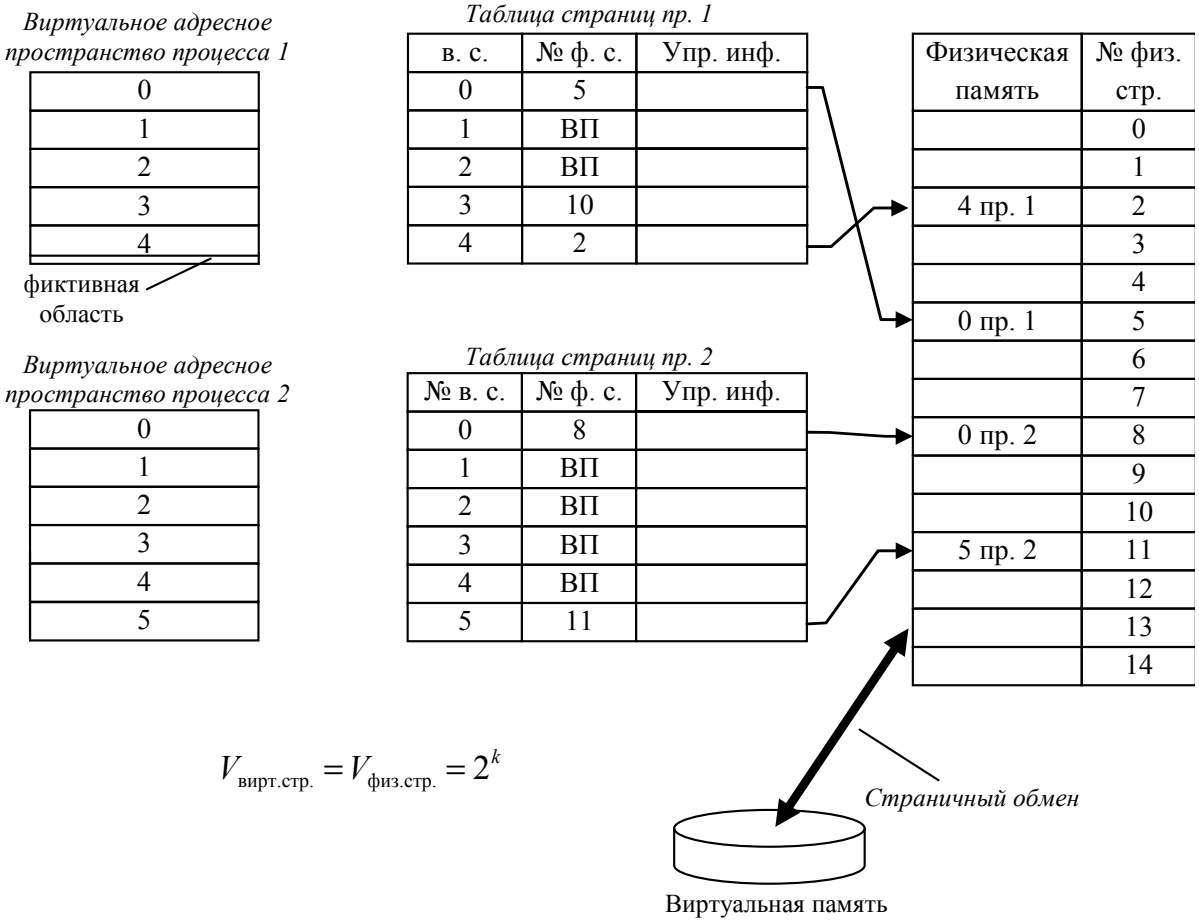

Рис. 11. Страничное распределение памяти

Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками). Размер страницы обычно выбирается, равным степени двойки: 512, 1024 и так далее, это позволяет упростить механизм преобразования адресов.

При загрузке процесса часть его виртуальных страниц помещается в оперативную память, а остальные – на диск. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. При загрузке операционная система создаёт для каждого процесса информационную структуру – таблицу страниц. В ней устанавливается соответствие между номерами вирту-

альных и физических страниц для страниц, загруженных в оперативную память, или делается отметка о том, что виртуальная страница выгружена на диск. Кроме того, в таблице страниц содержится управляющая информация, такая как признак модификации страницы, признак невыгружаемости (выгрузка некоторых страниц может быть запрещена), признак обращения к странице (используется для подсчета числа обращений за определённый период времени) и другие данные, формируемые и используемые механизмом виртуальной памяти.

При активизации очередного процесса в специальный регистр процессора загружается адрес таблицы страниц данного процесса. При каждом обращении к памяти происходит чтение из таблицы страниц информации о виртуальной странице, к которой произошло обращение. Если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический. Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое страничное прерывание. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди готовых. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается загрузить её в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то решается вопрос, какую страницу следует выгрузить из оперативной памяти.

После того, как выбрана страница, которая должна покинуть оперативную память, анализируется её признак модификации (из таблицы страниц). Если выталкиваемая страница с момента загрузки была модифицирована, то её новая версия должна быть переписана на диск. Если нет, то она может быть просто уничтожена, то есть соответствующая физическая страница объявляется свободной.

При *сегментном распределении* виртуальное пространство приложения условно разбивается на сегменты, в том числе на сегменты данных и кода. В отличие от страниц сегменты могут быть различной длины в зависимости от особенностей и принципов построения приложения. Как правило, каждый сегмент соответствует

логической структуре данных или кода, которая формируется разработчиком приложения. Ещё одно полезное свойство сегментного распределения – возможность создавать общие сегменты для двух и более одновременно выполняемых приложений.

При запуске приложения операционная система формирует таблицы сегментов, аналогичные таблицам страниц. Физическая память также разбивается на сегменты. Они могут выгружаться в виртуальную память или вновь загружаться из неё в оперативную память. Доступ к ячейкам оперативной памяти выполняется посредством пары адресов: адреса начала сегмента и адреса смещения внутри него. При 16-разрядной адресации размер сегмента не превышал 64 килобайт. При 32- и 64-разрядной адресации максимальный размер сегмента мог быть значительно выше.

Сегментно-страничное распределение объединяет свойства и подходы двух предыдущих методов. Виртуальное адресное пространство разбивается на сегменты, а сами сегменты на страницы фиксированной длины. Физическая память разделяется только на страницы той же длины, что и в виртуальном адресном пространстве приложения. Для доступа к требуемым данным используются три числа: номер сегмента, номер страницы, смещение внутри страницы.

3.3. Управление вводом/выводом

Одна из главных функций ОС – управление всеми устройствами ввода/вывода компьютера. ОС должна передавать устройствам команды, перехватывать прерывания и обрабатывать ошибки; она также должна обеспечивать интерфейс между устройствами и остальной частью системы. В целях унификации интерфейс должен быть одинаковым для всех типов устройств (независимость от устройств).

Физическая организация устройств ввода/вывода

Устройства ввода/вывода делятся на два типа: блок-ориентированные устройства и байт-ориентированные устройства. Блок-ориентированные устройства хранят информацию в блоках фиксированного размера, каждый из которых имеет свой соб-

ственный адрес. Самое распространенное блок-ориентированное устройство – диск. Байт-ориентированные устройства не адресуемы и не позволяют производить операцию поиска, они генерируют или потребляют последовательность байтов. Пример: терминалы, сканеры, сетевые адаптеры. Однако некоторые внешние устройства не относятся ни к одному классу, например часы, которые, с одной стороны, не адресуемы, а с другой стороны, не порождают потока байтов. Это устройство только выдает сигнал прерывания в некоторые моменты времени.

Внешнее устройство обычно состоит из механического и электронного компонента. Электронный компонент называется контроллером устройства или адаптером. Механический компонент представляет собственно устройство. Некоторые контроллеры могут управлять несколькими устройствами. Если интерфейс между контроллером и устройством стандартизован, то независимые производители могут выпускать совместимые как контроллеры, так и устройства.

Операционная система обычно имеет дело не с устройством, а с соответствующим контроллером. Контроллер, как правило, выполняет простые функции, например, преобразует поток бит в блоки, состоящие из байт, и осуществляет контроль и исправление ошибок. Каждый контроллер имеет несколько регистров, которые используются для взаимодействия с центральным процессором. В некоторых компьютерах эти регистры – часть физического адресного пространства. В таких компьютерах нет специальных операций ввода/вывода. В других компьютерах адреса регистров ввода/вывода, называемых часто портами, образуют собственное адресное пространство за счет введения специальных операций ввода/вывода. ОС выполняет ввод/вывод, записывая команды в регистры контроллера.

Организация программного обеспечения ввода/вывода

Основная идея организации программного обеспечения ввода/вывода состоит в разбиении его на несколько уровней. Нижние уровни обеспечивают экранирование особенностей аппаратуры от верхних, а те, в свою очередь, обеспечивают удобный интерфейс для пользователей.

Ключевым принципом является независимость от устройств. Вид программы не должен зависеть от того, читает ли она данные с гибкого диска или с жесткого диска. Очень близка к идее независимости от устройств идея единообразного именования. Иными словами, для именования устройств должны быть приняты единые правила.

Другой важный вопрос для программного обеспечения ввода/вывода – обработка ошибок. Вообще говоря, ошибки следует обрабатывать как можно ближе к аппаратуре. Если контроллер обнаруживает ошибку чтения, то он должен попытаться её скорректировать. Если же это ему не удаётся, то исправлением ошибок должен заняться драйвер устройства. Многие ошибки могут исчезать при повторных попытках выполнения операций ввода/вывода, например ошибки, вызванные наличием пылинок на головках чтения или на диске. И только если нижний уровень не может справиться с ошибкой, он сообщает об ошибке верхнему уровню.

Еще один ключевой вопрос – это использование блокирующих (синхронных) и неблокирующих (асинхронных) передач. Большинство операций физического ввода/вывода выполняется асинхронно – процессор начинает передачу и переходит на другую работу, пока не наступает прерывание. Пользовательские программы намного легче писать, если операции ввода/вывода блокирующие. Например, после команды чтения программа автоматически приостанавливается до тех пор, пока данные не попадут в буфер программы. ОС выполняет операции ввода/вывода асинхронно, но представляет их для пользовательских программ в синхронной форме.

Последняя проблема состоит в том, что одни устройства являются разделяемыми, а другие – выделенными. Диски – это разделяемые устройства, так как одновременный доступ нескольких пользователей к диску не представляет собой проблему. Принтеры – это выделенные устройства, потому что нельзя «смешивать» данные, печатаемые различными пользователями. Наличие выделенных устройств создаёт для операционной системы некоторые проблемы.

Для решения поставленных проблем целесообразно разделить программное обеспечение ввода/вывода на четыре слоя (рис. 12):

- обработка прерываний;
- драйверы устройств;
- независимый от устройств слой операционной системы;
- пользовательский слой программного обеспечения.

Рис. 12. Четырехуровневая организация ввода/вывода

Обработка прерываний скрывается как можно глубже в недрах операционной системы. Тем самым, немногое программное обеспечение сталкивается с прерываниями. Весь зависимый от устройства код помещается в *драйвер устройства*. Каждый драйвер управляет устройствами одного типа, либо одного класса. Большая часть программного обеспечения ввода/вывода независима от устройств. Оно формирует *независимый от устройств слой операци-*

онной системы. Точная граница между драйверами и независимыми от устройств программами определяется системой, так как некоторые функции, которые могли бы быть реализованы независимым способом, в действительности выполнены в виде драйверов для повышения эффективности или по другим причинам. *Пользовательский слой программного обеспечения* представляет собой набор системных библиотек, которые используются при создании пользовательских программ.

Лекция 4. ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

К прикладному программному обеспечению (ППО) относятся программы, разработанные для пользователей либо самими пользователями, для решения с помощью ЭВМ конкретной задачи или группы задач. Программы обработки заказов, ведения бухгалтерского учёта, работы с электронной почтой – только малая часть внушительного перечня видов ППО.

Классифицировать прикладное программное обеспечение можно следующим образом.

Корпоративные информационные системы [6]. Данные системы обеспечивают частичное или полное информационное сопровождение деятельности предприятия или корпорации. В число таких систем входят системы электронного документооборота и контроля над исполнением решения, системы корпоративного бухгалтерского учёта, специализированные корпоративные базы данных, инфраструктурные системы управления локальной сетью, предоставления удалённого доступа и электронной почты.

Программное обеспечение офисного назначения [7]. Такие программы (пакеты программ) обслуживают потребности индивидуальных пользователей в создании и управлении информацией. В число наиболее распространённых «офисных» программ входят: текстовый процессор (редактор), табличный процессор (электронная таблица), средства работы с презентациями, клиенты электронной почты.

Программное обеспечение для работы с удалёнными ресурсами. Данные программы позволяют получить через сеть доступ к

ресурсам различного вида: html-страницы, удалённые хранилища мультимедиа (например фотоальбомы), удалённые файловые хранилища и другим. К числу таких программ относятся интернет-обозреватели, клиенты доступа к файловым хранилищам (FTP-клиент), специализированные клиенты.

Программные средства передачи для удаленного общения. Они позволяют осуществлять передачу сообщений между пользователями, расположенными на определённой дистанции. В состав таких программных средств входят интернет пейджеры (протоколы ICQ, Google Talk, MailRu Agent и т.д.), системы интерактивных текстовых телеконференций – «чаты», средства IP-телефонии (обеспечение голосового- и видеоканала посредством интернет-протоколов), средства интерактивных аудио- и видео-конференций.

Образовательное программное обеспечение. Оно по содержанию близко к ПО для работы с медиа и для развлечений. Однако в отличие от него имеет четкие требования по тестированию знаний пользователя и отслеживанию усвоения того или иного материала. Многие образовательные программы включают функции совместного удалённого использования тех или иных ресурсов.

Имитационное программное обеспечение – используется для симуляции физических или абстрактных систем в целях научных исследований, обучения или развлечения.

Инструментальные программные средства для работы с мультимедиа содержимым. Данные программы позволяют создавать и изменять мультимедиа ресурсы различных типов: музыкальные ресурсы, изображения (включая обработанные фото), дизайнерские решения, видеоролики, комплексные мультимедиа ресурсы (например, сложные флеш-ролики).

Прикладные программы для проектирования и конструирования. Используются при разработке (моделирования или прототипирования) аппаратного и программного обеспечения. Охватывают автоматизированный дизайн (computer aided design – CAD), автоматизированное проектирование (computer aided engineering – CAE), редактирование и компилирование языков программирования, программы интегрированной среды разработки (Integrated Development Environments -IDE), интерфейсы для прикладного программирования (Application Programmer Interfaces – API).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Максимов, Н. В.* Архитектура ЭВМ и вычислительных систем / Н. В. Максимов, Т. Л. Партыка, И. И. Попов. – 3-е изд. – М. : Форум-Инфра-М, 2010. – 512 с.

2. *Глушаков, С. В.* Персональный компьютер : учеб. курс / С. В. Глушаков, А. С. Сурядный, Т. С. Хачинов. – М. : АСТ, 2008. – 480 с. – ISBN 978-5-17-048944-2.

3. *Иртегов, Д. В.* Введение в операционные системы : учеб. пособие / Д. В. Иртегов. – СПб. : БХВ-Петербург, 2008. – 1040 с. – ISBN 978-5-94157-695-1.

4. *Илюшечкин, В. М.* Операционные системы / В. М. Илюшечкин. – М. : Бином. Лаборатория знаний, 2009. – 112 с. – ISBN 978-5-94774-963-2.

5. *Молчанов, А. Ю.* Системное программное обеспечение : учеб. для вузов / А. Ю. Молчанов. – 3-е изд. – СПб. : Питер, 2010. – 400 с. – ISBN 978-5-49807-153-4.

6. *Мельников, В. П.* Информационные технологии / В. П. Мельников. – 2-е изд. – М. : Академия, 2009. – 432 с. – ISBN 978-5-76956-646-2.

7. *Уваров, С.* 500 лучших программ для вашего компьютера / С. Уваров. – СПб. : Питер, 2009. – 320 стр. – ISBN 978-5-91134-374-3.

8. *Олифер, Н. А.* Сетевые операционные системы / Н. А. Олифер, В. Г. Олифер. – М. : МГУ. Центр информационных технологий, 2009. – Режим доступа: http://www.citforum.ru/operating_systems/sos/contents.shtml

ОГЛАВЛЕНИЕ

КОНСПЕКТ ЛЕКЦИЙ ПО ДИСЦИПЛИНЕ
«СИСТЕМНОЕ И ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ»

Составители:

АБРАХИН Сергей Иванович
ДУХАНОВ Алексей Валентинович

Ответственный за выпуск – зав. кафедрой профессор С. М. Аракелян

Подписано в печать 10.02.10.
Формат 60x84/16. Усл. печ. л. 2,32. Тираж 100 экз.
Заказ
Издательство
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87