Основы современных операционных систем

1. Лекция: Понятие операционной системы (ОС), цели ее работы. Классификация компьютерных систем.

В лекции дано определение понятия "операционная система" (ОС). Дан краткий обзор функциональности и назначения ОС, краткий обзор широкого спектра видов и архитектур современных компьютерных систем (настольные, распределенные, мобильные, облачные и др.) и операционных систем для них.

Введение

Данный курс познакомит Вас с основами современных операционных систем и сетевых технологий и научит их практически использовать.

Краткое содержание курса

Данный курс посвящен основным концепциям операционных систем и сетей. Однако он не является чисто теоретическим, а дает практические навыки работы в современных ОС, рассматривает методы и приемы администрирования ОС и сетей, а также содержит лабораторные работы, помогающие студентам практически освоить рассматриваемые концепции.

Набор операционных систем, рассматриваемых в курсе, очень широк. Это прежде всего ОС семейства Windows (2000, XP, 2003, Vista, 2008, 7), в том числе – Windows для встроенных систем (Windows Embedded), Windows для **мобильных устройств** (Windows Mobile) и Windows для **облачных вычислений** (Windows Azure).

Кроме того, рассматриваются популярные ОС семейства Linux, а также особенно хорошо известная автору ОС Solaris разработки Sun / Oracle, которая была для автора основным рабочим инструментом в течение 10 лет.

Рассмотрена также популярная современная ОС для **мобильных устройств** Google Android.

Современность курса нашла свое выражение также в рассматриваемых сетевых протоколах. Рассмотрена не только классическая модель сетевых протоколов ISO / OSI, но и некоторые современные протоколы, например, Wi-Fi, GPRS, EVDO, SIMPLE/SIP.

Почему важно знать операционные системы – мнение эксперта из Microsoft

По мнению Дэвида Проберта, менеджера по разработке ОС Windows (Microsoft), знание операционных систем является основой успешной карьеры в сфере программирования. Предмет ОС сочетает в себе как математические методы, так и методы проектирования современного программного обеспечения, которые используются и во многих других современных областях – при разработке игр, клиент-серверных приложений, бизнес-приложений, Web-технологий и программных инструментов.

Знание ОС способствует становлению зрелого мышления программиста и хорошему знанию сетевых технологий и протоколов, виртуальных машин, методов современного программирования.

С этим компетентным мнением нельзя не согласиться.

Расцвет ОС в 2000-х гг

В настоящее время мы являемся свидетелями небывалого расцвета операционных систем, поэтому для их изучения сейчас для студентов открываются огромные возможности: выпускаются новые ОС для **настольных компьютеров**, **кластеров компьютеров** и параллельных вычислений, **мобильных устройств**, **облачных вычислений**.

Бесспорным лидером в данной области является корпорация Microsoft, выпустившая менее чем за 10 недавних лет целую серию ОС семейства Windows: Windows XP, Windows 2003, Windows Vista (2007), Windows 2008, Windows 2008 High-Performance Computing (HPC), Windows 7.

Развиваются также диалекты ОС Linux (Red Hat, Fedora, Mandrake, Ubuntu, SuSE и др.– сотни диалектов). Linux – операционная система типа UNIX, ядро которой свободно распространяется с исходными кодами.

Фирма Sun (в 2010 г. вошедшая в состав фирмы Oracle) разрабатывает и выпускает ОС Solaris – одну из наиболее современных ОС типа UNIX с развитой поддержкой параллельного программирования, новыми видами файловых систем, отличающуюся своей повышенной надежностью.

Это лишь некоторые ОС, которым в данном курсе будет уделено значительное внимание. Существует также много других операционных систем. В США и Канаде, как известно, весьма популярны компьютеры семейства Macintosh фирмы Apple (коротко – Mac) со своей операционной системой MacOS, являющейся законодателем мод в области графических пользовательских интерфейсов (GUI) и обмена мультимедийной информацией (например, речевого ввода). Назовем также ОС фирмы IBM для **суперкомпьютеров** и **компьютеров общего назначения (mainframes).**

Особенно важно для успешного изучения операционных систем то, что в настоящее время многие из них (или их крупные части, например, ядро) доступны с **открытым исходным кодом**.

Корпорация Microsoft положила начало этому движению в 2003 г., когда была объявлена академическая программа Windows Embedded Shared Source – был открыт исходный код Windows для встроенных систем. А в 2006 г. произошло и вовсе невероятное доселе в программистском мире событие – Microsoft открыла "святую святых", исходный код ядра ОС Windows семейства NT (NT/2000/XP/2003/2008/7) и предоставила в распоряжение университетов и академических организаций Windows Research Kernel (WRK) – самодокументированный исходный код "исследовательского" ядра Windows. Теперь каждый студент, преподаватель и исследователь имеют возможность изучать систему Windows "изнутри" и даже развивать ее, но только для целей обучения и исследований, а не для коммерции.

Фирма Sun (ныне – Oracle) положила начало аналогичной инициативе для ОС Solaris – несколько лет назад был начат проект OpenSolaris. Результаты этого академического проекта используются при выпуске новых версий коммерческой ОС Solaris.

По традиции, еще с начала 1990-х гг., ядро ОС Linux также распространяется свободно, с исходными кодами, что вызвало целую волну работ по созданию новых диалектов Linux, а также по разработке новых ОС для **мобильных устройств** на базе ядра Linux (например, ОС Google Android).

Также интенсивно развиваются **ОС для мобильных устройств**. Еще несколько лет назад наиболее используемыми ОС в этой области были ОС семейства Symbian. Однако сейчас ОС Microsoft Windows Mobile и Google Android активно теснят Symbian с рынка.

ОС для облачных вычислений – принципиально новый вид ОС, отражающий современную тенденцию к организации вычислений как **облачных (cloud computing).** Облако – это метафора Интернета. При облачных вычислениях пользователь со своего компьютера получает платный доступ через Интернет к Web-сервисам, работающим на компьютерах мощных **центров обработки данных** (например, на серверах Microsoft). При этом не только используемое программное обеспечение (в виде набора Web-сервисов), но и сами обрабатываемые данные пользователя хранятся на серверах "облачного" центра обработки данных. На своем компьютере пользователь имеет лишь простой и удобный и не требующий больших ресурсов "облачный" Web-интерфейс. Наиболее распространенной ОС для облачных вычислений является в настоящее время Microsoft Windows Azure.

Вот лишь очень краткий обзор развития операционных систем в наши дни.

По мнению автора, крупные фирмы открывают исходные коды своих операционных систем, привлекая молодых талантливых специалистов интересными проектами ОС с открытым исходным кодом, так как им необходимы молодые программисты и новые интересные идеи, которые позволят сделать ОС еще более мощными, масштабируемыми, удобными, эффективными, надежными и безопасными.

Понятие операционной системы и цели ее работы

После краткого вводного обзора перейдем к основным понятиям и их определениям. Прежде всего, дадим определение операционной системы.

Операционная система (**ОС**, в англоязычном варианте **- operating system**) – базовое системное программное обеспечение, управляющее работой компьютера и являющееся посредником (**интерфейсом**) между **аппаратурой** (**hardware**), **прикладным программным обеспечением** (**application software**) и **пользователем** компьютера (**user**). Фактически операционная система с точки зрения пользователя– это как бы продолжение аппаратуры, надстройка над ней, обеспечивающая более удобное, надежное и безопасное использование компьютеров и компьютерных сетей.

Основные **цели** работы операционной системы следующие.

1. **Обеспечение удобства, эффективности, надежности, безопасности выполнения пользовательских программ**. Для пользователя самое главное – чтобы его программа работала, вела себя предсказуемо, выдавала необходимые ему правильные результаты, не давала сбоев, не подвергалась внешним атакам. Вычислительную среду для такого выполнения программ и обеспечивает операционная система.
2. **Обеспечение удобства, эффективности, надежности, безопасности использования компьютера**. Операционная система обеспечивает максимальную полезность и эффективность использования компьютера и его ресурсов, обрабатывает прерывания, защищает компьютер от сбоев, отказов и хакерских атак. Эта деятельность ОС может быть не столь заметной для пользователя, но она осуществляется постоянно.
3. **Обеспечение удобства, эффективности, надежности, безопасности использования сетевых, дисковых и других внешних устройств, подключенных к компьютеру**. Особая функция операционной системы, без которой невозможно использовать компьютер, - это работа с внешними устройствами. Например, ОС обрабатывает любое обращение к жесткому диску, обеспечивая работу соответствующего **драйвера** (низкоуровневой программы для обмена информацией с диском) и **контроллера** (специализированного процессора, выполняющего команды ввода-вывода с диском). Любая "флэшка", вставленная в USB-слот компьютера, распознается операционной системой, получает свое логическое имя (в системе Windows – в виде буквы, например, G) и становится частью файловой системы компьютера на все время, пока она не будет извлечена (демонтирована).
4. Подчеркнем особую важность среди функций современных ОС обеспечения **безопасности, надежности и защиты данных**. Следует учитывать, что компьютер и операционная система работают в сетевом окружении, в котором постоянно возможны и фактически происходят атаки хакеров и их программ, ставящие своей целью нарушение работы компьютера, "взлом" конфиденциальных данных пользователя, хранящихся на нем, похищение логинов, паролей, использование компьютера как "робота" для рассылки реклам или вирусов и др. В связи с этим в 2002 г. фирма Microsoft объявила **инициативу по надежным и безопасным вычислениям (trustworthy computing initiative**), целью которой является повышение надежности и безопасности всего программного обеспечения, прежде всего – операционных систем. В данном курсе мы будем подробно останавливаться на том, какие действия по обеспечению надежности, безопасности и защите данных предпринимают современные ОС.

Компоненты компьютерной системы

Чтобы лучше понять место и роль операционной системы в процессе вычислений, рассмотрим компьютерную систему в целом. Она состоит из следующих компонентов:

1. **Аппаратура (hardware)** компьютера, основные части которой – **центральный процессор (Central Processor Unit - CPU**), выполняющий **команды (инструкции)** компьютера; **память (memory),** хранящая данные и программы, и **устройства ввода- вывода, или внешние устройства (input-output devices, I/O devices**), обеспечивающие ввод информации в компьютер и вывод результатов работы программ в форме, воспринимаемой

пользователем-человеком или другими программами. Часто на программистском слэнге аппаратуру называют "железом".

2. **Операционная система (operating system)** – основной предмет нашего курса; системное программное обеспечение, управляющее использованием аппаратуры компьютера различными программами и пользователями.

3. **Прикладное программное обеспечение (applications software)** – 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.

4. **Пользователи (users)** – люди и другие компьютеры. Отнесение пользователя-человека к компонентам компьютерной системы - вовсе не шутка, а реальность: любой пользователь фактически становится частью вычислительной системы в процессе своей работы на компьютере, так как должен подчиняться определенным строгим правилам, нарушение которых приведет к ошибкам или невозможности использования компьютера, и выполнять большой объем типовых рутинных действий – почти как сам компьютер. 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.

Девизом фирмы Sun Microsystems еще в 1982 г., при ее создании, стал афоризм "**The network is the computer**" (Сеть – это компьютер). Эту истину следует помнить всем пользователям компьютеров и их операционных систем и шире использовать возможности компьютерных сетей, распределяя различные функции между ее различными компьютерами (или **хостами – hosts**, как на компьютерном слэнге принято называть компьютеры в сети). Изолированный от сети компьютер ныне – это "каменный век". Отсюда – неразрывная связь операционных систем и сетей.

Общая картина функционирования компьютерной системы

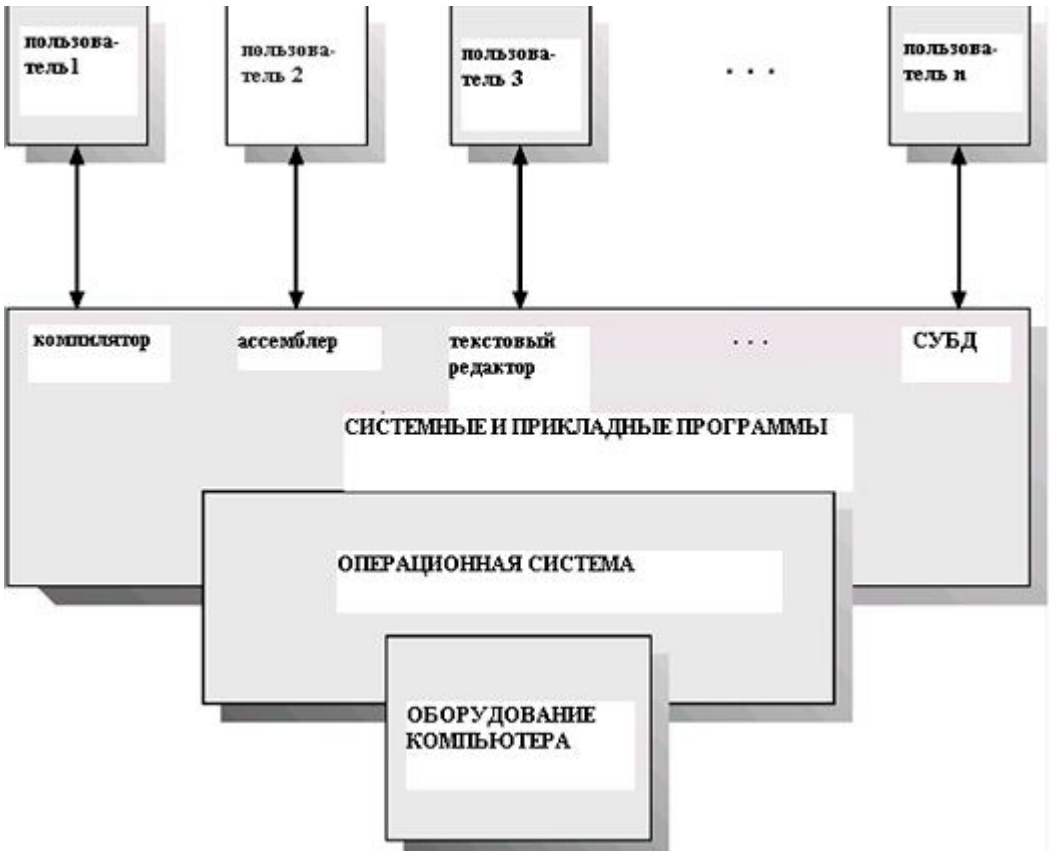

Рис. 1.1. Общая картина функционирования компьютерной системы

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.

Классификация компьютерных систем

Для того, чтобы представить себе разнообразие и масштабируемость операционных систем, рассмотрим прежде всего классификацию современных компьютерных систем, для которых разрабатываются и используются ОС – от **суперкомпьютеров** до **мобильных устройств**, - и суммируем требования к ОС для этих классов компьютеров.

Суперкомпьютеры (super-computers) – мощные многопроцессорные компьютеры, наиболее современные из которых имеют производительность до нескольких **petaflops** (10^{15} вещественных операций в секунду; аббревиатура **flops** расшифровывается как **floating-point operations per second**). Пример – суперкомпьютер "Ломоносов", установленный в МГУ. Суперкомпьютеры используются для вычислений, требующих больших вычислительных мощностей, сверхвысокой производительности и большого объема памяти. В реальной практике это прежде всего задачи моделирования – например, моделирования климата в регионе и прогнозирования на основе построенной модели погоды в данном регионе на ближайшие дни. Особенностью суперкомпьютеров является их параллельная архитектура – как правило, все они являются многопроцессорными. Соответственно, ОС для суперкомпьютеров должны поддерживать распараллеливание решения задач и синхронизацию параллельных процессов, одновременно решающих подзадачи некоторой программы.

Многоцелевые компьютеры, или **компьютеры общего назначения (mainframes)** – традиционное историческое название для компьютеров, распространенных в 1950-х – 1970-х гг., еще до эпохи всеобщего распространения персональных компьютеров. Именно для mainframe-компьютеров создавались первые ОС. Типичные примеры таких компьютеров: IBM 360/370; из отечественных – М-220, БЭСМ-6. На таких компьютерах решались все

необходимые задачи – от расчета зарплаты сотрудников в организации до расчета траекторий космических ракет. Подобный компьютер выглядел достаточно неуклюже и громоздко и мог занимать целый большой зал. Вспомните, например. огромный компьютер HAL на космическом корабле в фантастическом фильме 1960-х гг. Стэнли Кубрика "Космическая одиссея 2001 г." Но никакие фантасты не смогли предвидеть прогресса компьютерной техники XXI века – прежде всего, того, что мощный компьютер будет не занимать целую комнату, а помещаться в небольшом ящике. Параметры ранних mainframe-компьютеров были весьма скромными: быстродействие - несколько тысяч операций в секунду, оперативная память – несколько тысяч ячеек (слов). Недостаточно удобным был пользовательский интерфейс (интерактивное взаимодействие с компьютерами было реализовано гораздо позже, в 1960-х гг.). Тем не менее, на таких компьютерах решались весьма серьезные задачи оборонного и космического назначения. С появлением персональных и портативных компьютеров классические mainframe-компьютеры ушли в прошлое. Однако следует подчеркнуть, что в именно в операционных системах для mainframe-компьютеров были реализованы все основные методы и алгоритмы, рассмотренные в данном курсе, которые впоследствии были использованы в ОС для персональных, карманных компьютеров и **мобильных устройств**.

Кластеры компьютеров (computer clusters) – группы компьютеров, физически расположенные рядом и соединенные друг с другом высокоскоростными шинами и линиями связи. Кластеры компьютеров используются для высокопроизводительных параллельных вычислений. Наиболее известны в мире компьютерные кластеры, расположенные в исследовательском центре CERN (Швейцария) – том самом, где находится большой адронный коллайдер. Как правило, компьютерные кластеры располагаются в исследовательских институтах и в университетах, в том числе, например, в Петродворцовом учебно-научном комплексе СПбГУ они используются в Петродворцовом телекоммуникационном центре (ПТЦ), на нашем математико-механическом и на физическом факультетах. Операционная система для кластеров должна, помимо общих возможностей, предоставлять средства для конфигурирования кластера, управления компьютерами (процессорами), входящими в него, распараллеливания решения задач между компьютерами кластера и мониторинга кластерной компьютерной системы. Примерами таких ОС являются ОС фирмы Microsoft – Windows 2003 for clusters; Windows 2008 High-Performance Computing (HPC).

Настольные компьютеры (desktops) – это наиболее распространенные в настоящее время компьютеры, которыми пользуются дома или на работе все люди, от школьников и студентов до домашних хозяек. Такой компьютер размещается на рабочем столе и состоит из монитора, системного блока, клавиатуры и мыши. Параметры современного (2010 г.) **настольного компьютера**, наиболее приемлемые для использования современных ОС: быстродействие процессора 1 – 3 ГГц, оперативная память – 1 – 8 гигабайт и более, объем жесткого диска (hard disk drive – HDD) – 200 Гб – 1 Тб и более (1 терабайт, Тб = 1024 Гб). Все разнообразие современных операционных систем (Windows, Linux и др.) – к услугам пользователей **настольных компьютеров**. При необходимости на **настольном компьютере** можно установить две или более операционных системы, разделив его дисковую память на несколько разделов (partitions) и установив на каждый из них свою операционную систему, так что при включении компьютера пользователю предоставляется стартовое меню, из которого он выбирает нужную операционную систему для загрузки.

Портативные компьютеры (laptops, notebooks – дословно "компьютеры, помещающиеся на коленях"; "компьютеры-тетрадки") – это миниатюрные компьютеры, по своим параметрам не уступающие настольным, но по своим размерам свободно помещающиеся в небольшую сумку или рюкзак или, например, на коленях пользователя, летящего в самолете в командировку и не желающего терять времени даром. Ноутбуки стоят обычно в несколько раз дороже, чем настольные компьютеры с аналогичными характеристиками. На **ноутбуках** используются те же операционные системы, что и для настольных компьютеров (например, Windows или MacOS). Характерными чертами **портативных компьютеров** являются всевозможные встроенные порты и адаптеры для

беспроводной связи: Wi-Fi (официально IEEE 802.11) – вид радиосвязи, позволяющая работать в беспроводной сети с производительностью 10-100 мегабит в секунду (используется обычно на конференциях, в гостиницах, на вокзалах, аэропортах – т.е. в зоне радиусом в несколько сотен метров от источника приема-передачи); Bluetooth – также радиосвязь на более коротких расстояниях (10 – 100 м для Bluetooth 3.0), используемая для взаимодействия компьютера с мобильным телефоном, наушниками, плейером и др. **Внешние устройства** (дополнительные жесткие диски, принтеры, иногда даже DVD-ROM) подключаются к ноутбуку через порты USB. Еще лет 10 назад на ноутбуках активно использовались **инфракрасные порты** (**IrDA**), которые, однако, неудобны, так как требуют присутствия "ответного" IrDA – порта другого устройства на расстоянии 20-30 см от порта ноутбука, при отсутствии между ними препятствий. Другая характерная черта ноутбуков – это наличие кард-ридеров – портов для чтения всевозможных карт памяти, используемых в мобильных телефонах или цифровых фотокамерах; обеспечивается также интерфейс FireWire (официально – IEEE 1394) для подключения цифровой видеокамеры; таким образом, ноутбуки хорошо приспособлены для ввода, обработки и воспроизведения обработки мультимедийной информации. Ныне портативный компьютер имеется почти у каждого студента, что они и используют для подготовки к ответу на экзамене, либо для решения задач практикума, иногда прямо в университетском буфете. Один из критических параметров ноутбука – время работы его батарей без подзарядки; очень хорошо, если это время составляет порядка 10 часов, что пока сравнительно редко; на компьютерах, используемых автором, это время составляет не более 5 часов. Популярная разновидность ноутбука ныне – это **нетбук** - ноутбук, предназначенный для работы в сети, обычно менее мощный и поэтому более дешевый, а также более миниатюрный.

Планшетные компьютеры – Это довольно новый вид компьютера, его особенностью является то, что размеры у него, мягко говоря, маленькие, его можно спокойно носить с собой в сумочке или в руке. По сути, размер планшетного компьютера определяет величина его дисплея, поскольку она и занимает почти всю его площадь. Еще одной характерной особенностью является сенсорный дисплей. Так как, почти во всех случаях, мышку или клавиатуру к нему подключить нельзя, по причине отсутствия входов под них, то сенсорный дисплей является единственным средством управления и ввода информации. Хотя некоторые модели позволяют подключение дополнительных устройств управления, но их очень мало.

Для чего же нужен планшетный компьютер? В основном это устройство хорошо, для просмотра графических изображений, чтения книг, просмотра видео-файлов или web-страниц. Особенно оно удобно для последнего варианта, так как можно использовать беспроводной интернет, и общаться с друзьями, как с помощью сообщений, так и используя видео связь. Почти каждый планшет оснащен камерой, что позволяет осуществлять видео звонки.

Современные планшетные компьютеры имеют фактически те же порты и адаптеры, что и ноутбуки – Wi-Fi, Bluetooth, IrDA, USB (microUSB). Операционные системы для них аналогичны ОС для ноутбуков, но все же учитывают более жесткие ограничения по объему оперативной памяти. В настоящее время широко используется ОС Windows RT или – аналог Windows для мобильных устройств, Windows 8, iOS и Google Android Разумеется, для КПК имеется аппаратура и программное обеспечение для подключения к ноутбуку или настольному компьютеру с целью синхронизации данных, что обеспечивает дополнительную надежность.

Мобильные устройства (mobile intelligent devices – мобильные телефоны, коммуникаторы, смартфоны) – это устройства, которыми каждый из нас пользуется постоянно для голосовой связи, реже – для записи или обработки какой-либо информации или для выхода в Интернет. Наиболее важные параметры мобильного устройства – это по-прежнему качество голосовой связи и время автономной работы батареи. Однако все большее значение приобретают встроенные в них цифровые фото- и видеокамеры. Операционные системы для мобильных устройств отличаются большей компактностью, ввиду более жестких ограничений по памяти. ОС для смартфонов, как и для плашетов Google Android, iOS и

Microsoft Windows 8. Для мобильных устройств весьма важная характеристика ОС – это ее надежность, в частности, сохранность данных после переполнения памяти, возникающего, например, в результате приема большого числа SMS-сообщений, интенсивной фото- или видеосъемки, обеспечить сохранность данных поможет применение внешних карт памяти (выпускаются емкостью до 128 Гб)

Носимые компьютеры (wearable computers) – для повседневной жизни достаточно экзотические устройства, однако для специальных применений (например, встроенные в скафандр космонавта или в кардиостимулятор) они жизненно важны. Разумеется, их память и быстродействие значительно меньше, чем у настольных компьютеров, но критическим фактором является их сверхвысокая надежность, а для их операционных систем и прочего программного обеспечения – минимальное возможное **время ответа (response time)** – интервал, в течение которого система обрабатывает информацию от датчиков, от пользователя или из сети, превышение которого грозит катастрофическими последствиями. С этой точки зрения, ОС для **носимых компьютеров** можно отнести к **системам реального времени**.

Распределенные системы (distributed systems) – это системы, состоящие из нескольких компьютеров, объединенных в проводную или беспроводную сеть. Фактически, таковы ныне все компьютерные системы (вспомните девиз **"Сеть – это компьютер**"). Все операционные системы должны, таким образом, поддерживать распределенный режим работы, средства сетевого взаимодействие, высокоскоростную надежную передачу информации через сеть. Все эти вопросы подробно рассмотрены в данном курсе.

Системы реального времени (real-time systems) – вычислительные системы, предназначенные для управления различными техническими, военными и другими объектами в режиме реального времени. Характеризуются основным требованием к аппаратуре и программному обеспечению, в том числе к операционной системе: **недопустимость превышения времени ответа** системы, т.е. ожидаемого времени выполнения типичной операции системы. Для ОС требования реального времени накладывают весьма жесткие ограничения – например, в основном цикле работы системы недопустимы прерывания (так как они приводят к недопустимым временным затратам на их обработку). Системы реального времени – особая весьма серьезная и специфическая область, изучение которой выходит за рамки данного курса.

Приведенный обзор дает некоторое представление о разнообразии компьютерных систем в наше время. Для каждой из них должна быть разработана адекватная операционная система.

Классификация компьютерных архитектур

Компьютерные системы отличаются между собой не только по своим параметрам и своему назначению, но и по своим внутренним архитектурным принципам. Наиболее известны следующие подходы к архитектуре компьютерных систем.

CISC (Complicated Instruction Set Computers – компьютеры с усложненной системой команд) – исторически первый подход к компьютерной архитектуре, суть которого в том, что в систему команд компьютера включаются сложные по семантике операции, реализующие типовые действия, часто используемые при программировании и при реализации языков – например, вызов рекурсивных процедур и автоматическое обновление дисплей-регистров, групповые операции пересылки строк и массивов и др. Типичными представителями **CISC**-компьютеров были: из зарубежных компьютерных систем – машины серии **IBM 360/370,** из отечественных – многопроцессорные вычислительные комплексы (МВК) **"Эльбрус**". В IBM 360, например, была реализована команда MVC (move characters), которая выполняла пересылку массива символов (строки) из одной области памяти в другую, причем адреса источника, получателя и длина пересылаемой строки задавались в регистрах. В "Эльбрусе" был аппаратно реализован в общем виде вход в процедуру с передачей через стек параметров, обновлением дисплей-регистров, указывающих на доступные процедуре области локальных данных. Другой пример – в "Эльбрусе" команда считывания в стек значения по заданному адресу осуществляла автоматический проход "косвенной цепочки" заранее не

известной длины – если значение оказывалось также адресом, то происходило считывание в стек значения по нему и т.д., до тех пор, пока считанная в стек величи на не окажется значением, а не адресом. С одной стороны, понятно стремление авторов **CISC**-архитектур сделать аппаратуру как можно более "умной". С другой стороны, жесткое "вшивание" сложных алгоритмов выполнения команд в "железо" приводило к тому, что аппаратура исполняла каждый раз некоторый общий алгоритм команды, требовавший десятков или даже сотен тактов процессора, но как-либо оптимизировать выполнение этих команд с использованием конкретной информации о длине строки, косвенной цепочки и т.д. возможности не было. Другой недостаток **CISC**-архитектур в том, что подобные групповые операции на время их выполнения фактически останавливали работу **конвейера (pipeline)** - реализованной в любой компьютерной архитектуре аппаратной оптимизации, параллельного выполнения нескольких соседних команд при условии их независимости друг от друга по данным.

RISC (Reduced Instruction Set Computers – компьютеры с упрощенной системой команд) – упрощенный подход к архитектуре компьютеров, предложенный в начале 1980-х гг. профессором Дэвидом Паттерсоном (университет Беркли, США) и его студентом Дэвидом Дитцелом (впоследствии – крупным ученым, руководителем компании Transmeta). Примеры семейств **RISC**-компьютеров: **SPARC, MIPS, PA-RISC, PowerPC**. Принципы данного подхода: упрощение семантики команд, отсутствие сложных групповых операций (которые могут быть реализованы последовательностями команд, содержащими циклы); одинаковая длина команд (32 бита – архитектура была разработана в расчете на 32-битовые процессоры); выполнение арифметических операций только в регистрах и использование специальных команд считывания из памяти в регистр и записи из регистра в память; отсутствие специализированных регистров (например, дисплей-регистров для адресации доступных областей локальных данных в стеке); использование большого набора регистров (**регистрового файла**) общего назначения– 512, 1024, 2048 регистров и т.д., в зависимости от конкретной модели процессора; передача при вызове процедур параметров через регистры. Подобная архитектура дает широкий простор для оптимизаций, выполняемых компиляторами, что и демонстрируют компиляторы Sun Studio разработки фирмы Sun / Oracle для ОС Solaris и Linux. RISC-архитектура до сих пор используется при разработке новых компьютеров.

VLIW (Very Long Instruction Word – компьютеры с широким командным словом) – подход к архитектуре компьютеров, сложившийся в 1980-х – 1990-х гг. Основная идея данного подхода – **статическое планирование параллельных вычислений компилятором** на уровне отдельных последовательностей команд и подкоманд. При данной архитектуре каждая команда является "**широкой" (long) и** содержит несколько **подкоманд**, выполняемых параллельно за один машинный такт на нескольких однотипных устройствах процессора – например, в таком компьютере может быть два устройства сложения, два логических устройства, два устройства для выполнения переходов и т.д. Задачей компилятора является оптимальное планирование загрузки всех этих устройств в каждом машинном такте и генерация таких (широких) команд, которые позволили бы оптимально загрузить на каждом такте каждое из устройств. Достоинством такой архитектуры является возможность распараллеливания вычислений, недостатком – сложность (по сравнению с **RISC**-архитектурой). Примеры компьютеров таких архитектур: из зарубежных – компьютеры Cray X/MP, Cray Y/MP и др., разработанные компьютерным гением Сеймуром Креем (Cray) и его фирмой Cray Research; из отечественных – многопроцессорный вычислительный комплекс "Эльбрус-3".

EPIC (Explicit Parallelism Instruction Computers – компьютеры с явным распараллеливанием) – по архитектуре аналогичны **VLIW**, но с добавлением ряда важных усовершенствований: например, **спекулятивных** вычислений – параллельного выполнения обеих веток условной конструкции с вычислением условия. Подход сложился и используется с 1990-х гг. Примеры процессоров данной архитектуры - Intel **IA-64, AMD-64**.

Multi-core computers (многоядерные компьютеры) – получившая наиболее широкую популярность в настоящее время архитектура компьютеров, при которой каждый процессор имеет несколько **ядер (cores),** объединенных в одном кристалле и параллельно работающих на одной и той же общей памяти, что дает широкие возможности для параллельных вычислений. В настоящее время известны **многоядерные** процессоры фирмы Intel (Core 2 Duo, Dual Core и др.), а также мощные **многоядерные** процессоры фирмы Sun / Oracle: **Ultra SPARC-T1 ("Niagara")** - 16-ядерный процессор**; Ultra SPARC-T2 ("Niagara2")** – 32-ядерный процессор. Все ведущие фирмы мира заняты разработкой и выпуском все более мощных **многоядерных** процессоров. Соответственно, создатели операционных систем для таких компьютеров разрабатывают базовые библиотеки программ, позволяющие в полной мере использовать возможности параллельного выполнения на **многоядерных** процессорах.

Hybrid processor computers (компьютеры с гибридными процессорами) – новый, все шире распространяющийся подход к архитектуре компьютеров, при котором процессор имеет **гибридную** структуру – состоит из (**многоядерного**) **центрального процессора (CPU)** и (также **многоядерного**) **графического процессора (GPU – Graphical Processor Unit**). Такая архитектура была разработана, в связи с необходимостью параллельной обработки графической и мультимедийной информации, что особенно актуально для компьютерных игр, просмотре на компьютере высококачественного цифрового видео и др. Гибридная архитектура является новым "интеллектуальным вызовом" для разработчиков компиляторов, которым необходимо разработать и реализовать адекватный набор оптимизаций как для центральных, так и для графических процессоров. Примерами таких архитектур являются новые процессоры фирмы AMD, а также графические процессоры серии Tesla фирмы NVidia.

Основные компоненты операционной системы

Рассмотрим теперь основные части ОС.

Ядро (kernel) – низкоуровневая основа любой операционной системы, выполняемая аппаратурой в особом **привилегированном режиме** (подробно о нем речь в следующей лекции). Ядро загружается в память один раз и находится в памяти **резидентно** – постоянно, по одним и тем же адресам.

Подсистема управления ресурсами (resource allocator) – часть операционной системы, управляющая вычислительными ресурсами компьютера - оперативной и внешней памятью, процессором и др.

Управляющая программа (control program, supervisor) – подсистема ОС, управляющая исполнением других программ и функционированием устройств ввода-вывода.

Ключевые термины

CISC (Complicated Instruction Set Computer – компьютер с усложненной системой команд) – исторически первый подход к компьютерной архитектуре, суть которого в усложненности в системы команд вследствие реализации в них сложных по семантике операций, реализующие типовые действия, часто используемые при программировании и при реализации языков (например, групповая пересылка строк).

EPIC (Explicit Parallelism Instruction Computers – компьютеры с явным распараллеливанием команд) – подход к архитектуре компьютера, аналогичный **VLIW**, но с добавлением ряда усовершенствований, например, спекулятивных вычислений – параллельного выполнения обеих веток условной конструкции с вычислением условия.

RISC (Reduced Instruction Set Computer – компьютер с упрощенной системой команд) – упрощенный подход к архитектуре компьютеров, характеризующийся следующими принципами: упрощение семантики команд; отсутствие сложных групповых операций; одинаковая длина команд (32 или 64 бита, по размеру машинного слова); выполнение арифметических операций только в регистрах и использование специальных команд записи и считывания регистр память; отсутствие специализированных регистров; использование большого набора регистров общего назначения (регистрового файла); передача при вызове процедур параметров через регистры.

VLIW (Very Long Instruction Word – компьютеры с широким командным словом) – подход к архитектуре компьютеров, основанный на следующих принципах: статическое планирование параллельных вычислений компилятором на уровне отдельных последовательностей команд и подкоманд.; представление команды как "широкой" - содержащей несколько подкоманд, выполняемых параллельно за один и тот же машинный такт на нескольких однотипных устройствах процессора – например, двух устройствах сложения и двух логических устройствах.

Внешние устройства - см. **Устройства ввода-вывода**

Гибридный процессор – новый, все шире распространяющийся подход к архитектуре компьютеров, при котором процессор имеет гибридную структуру – состоит из (многоядерного) центрального процессора **(CPU)** и (также многоядерного) графического процессора (GPU – Graphical Processor Unit).

Инициатива по надежным и безопасным вычислениям (trustworthy computing initiative) – инициатива корпорации Microsoft (2002), целью которой является повышение надежности и безопасности программного обеспечения, прежде всего – операционных систем.

Карманный портативный компьютер (КПК, органайзер) - миниатюрный компьютер, помещающийся на ладони или в кармане, по своим параметрам почти сравнимый с ноутбуком, предназначенный для повседневного использования с целью записи, хранения и чтения информации, в том числе – мультимедийной, и коммуникации через Интернет.

Кластеры компьютеров – группы компьютеров, физически расположенные рядом и соединенные друг с другом высокоскоростными шинами и линиями связи.

Многоцелевые компьютеры (**компьютеры общего назначения, mainframes)** – традиционное историческое название для компьютеров, распространенных в 1950-х – 1970-х гг., использовавшихся для решения любых задач.

Многоядерный компьютер (multi-core computer) – наиболее распространенная в настоящее время (2010 г.) архитектура компьютеров, при которой каждый процессор имеет несколько ядер (cores), объединенных в одном кристалле и параллельно работающих на одной и той же общей памяти, что дает широкие возможности для параллельных вычислений.

Мобильное устройство (мобильный телефон, коммуникатор) – карманное устройство, предназначенное для голосовой связи, обмена короткими сообщениями, а также для чтения, записи и воспроизведения мультимедийной информации и коммуникации через Интернет.

Настольный компьютер – персональный компьютер, размещаемый на рабочем столе и используемый на работе или дома.

Носимый компьютер – сверхминиатюрный компьютер, встроенный в одежду или имплантированный в тело человека, предназначенный для обработки информации от датчиков, управления специализированными устройствами (например, кардиостимулятором), или выдачи рекомендаций по навигации и выполнению других типовых действий человеком.

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

Память – часть компьютера, хранящая данные и программы.

Подсистема управления ресурсами – компонент операционной системы, управляющая вычислительными ресурсами компьютера.

Портативный компьютер (ноутбук, лаптоп) – миниатюрный компьютер, по своим параметрам не уступающий настольному, но по своим размерам свободно помещающийся в небольшую сумку и предназначенный для использования в поездке, дома, на даче.

Прикладное программное обеспечение – программы, предназначенные для решения различных классов задач.

Распределенная система – вычислительная система, состоящая из нескольких компьютеров, объединенных в проводную или беспроводную сеть.

Система реального времени – вычислительная система, предназначенная для управления техническим, военным или другим объектом в режиме реального времени.

Суперкомпьютер – мощный многопроцессорный компьютер, производительностью до нескольких петафлопс (10^{15} вещественных операций в секунду), предназначенный для решения задач, требующих больших вычислительных мощностей, например, моделирование, прогнозирование погоды.

Управляющая программа – компонента операционной системы, управляющая исполнением других программ и функционированием устройств ввода-вывода.

Устройства ввода-вывода – устройства компьютера, обеспечивающие ввод информации в компьютер и вывод результатов работы программ в форме, воспринимаемой пользователем или другими программами

Центральный процессор – центральная часть компьютера, выполняющая его команды (инструкции)

Ядро – низкоуровневая основная компонента любой операционной системы, выполняемая аппаратурой в привилегированном режиме, загружаемая при запуске ОС и резидентно находящаяся в памяти

Краткие итоги

В настоящее время наблюдается бурное развитие операционных систем (Windows, Linux, Solaris, MacOS и др.) , в том числе – с открытым исходным кодом (Windows Research Kernel, Linux, OpenSolaris и др.).

По мнению Дэвида Проберта (менеджера Microsoft по развитию Windows), Знание ОС способствует становлению зрелого мышления программиста и хорошему знанию сетевых технологий и протоколов, виртуальных машин, методов современного программирования.

Операционная система (ОС) – общее системное программное обеспечение, являющееся интерфейсом между аппаратурой компьютера, пользователем, прикладным программным обеспечением и другими компьютерами в сети.

Цели работы ОС – обеспечение удобства, эффективности, надежности и безопасности выполнения пользовательских программ, использования компьютерного оборудования и **внешних устройств**, подключенных к компьютеру. Особая важность в настоящее время придается **инициативе по надежным и безопасным вычислениям** (trustworthy computing) фирмы Microsoft.

Компонентами компьютерной системы (в широком смысле слова) являются аппаратура, операционная система, прикладное программное обеспечение, пользователи (клиенты) – люди и другие компьютеры в сети.

Компьютерные системы очень разнообразны по своему назначению и по архитектуре.

По своему назначению и параметрам различаются суперкомпьютеры, многоцелевые компьютеры (mainframes), кластеры компьютеров, настольные компьютеры, портативные компьютеры, карманные портативные компьютеры (КПК), мобильные устройства (мобильные телефоны, коммуникаторы), носимые компьютеры, распределенные компьютерные системы, компьютерные системы реального времени.

Основные архитектуры компьютеров: CISC, RISC, VLIW, EPIC, многоядерные компьютеры, компьютеры с гибридными процессорами.

Основные компоненты операционной системы: ядро, подсистема управления ресурсами, управляющая программа.

Набор для практики

Вопросы

1. Назовите наиболее распространенные операционные системы, в том числе – с открытым исходным кодом.
2. Дайте определение операционной системы.
3. Каковы цели работы операционной системы?

4. Назовите компоненты компьютерной системы (включая программное обеспечение и пользователей).

5. Назовите основные виды компьютерных систем, различающиеся по своему назначению и параметрам.

6. Назовите основные архитектуры компьютерных систем и кратко определите, в чем суть каждой из них.

7. Каковы основные компоненты операционной системы?

Темы для курсовых работ, рефератов, эссе

1. Краткий обзор современных операционных систем (реферат).
2. Классификация современных компьютерных систем (реферат).
3. Обзор современных компьютерных архитектур (реферат).

2. Лекция: История ОС. Отечественные ОС. Диалекты UNIX.

В лекции дан исторический обзор ОС, как зарубежных, так и отечественных (ОС ДИСПАК, ОС "Эльбрус" и др.). Рассмотрены основные режимы работы пользователей и заданий в ОС (пакетный, мультипрограммирование, разделение времени).

Введение

Операционные системы имеют долгую (более 50 лет) и весьма насыщенную историю. Не следует полагать, что в России и в СССР использовались и используются лишь зарубежные ОС. Известны также выдающиеся, оригинальные отечественные работы в данной области, их мы также рассмотрим. По мере эволюции ОС были реализованы все более гибкие и удобные режимы их использования.

История ОС

В ранних mainframe-компьютерах (1940-1950 гг.), первым из которых был компьютер ENIAC (1947 г., США), операционные системы отсутствовали. Обращение к памяти в этих компьютерах осуществлялось по реальным (физическим) адресам, а обращение к внешним устройствам (например, к устройству ввода с перфокарт или накопителю на магнитной ленте) осуществлялось специальными командами, также по физическим адресам. Подобные компьютеры были весьма громоздкими, каждый из них занимал большой зал, в котором пользователи по очереди работали на компьютере, используя столь неудобный интерфейс, как инженерный пульт. Каждый пользователь перед уходом "с машины" (как тогда говорили) останавливал и "обнулял" ее нажатием кнопок на пульте и уступал место следующему пользователю, который вводил свою программу и данные с перфокарт или перфоленты, набирал ее начальный адрес тумблерами на пульте и запускал ее с помощью специальной кнопки. При любом сбое или ошибке в программе, в ситуации приходилось разбираться, изучая комбинации лампочек на пульте, воспроизводящие в двоичном виде содержимое регистров.

Разумеется, подобный способ взаимодействия с компьютером был очень неудобен. Требовалась хотя бы минимальная автоматизация. Для этого в 1950-х – 1960-х гг. – были созданы **диспетчеры (dispatchers)** - предшественники ОС, системные программы, управлявшие прохождением пакета задач, вводимых с перфокарт. Например, такой **диспетчер** (названный ДМ-222) использовался на ЭВМ М-222 в середине 1970-х гг., на котором студенты мат-меха, в том числе и автор курса, пропускали свои студенческие задания. Выглядело это следующим образом. Студент писал свою программу (или исправления к ней – так называемую "добивку") на специальных бланках и сдавал в перфорацию, затем получал перфокарты и отдавал колоду перфокарт с программой оператору машинного зала. Через несколько часов он мог рассчитывать получить результаты своей программы – колоду перфокарт обратно и распечатку результатов. В машинном зале оператор вводил очередное задание с перфокарт. Программа-**диспетчер** копировала образ введенной колоды перфокарт с заданием на **ленту ввода**, на которой хранились все образы заданий в хронологическом порядке их ввода, независимо от требуемых для них ресурсов – времени и объема памяти. **Диспетчер** осуществлял запуск заданий по очереди, по принципу **FIFO (First-In-First-Out)** – в порядке поступления. Выбирая из очереди некоторое задание, **диспетчер** размещал его в памяти и запускал. По окончании задания (или при его прерывании вследствие ошибки) на печатающее устройство выдавалась распечатка результатов. Затем управление передавалось следующему заданию. Такой режим доступа к компьютеру был, конечно, гораздо удобнее, чем работа с пульта. Недостатки его в том, что, во-первых, программы пропускались по очереди (отсутствовала одновременная обработка нескольких заданий), во-вторых, ресурсы, требуемые для выполнения задания, никак не учитывались, и программа, требующая для выполнения всего 1 мин., должна была ожидать завершения большой задачи, требовавшей для выполнения, например, пяти часов, - только потому, что последняя была раньше введена в систему.

В 1960-х – 1970-х гг. были разработаны классические операционные системы, которые все более и более усложнялись. Все более сложными становились их системы файлов и другие компоненты ОС. Наиболее известные из операционных систем этого периода: среди зарубежных - ATLAS, MULTICS, OS IBM/360, среди отечественных – ОС ДИСПАК для ЭВМ БЭСМ-6. Для классических операционных систем были характерны следующие основные возможности:

1. мультипрограммирование (multi-programming) – одновременная обработка нескольких заданий;
2. пакетная обработка (batch mode) – обработка пакета заданий, введенных с перфокарт или с терминалов, с учетом их приоритетов и требуемых ресурсов
3. разделение времени (time sharing) – параллельная работа нескольких пользователей с терминалов (телетайпов или дисплеев), управлявших прохождением своих заданий, выполнявших их ввод в текстовых редакторах, компиляцию, выполнение и отладку;
4. управление процессами – параллельное (или попеременное, если компьютер был однопроцессорным) выполнение пользовательских процессов; возможность явного запуска параллельного процесса.

Разработка каждой операционной системы для каждой новой модели компьютера требовала многих лет напряженной высококвалифицированной работы. При этом каждая ОС первоначально разрабатывалась на низкоуровневом языке – языке ассемблера. Поэтому еще в 1960-х гг. возникла идея разработки **мобильных (переносимых) ОС** – операционных систем, которые могли бы использоваться на нескольких семействах компьютеров путем переноса их кода (возможно, с небольшими изменениями) с более старых моделей на более новые. Заметим, что термин **мобильный** используется здесь в ином понимании, отличном от того, к которому мы привыкли ныне (**мобильные телефоны и операционные системы для них).**

Первая **мобильная ОС** была разработана в 1970 г. Брайаном Керниганом (B. Kernighan) и Деннисом Ритчи (D. Ritchie) в фирме AT & T и получила название UNIX. Даже в самом ее названии заложено своего рода противопоставление MULTICS (multi - много, uni – один) – последняя известна своей усложненностью. Этим названием авторы подчеркивали основную идею UNIX – **унификацию** и **упрощение** представления файлов и операций над ними (в UNIX файл – это последовательность байтов), пользовательских программ и процессов. Унифицированным, не зависимым от целевой аппаратной платформы, был также исходный код UNIX, который был полностью написан на специально разработанном новом языке Си (основными авторами Си, как и UNIX, являются Б. Керниган и Д. Ритчи). Использование языка высокого уровня для разработки UNIX было революционным шагом в истории ОС и позволило, во-первых, значительно ускорить и облегчить разработку, во-вторых – перенести UNIX на многие модели компьютеров (для которых при этом, разумеется, необходимо было разработать компилятор с языка Си). Впервые система UNIX была использована в 1970 г. на миникомпьютере PDP-10. Компьютеры фирмы PDP образца начала 1970-х гг. принято относить к классу **миникомпьютеров**. Хотя данное название с современной точки зрения не вполне правомерно: такой компьютер занимал ... два небольших шкафа, по сравнению с mainframe-компьютером образца 1960-х гг., занимавшим целый зал. Объем оперативной памяти миникомпьютеров составлял всего порядка 32 килобайт (!). Однако на них успешно работала ОС UNIX (были и другие ОС – например, RSX-11), был компилятор с языка Паскаль, была реализована удобная система файлов и программа для работы с ними, были доступны математические библиотеки программ.

В начале 1980-х годов появились персональные компьютеры. Операционные системы для них фактически повторили в своем развитии операционные системы для компьютеров общего назначения: в них были использованы аналогичные идеи и методы. Однако первые персональные компьютеры были менее мощными, чем mainframes, как по объему памяти, так и по быстродействию и разрядности **микропроцессора**. Первый распространенный микропроцессор фирмы Intel был 8-разрядным, и для него была разработана также 8-разрядная операционная система CP/M. В 1975 г. была создана фирма Microsoft, и ее первой

разработкой была 16-разрядная операционная система MS DOS для персональных компьютеров с процессорами Intel 8086 (или, коротко, x86). В командном языке MS-DOS чувствуется явное влияние UNIX, однако MS-DOS предоставляет гораздо меньшие возможности командного языка.

В начале 1980-е гг. фирма Apple выпустила персональные компьютеры Lisa и Macintosh с операционной системой MacOS. Ее характерной чертой была реализация удобного **графического пользовательского интерфейса (GUI)** в виде окон, меню, "иконок" и многих других элементов GUI, к которым мы с Вами ныне так привыкли. MacOS стала первой ОС с развитой поддержкой GUI (для сравнения, MS-DOS предоставляла возможности работы непосредственно на командном языке).

В конце 1980-х - начале 1990-х гг., под влиянием MacOS, Microsoft разработала графическую оболочку Windows над операционной системой MS-DOS. Первая версия Windows, таким образом,еще не была операционной системой; она запускалась командой **win** из командного языка MS-DOS. Однако многие современные черты GUI, характерного для Windows, ставшие "родными" для пользователей Windows, в ней уже присутствовали. Затем были выпущены Windows 3.x и Windows for Workgroups (уже операционные системы), в 1995 г. – Windows 95 (с развитыми мультимедийными возможностями, большим набором встроенных драйверов для различных устройств и поддержкой механизма Plug-and-Play подключения нового устройства без остановки компьютера) и Windows NT с развитыми сетевыми возможностями и повышенной надежностью. Именно Windows NT стала основой для последующего развития Windows. В настоящее время наиболее популярными моделями Windows являются Windows XP (поддержка которой фирмой Microsoft уже завершается – система выпущена в 2001 г.), Windows 2003 Server, Windows Vista, Windows 2008 Server и Windows 7.

В начале 1990-х гг. появилась первая версия ОС Linux (ОС типа UNIX с открытыми исходными кодами ядра), которая постепенно приобрела значительную популярность, но, главным образом, используется на серверах. Большинство клиентов (пользователей) в мире предпочитают на своих компьютерах Windows или MacOS (заметим, что, например, в США и Канаде компьютеры Macintosh более популярны, чем Windows-машины с процессорами Intel или их аналогами.

Не будем также забывать, что в ответ на такой, на первый взгляд, простой вопрос: "Какая ОС самая популярная в мире?" даже сотрудники Microsoft не отвечают "Windows". Дело в том, что наиболее популярными в мире компьютерными устройствами являются не настольные или портативные компьютеры, а более дешевые и компактные смартфоны, для которых пока первенство удерживает специализированная ОС семейства ОС Google Android и «яблочная» iOS. Так что, операционные системы семейства Windows по своей распространенности оказываются лишь на третьем месте.

Диалекты UNIX

Одним из наиболее широко используемых семейств операционных систем с 1970-х гг. является UNIX. Существуют сотни диалектов UNIX. Все они имеют ряд общих возможностей, в том числе – мощные командные языки и развитые системные библиотеки. Однако все они несколько отличаются друг от друга. Фактически большинство крупных фирм в области ИТ разработали или разрабатывают собственные диалекты UNIX. Среди них наиболее известны следующие.

- Berkeley Software Distribution (BSD), в настоящее время – FreeBSD (University of Berkeley) – один из наиболее известных диалектов UNIX, разработанный в Университете Беркли, США. В нем впервые были реализованы сетевые сокеты. Именно этот диалект был положен в основу первой версии ОС Solaris фирмы Sun (Solaris 1.x) при ее создании в 1982 г. Один из авторов данного диалекта – Билл Джой (Bill Joy), один из четверых легендарных создателей фирмы Sun.
- System V Release 4 (SVR4) – диалект UNIX, разработанный в фирме AT&T. Для него наиболее характерны расширенные возможности параллельного многопоточного

программирования (multi-threading). Данный диалект был положен в основу второй версии ОС Solaris (Solaris 2.x) фирмы Sun в начале 1990-х годов.

- Linux (RedHat, SuSE, Mandrake, Caldera, Debian, Fedora и др.) – ОС типа UNIX со свободно распространяемым с исходными кодами ядром. Первая версия Linux была разработана в начале 1990-х гг. В настоящее время диалекты Linux активно используются как **серверные ОС** (ОС, управляющие работой всевозможных серверов), а также как основа для разработки ОС для мобильных устройств.
- Solaris (Sun Microsystems, ныне – Oracle / Sun) – один из наиболее известных и развитых диалектов UNIX. Имеет удобную графическую оболочку, развитые средства параллелизма и синхронизации процессов, удобные сетевые возможности (в частности, классическую сетевую файловую систему NFS), ряд новых оригинальных файловых систем (в частности, ZFS – файловая система с большим размером файлов и возможностью криптования информации). В настоящее время распространяется ОС Solaris 10.
- IRIX (Silicon Graphics) – диалект UNIX, разработанный фирмой Silicon Graphics (SGI), США, широко известным производителем графических рабочих станций.
- HP-UX (Hewlett-Packard) – диалект UNIX, разработанный и используемый одной из крупнейших "акул" в мире ИТ – фирмой Hewlett-Packard.
- Digital UNIX (DEC) – диалект UNIX, разработанный в начале – середине 1990-х гг. фирмой Digital Equipment Corporation (DEC), впоследствии приобретенной фирмой Compaq. Первая версия UNIX, поддерживавшая 64-разрядные процессоры.

Отечественные операционные системы

При анализе истории развития области ИТ следует иметь в виду особые условия, в которых развивались эти разработки как в СССР (России), так и в США, начиная с 1950-х гг. – "холодная война" и "железный занавес". Вследствие этого, все эти разработки, как в области аппаратуры, так и в области программного обеспечения, были строго засекречены (по мнению автора, в СССР – даже более строго, чем в США, так как о работах американских специалистов мы все же имели возможность узнавать из журналов). Такая ситуация приводила к тому, что аналогичные идеи подчас возникали и реализовывались по обе стороны "железного занавеса" примерно в одно и то же время, при почти полном отсутствии информации о работах друг друга. Однако в этом были и своего рода положительные стороны: на эти работы выделялись значительные средства правительством и отраслевыми министерствами, на эти средства создавались и развивались весьма сильные команды разработчиков (прежде всего – в области аппаратуры, операционных систем и компиляторов). В конце 1980-х – начале 1990-х гг., в известный переходный период в СССР и России, когда начинали создаваться группы для работы по outsourcing – проектам, финансируемые фирмами США, американские специалисты были просто поражены, обнаружив в России сильнейшие исследовательские и промышленные группы в области ИТ, предлагающие и реализующие массу своих идей, находящиеся в курсе новейших разработок, а своим теоретическим уровнем, разработками в области структур данных и эффективных алгоритмов подчас опережавшие лучшие американские команды. Все описанные тенденции существенно повлияли на разработку аппаратуры и операционных систем. Отечественные разработчики, почти ничего не зная об аналогичных работах американских коллег, создавали свои оригинальные системы, в том числе – ОС. Например, идея многопоточности (multi-threading) была реализована в ОС "Эльбрус" еще в конце 1970-х гг., а в популярных зарубежных ОС (UNIX, Solaris, Windows NT) многопоточность появилась только в конце 1980-х – начале 1990-х гг. К сожалению, имело место и существенное отставание советских и российских ИТ-специалистов от американцев – прежде всего, в области разработки элементной базы и технологии производства компьютеров, а также в области графических пользовательских интерфейсов (GUI).

Среди передовых оригинальных отечественных разработок в области компьютерной аппаратуры и ОС 1960-х – 1970-х гг. следует выделить прежде всего ЭВМ БЭСМ-6, ее операционные системы: **ОС ДИСПАК**, ОС ДИАПАК, ОС ИПМ и ее системное и прикладное

программное обеспечение. Разработчиком БЭСМ-6, ОС ДИСПАК и ОС ДИАПАК был Институт точной механики и вычислительной техники АН СССР под руководством академика Сергея Алексеевича Лебедева, основателя всей нашей отечественной вычислительной техники. Разработчик ОС ИПМ – Институт прикладной математики АН СССР. ЭВМ БЭСМ-6 и ее программное обеспечение следует признать уникальными. В их развитии участвовали многие академические и университетские коллективы не только СССР, но и зарубежных стран - достаточно вспомнить такие системы, как АЛГОЛ-ГДР - реализацию расширения Алгола-60 с развитыми математическими библиотеками, выполненную нашими коллегами из Германии, а также реализацию Паскаля для БЭСМ-6, разработанную специалистами из Польской Академии наук. Операционные системы для БЭСМ-6 поддерживали пакетный (с учетом приоритетов и ресурсов заданий) и диалоговый режимы взаимодействия с компьютером, страничную организацию виртуальной памяти, работу с внешними устройствами и телекоммуникационными каналами, работу в локальных сетях. К каждой БЭСМ-6 были подключены десятки терминалов, работавших под управлением диалоговых систем ДИМОН, ДЖИН и др. (это при объеме оперативной памяти БЭСМ-6 всего в 32 страницы по 4096 байтов и быстродействии до 1 млн. операций в секунду). Работу БЭСМ-6 и ее ОС отличала высокая надежность. Руководитель разработки ОС ДИСПАК – В.Ф. Тюрин.

Другой передовой отечественной разработкой 1970-х – 1980-х гг. была разработка многопроцессорных вычислительных комплексов (МВК) "Эльбрус-1" и "Эльбрус-2". Идейным вдохновителем проекта "Эльбрус" стал сам С.А. Лебедев, затем им руководили академик Всеволод Сергеевич Бурцев, а после него – чл.-корр. АН СССР Борис Арташесович Бабаян. Следует признать, что у "Эльбруса" были зарубежные прототипы и задолго до его появления были написаны академические зарубежные работы, заложившие научные основы подобных компьютерных архитектур. Коммерческим прототипом "Эльбруса" была известная серия компьютеров фирмы Burroughs (США): B5000 / B5500 / B6700 / B7700. Однако разработчикам "Эльбруса" и его операционной системы удалось предложить и реализовать целый ряд собственных оригинальных идей и методов. Основными принципами "Эльбруса", как и его предшественников, являлись: **теговая архитектура** (каждое слово памяти, кроме данных, содержало **тег –** код типа данных, хранящихся в этом слове, по которому аппаратура контролировала правильность выполнения операции), **динамизм** и аппаратная поддержка типичных (подчас весьма сложных) последовательностей действий, используемых при реализации языков высокого уровня - например, вход в процедуру по указателю на нее, с установкой дисплей-регистров, ссылающихся на доступные процедуре области локальных данных. ОС "Эльбрус" поддерживала создание процессов и операции над ними, аналогичные тем, которые впоследствии в зарубежных разработках были названы **многопоточностью (multi-threading);** была реализована **математическая** (виртуальная) память с поддержкой страничного распределения виртуальной памяти (на диске) и сегментного распределения физической (оперативной) памяти. Динамизм выражался в том, что отсутствовала статическая линковка; все программы и модули загружались в память только динамически, при первом вызове. Также динамически, при первом запросе, по прерыванию, выделялся каждый массив математической памяти. Подобные принципы были для своего времени передовыми, использование тегов значительно повысило надежность. Однако, с современной точки зрения, идеологию "Эльбруса", по-видимому, нельзя считать гибкой и эффективной, так как все аппаратные операции и соответствующие действия ОС были реализованы в общем виде, и практически отсутствовала какая-либо возможность оптимизаций, например, для более быстрого вызова процедуры в случае отсутствия необходимости обращения к ее аргументам, для быстрого доступа к статической области памяти и т.д.

Были и другие интересные отечественные разработки новых архитектур компьютеров и их операционных систем, прежде всего - оригинальные специализированные компьютеры для различных применений и их операционные системы (в основном, по своему классу и назначению, они были системами реального времени).

Однако в начале 1970-х годов в развитии отечественной вычислительной техники и ее системного программного обеспечения начался новый, неожиданный для большинства пользователей и специалистов, этап. Правительство СССР приняло беспрецедентное решение о создании, в качестве основной на достаточно долгий период времени (как изначально планировалось, на 20-30 лет, что оказалось утопией), отечественной серии -**Единой Системы ЭВМ (ЕС ЭВМ)** - путем копирования американских компьютеров серии IBM 360. Соответственно, все базовое системное программное обеспечение, в том числе и ОС, также было адаптировано к использованию в СССР (либо использовалось в оригинальном виде – с сообщениями на английском языке и т.д.). Это решение вызвало большие проблемы с финансированием у разработчиков отечественных архитектур компьютеров. Это также вызвало большие сложности у пользователей и разработчиков программного обеспечения, так как далеко не все хорошо владели английским языком (ныне в этом последнем отношении ситуация гораздо лучше). Появились, например, системы-обертки, обеспечивающие русскоязычный интерфейс: с их помощью все задания для ЕС писались с использованием русскоязычной мнемоники, затем конвертировались в англоязычный Job Control Language (язык управления заданиями IBM 360), а все сообщения, выдаваемые в качестве результатов, переводились на русский язык. Это было интересным подходом, однако не прижилось. Документация по IBM 360 постепенно была переведена на русский язык, появилась русскоязычная справочная и учебная литература по ЕС ЭВМ. К сожалению, отечественные аналоги аппаратуры IBM 360 – машины серии ЕС ЭВМ – оказались гораздо менее надежными, чем их прототипы. В течение еще нескольких лет было принято еще одно правительственное решение – об аналогичном копировании американских миникомпьютеров серий PDP 10 и PDP 11, под общим названием "Система Мини-ЭВМ" (СМ ЭВМ). Были выпущены компьютеры этой серии СМ-1, СМ-2, СМ-3 и СМ-4. Были и другие аналогичные работы по копированию зарубежных архитектур компьютеров и выпуске на этой основе отечественных аналогов. Фактически, можно сказать, что, благодаря подобному подходу, срок использования зарубежных ОС в СССР и в России был продлен не менее чем на 15-20 лет, что просто беспрецедентно. Копирование машин IBM 360 и PDP, с одной стороны, дало возможность советским программистам освоить новые развитые операционные системы, языки программирования, библиотеки программ, с другой – отбросило нашу отчественную вычислительную технику еще дальше назад. Один из классиков компьютерной науки, профессор Эдсгер Дейкстра (E. Dijkstra) в 1977 г. на научном семинаре в Ленинграде в АН СССР не без иронии заметил, что "решение русских о копировании IBM-360 можно считать серьезной победой США в холодной войне".

Разумеется, история отечественных ОС на этом не закончилась. Например, в настоящее время ведется разработка отечественной свободно распространяемой операционной системы на базе Linux. Среди отечественных программистов многие являются специалистами весьма высокого уровня по операционным системам.

Набор для практики

Вопросы

1. Каким образом происходило обращение к памяти и к внешним устройствам для ранних моделей компьютеров, при отсутствии операционных систем?
2. Назовите классические операционные системы 1960-х – 1970-х гг., зарубежные и отечественные.
3. Каковы основная цель и идея разработки ОС UNIX?
4. Назовите операционные системы для 8-разрядных, 16-разрядных и современных персональных компьютеров.
5. Какая, по Вашему, операционная система является наиболее распрост раненной в мире?

6. Назовите известные Вам диалекты ОС UNIX.

7. Каковы основные возможности отечественной ОС ДИСПАК и для каких компьютеров она была разработана?

8. Какие оригинальные идеи были положены в основу системы "Эльбрус" и ее операционной системы?

9. Какие зарубежные серии компьютеров были скопированы в СССР в 1970-е гг. и под какими названиями? В чем, по-Вашему, состояли плюсы и минусы подобного подхода к развитию вычислительной техники?

10. Каковы особенности однозадачных ОС для mainframe-компьютеров с поддержкой пакетного режима?

11. Что такое монитор?

12. Как распределялась память в однозадачных ОС?

13. Что такое режим мультипрограммирования?

14. Как распределяется память в ОС с поддержкой мультипрограммирования?

15. Какие функции выполняла ОС с пакетной обработкой заданий и поддержкой мультипрограммирования?

16. Что такое режим разделения времени и каковы особенности ОС, поддерживающих этот режим?

17. Что такое откачка и подкачка заданий?

18. Какие возможности предоставлялись пользователю операционной системой для управления его заданием в режиме разделения времени?

Темы для курсовых работ, рефератов, эссе

1. История зарубежных операционных систем (реферат).
2. История отечественных операционных систем (реферат).
3. История и диалекты операционной системы UNIX (реферат).
4. Поддержка мультипрограммирования и разделения времени в операционных системах (реферат).

3. Лекция: Режимы пакетной обработки, мультипрограммирования, разделения времени

Особенности операционных систем для компьютеров общего назначения (mainframes)

Пакетный режим. Более подробное рассмотрение операционных систем начнем с особенностей ОС для mainframes.

Один из основных режимов работы ОС – **пакетный режим (batch mode)** – режим пропуска и одновременной обработки пользовательских **заданий** (**jobs**) – программ, введенных с внешнего носителя или с терминала, с учетом их приоритетов и требуемых ими ресурсов. При этом ОС пытается максимально сэкономить время пропуска пакета заданий, формируя их оптимальным образом, - например, запускать на процессоре короткое задание, пока более длинное выполняет ввод-вывод.

Уже в самых первых ОС была реализована другая основная возможность - автоматическая передача управления от одного задания к другому при завершении или прекращении предыдущего задания. Для этого ОС использует **резидентный** (постоянно находящийся в памяти по фиксированным адресам) **монитор** – программу, осуществляющую поочередную передачу управления от задания к заданию, по мере их завершения. Алгоритм работы монитора следующий. При запуске компьютера управление передается монитору, который выбирает очередное задание и передает ему управление. По окончании задания управление возвращается монитору, и т.д.

Распределение памяти в однозадачной ОС с пакетной обработкой заданий

Рис. 2.1. Распределение памяти в простой системе пакетной обработки

Оно очень простое: операционная система занимает постоянно смежную область памяти (например, по меньшим адресам), остальная область памяти отдана пользовательской программе. Такая операционная система является **однозадачной** – обрабатывает, выполняет и хранит в оперативной памяти в каждый момент времени только одно пользовательское задание (программу). По окончании текущего задания ОС загружает в освободившуюся область памяти следующее задание. Разумеется, такой режим работы недостаточно удобен и эффективен, так как при выполнении задания возможны прерывания на выполнение ввода-вывода и другие паузы, во время которых ОС могла бы дать возможность выполняться другим очередным заданиям.

ОС пакетной обработки с поддержкой мультипрограммирования

Более развитые операционные системы поддерживают режим **мультипрограммирования –** одновременной обработки и размещении в памяти сразу нескольких пользовательских заданий. Распределение памяти в такой системе изображено на рис. 2.2.

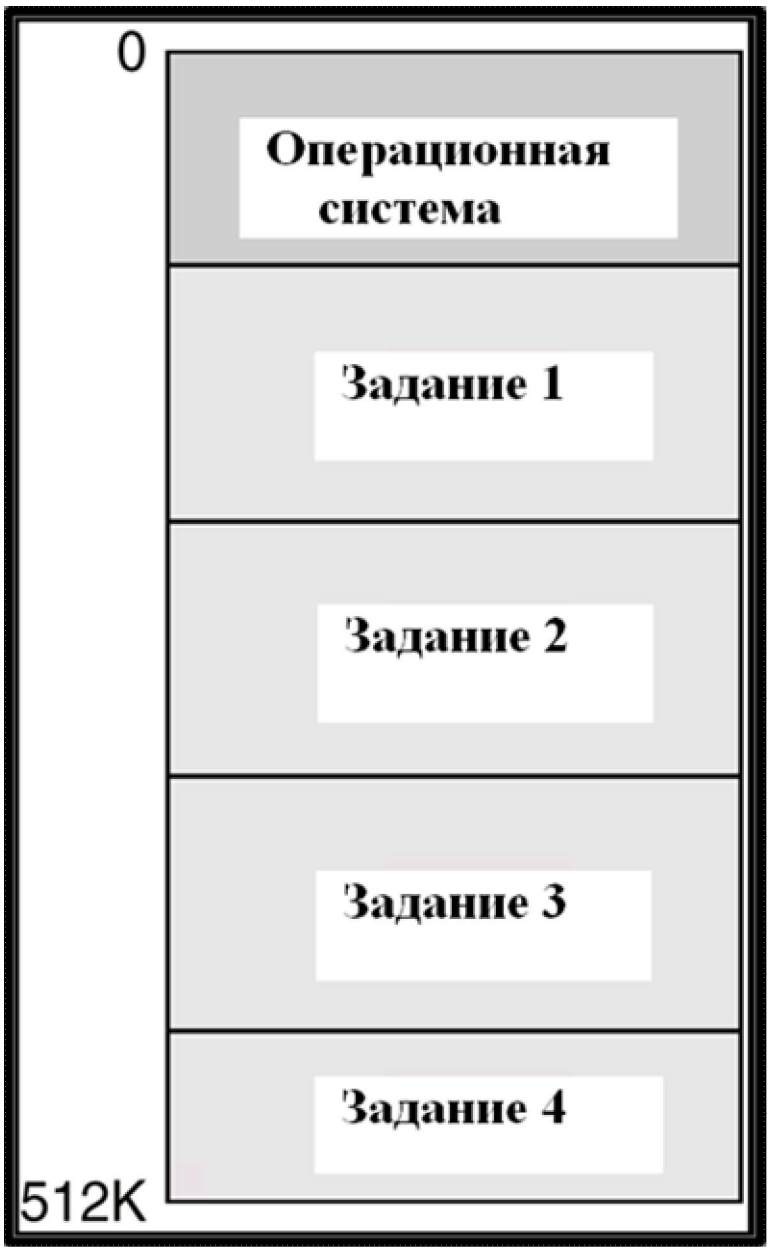

Рис. 2.2. Распределение памяти в системе пакетной обработки с поддержкой мультипрограммирования

В такой системе ОС занимает по-прежнему смежную область памяти по меньшим адресам, однако вслед за областью ОС размещаются несколько смежных областей памяти, занимаемых пользовательскими программами. Их число и размеры могут меняться.

Особенности ОС с поддержкой мультипрограммирования следующие.

Использование программ ввода-вывода, поддерживаемых операционной системой. При однозадачном режиме (см. предыдущий параграф) подобной необходимости не возникало: каждое очередное задание получало в полное распоряжение все ресурсы компьютера, в том числе – устройства ввода-вывода. При выполнении последнего процессор простаивал. В мультипрограммном режиме уже возникает потребность в реализации специальных подпрограмм для ввода-вывода, которые могли бы вызываться пользователем или операционной системой в необходимых случаях. Вызов в одной из пользовательских программ подпрограммы ввода-вывода означает для ОС возможность во время его выполнения предоставить процессор другому пользовательскому заданию.

Управление памятью. Поскольку заданий в памяти может быть несколько, причем число и размеры их областей могут меняться, перед операционной системой возникает задача

распределения памяти для пользовательских заданий – выделения памяти для загружаемого пользовательского задания и ее освобождения после завершения каждого задания. При решении этой классической задачи возникает целый ряд проблем: хранение списков свободной и занятой памяти, реализация оптимального алгоритма поиска и выделения свободной области памяти, реализация освобождения памяти, **фрагментация** - дробление свободной памяти на мелкие участки, вследствие неточного совпадения размеров свободных и требуемых участков памяти и др.

Планирование загрузки процессора (CPU scheduling) – реализация в ОС алгоритмов выбора очередного задания из набора загруженных в память заданий и выделения кванта времени центрального процессора очередному выбранному заданию. В отличие от однозадачного режима, в режиме мультипрограммирования операционная система, таким образом, в определенные моменты времени должна сделать выбор, какое из нескольких загруженных в память заданий запустить. Алгоритмы планирования и диспетчеризации процессов подробно рассмотрены ниже в данном курсе.

Управление внешними устройствами и буферизация ввода-вывода. В однозадачном режиме загруженная в память пользовательская программа для вывода на печать могла выполнить специальную машинную команду, которая выводила на устройство печати очередную строчку, что не вызывало проблем и не приводило к какой-либо путанице, вследствие монопольности "владения" компьютером очередным заданием. Однако в мультипрограммном режиме ситуация иная. Если сохранить тот же режим вывода на печать, то на печатающее устройство могут быть выведены фрагменты, принадлежащие разным заданиям, что недопустимо. Для группировки и отделения выводимой информации различных заданий друг от друга в мультипрограммной ОС используется **буферизация вывода (spooling)** – хранение для каждого задания буфера его вывода (в виде области памяти или файла), накопление в буфере выводимой заданием информации и ее вывод полностью на устройство (принтер) при завершении задания.

Режим разделения времени и особенности ОС с режимом разделения времени

Когда в составе компьютерных систем появились терминалы (вначале телетайпы, затем дисплеи), возникла необходимость реализации в ОС **режима разделения времени (time sharing)** – возможности одновременной работы пользователей со своими заданиями с терминалов, ввода заданий в систему, их запуска (при наличии свободного процессора), управления заданиями с терминала, их приостановки, отладки, визуализации на терминале их результатов. Рассмотрим особенности ОС с режимом разделения времени.

Хранение заданий в памяти или на диске. Ресурсы процессора в ОС с разделением времени распределены между несколькими заданиями, находящимися в памяти или на диске. Задание загружается в память (при наличии свободной памяти), если оно является пакетным и выбрано операционной системой для выполнения, либо если оно активируется пользователем с терминала. Процессор выделяется только тем заданиям, которые находятся в памяти.

Откачка и подкачка (swapping) - загрузка заданий с диска в память и их выгрузка из памяти на диск. В системе с разделением времени возможна ситуация, когда какое-либо задание, управляемое с терминала, неактивно (например, выполняет ввод-вывод, либо система ожидает ответа от пользователя, у которого в данный момент перерыв в работе). В этом случае ОС может принять решение о временной **выгрузке (swap out)** образа памяти задания из оперативной памяти на диск, с целью освобождения памяти для других заданий. При повторной активизации задания оно (при возможности) вновь загружается в память (**swapped in**). Подобная стратегия называется **откачкой и подкачкой.**

Поддержка диалогового взаимодействия между пользователем и системой. Когда ОС завершает исполнение пользовательской команды, она выполняет поиск следующего **управляющего оператора (control statement),** введенного с пользовательской клавиатуры.

Предоставление диалогового доступа к данным и коду пользовательской программы. В ОС с разделением времени обеспечивается возможность для пользователя ввода, запуска, редактирования, отладки своей программы с терминала, управления своим

заданием (приостановки, с последующим возобновлением), просмотра его промежуточных результатов, состояния памяти и регистров, просмотра окончательных результатов на терминале при завершении задания.

Следует учитывать, что в ОС с разделением времени обрабатываются как пакетные, так и интерактивные (диалоговые) задания, поэтому система должна обеспечивать их диспетчеризацию – переключение в нужный момент с диалогового задания на пакетное, либо с одного диалогового (пакетного) задания на другое.

Режим разделения времени, наряду с пакетным режимом, был основным в операционных системах 1960-х – 1970х- гг.

Ключевые термины

FIFO (First-In-First-Out) – режим обслуживания некоторой очереди (например, очереди введенных заданий) в порядке их поступления.

UNIX - первая **мобильная ОС** для миникомпьютеров, разработанная в 1970 г. Б. Керниганом и Д. Ритчи на новом языке программирования Си.

Буферизация вывода (spooling) – хранение для каждого задания буфера его вывода (в виде области памяти или файла), накопление в буфере выводимой заданием информации и ее вывод полностью на устройство (принтер) при завершении задания.

ДИСПАК – отечественная операционная система для ЭВМ БЭСМ-6.

Диспетчер (dispatcher) – ранняя упрощенная версия операционной системы , - системная программа, управляющая прохождением пакета вводимых заданий.

Единая система ЭВМ (ЕС ЭВМ) – семейство отечественных mainframe-компьютеров 1970-х – 1980-х годов, разработанных путем копирования американских компьютеров серии IBM 360.

Задание (job) – пользовательская программа, введенная в систему с внешнего носителя или с терминала.

Мобильная (переносимая) ОС – операционная система, использовуемая на нескольких семействах компьютеров путем переноса ее кода (возможно, с небольшими изменениями).

Монитор – упрощенный вариант операционной системы; программа, осуществляющая поочередную обработку пользовательских заданий, с последовательной передачей управления от задания к заданию, по мере их завершения.

Мультипрограммирование (multi-programming) – одновременная обработка операционной системой нескольких пользовательских заданий.

Однозадачная операционная система – ОС, обрабатывающая, выполняющая и хранящая в оперативной памяти в каждый момент времени только одно пользовательское задание (программу).

Откачка и подкачка заданий (swapping) – загрузка задания с диска в оперативную память при его активизации и его выгрузка из памяти на диск при неактивности задания; выполняется в режиме разделения времени.

Пакетная обработка (batch mode) – обработка пакета заданий, введенных пользователями, с учетом их приоритетов и требуемых ими ресурсов.

Планирование загрузки процессора (CPU scheduling) – реализация в ОС алгоритмов выбора очередного задания их набора загруженных в память заданий и выделения кванта времени центрального процессора очередному выбранному заданию.

Разделение времени (time sharing) – поддержка операционной системой одновременной работы в системе нескольких пользователей с терминалов, управление прохождением своих заданий, выполнение их ввода, редактирования, компиляции, выполнения, отладки, визуализации результатов.

Распределения памяти для пользовательских заданий – выделение памяти операционной системой для загружаемого пользовательского задания и ее освобождение после завершения каждого задания.

Резидентная программа - программа, постоянно находящаяся в оперативной памяти по фиксированным адресам.

Система мини-ЭВМ (СМ ЭВМ) - семейство отечественных миникомпьютеров 1970-х – 1980-х годов, разработанных путем копирования американских компьютеров серии PDP 10 – PDP 11.

Тег – числовой код типа данных, хранящихся в рассматриваемом слове памяти, по которому аппаратура контролирует правильность выполнения операции над данными.

Управление процессами – параллельное (или поочередное - на однопроцессорном компьютере) выполнение пользовательских процессов; возможность явного запуска параллельных процессов, управления ими и их синхронизации.

Фрагментация памяти - дробление свободной памяти на мелкие несмежные участки, вследствие неточного совпадения размеров свободных и требуемых при запросах к ОС участков памяти.

"Эльбрус" - семейство отечественных многопроцессорных суперкомпьютеров (Эльбрус-1, Эльбрус-2) 1970-х – 1980-х годов, архитектура которого основана на использовании тегов, принципах динамизма и аппаратной поддержке механизмов реализации языков высокого уровня; в операционной системе впервые был реализован аналог многопоточных вычислений (multi-threading), а также были поддержаны виртуальная память, пакетный режим, режим разделения времени, динамическое выделение памяти по запросу, динамическая линковка и загрузка выполняемых программ при первом вызове.

Краткие итоги

В ранних mainframe-компьютерах операционные системы отсутствовали. Обращение к памяти осуществлялось по конкретным физическим адресам, обращение к внешним устройствам – специальными командам, также по физическим адресам.

В 1960-х гг. были разработаны диспетчеры – упрощенные варианты ОС, осуществлявшие поочередный пропуск пользовательских заданий.

Для классических ОС 1960-х – 1970-х гг. (ATLAS, MULTICS, OS IBM 360) были характерны поддержка мультипрограммирования, пакетного режима, режима разделения времени, управление процессами.

Первой мобильной ОС, использованной на нескольких аппаратных платформах, стала система UNIX, первая версия которой разработана в 1970 г.

Первые ОС для персональных компьютеров (1980-е гг.) – CP/M (для 8-разрядных процессоров) и MS-DOS (для 16-разрядных процессоров).

Операционная система MacOS фирмы Apple характеризуется удобным графическим пользовательским интерфейсом.

С начала 1990-х гг. до настоящего времени имеет место эволюция Windows от графической оболочки к MS-DOS до наиболее популярной ОС для настольных и портативных компьютеров (Windows 7, Windows 2008 и др.). Также популярна ОС Linux (как серверная ОС).

Наиболее распространены в мире операционные системы для мобильных устройств, ввиду широкой распространенности последних. Это прежде всего ОС Symbian. Windows в этом отношении на втором месте.

Наиболее распространенные диалекты ОС UNIX: Berkeley Software Distribution (BSD), в настоящее время – FreeBSD (University of Berkeley); System V Release 4 (SVR4) – фирмы AT&T; Linux (RedHat, SuSE, Mandrake, Caldera, Debian, Fedora и другие диалекты); Solaris (Oracle / Sun); IRIX (Silicon Graphics); HP-UX (Hewlett-Packard); Digital UNIX (Digital / Compaq).

Из отечественных ОС следует отметить ОС ДИСПАК для БЭСМ-6 и ОС "Эльбрус" для МВК "Эльбрус", отличавшиеся оригинальными идеями и методами.

В 1970-х гг. в СССР было принято правительственное решение о копировании зарубежных компьютеров серии IBM 360, а затем – миникомпьютеров серий PDP-10 и PDP-

11, которое на долгие годы предопределило развитие отечественной вычислительной техники и на 15-20 лет продлило срок использования их операционных систем.

Первые операционные системы для mainframe-компьютеров поддерживали обработку пакетов заданий в однозадачном режиме. Затем в ОС появилась поддержка мультипрограммирования и разделения времени, что привело к необходимости реализации в ОС распределения памяти для пользовательских заданий, диспетчеризации процессора и буферизации ввода-вывода.

Особенности ОС с поддержкой режима разделения времени: хранение заданий в памяти либо на диске, с их откачкой и подкачкой (swapping) по мере необходимости; поддержка интерактивного взаимодействия между пользователями и ОС; поддержка диалогового доступа к коду и данным пользователей.

Набор для практики

Вопросы

19. Каким образом происходило обращение к памяти и к внешним устройствам для ранних моделей компьютеров, при отсутствии операционных систем?
20. Назовите классические операционные системы 1960-х – 1970-х гг., зарубежные и отечественные.
21. Каковы основная цель и идея разработки ОС UNIX?
22. Назовите операционные системы для 8-разрядных, 16-разрядных и современных персональных компьютеров.
23. Какая, по Вашему, операционная система является наиболее распрост раненной в мире?
24. Назовите известные Вам диалекты ОС UNIX.
25. Каковы основные возможности отечественной ОС ДИСПАК и для каких компьютеров она была разработана?
26. Какие оригинальные идеи были положены в основу системы "Эльбрус" и ее операционной системы?
27. Какие зарубежные серии компьютеров были скопированы в СССР в 1970-е гг. и под какими названиями? В чем, по-Вашему, состояли плюсы и минусы подобного подхода к развитию вычислительной техники?
28. Каковы особенности однозадачных ОС для mainframe-компьютеров с поддержкой пакетного режима?
29. Что такое монитор?
30. Как распределялась память в однозадачных ОС?
31. Что такое режим мультипрограммирования?
32. Как распределяется память в ОС с поддержкой мультипрограммирования?
33. Какие функции выполняла ОС с пакетной обработкой заданий и поддержкой мультипрограммирования?
34. Что такое режим разделения времени и каковы особенности ОС, поддерживающих этот режим?
35. Что такое откачка и подкачка заданий?
36. Какие возможности предоставлялись пользователю операционной системой для управления его заданием в режиме разделения времени?

Темы для курсовых работ, рефератов, эссе

5. История зарубежных операционных систем (реферат).
6. История отечественных операционных систем (реферат).
7. История и диалекты операционной системы UNIX (реферат).
8. Поддержка мультипрограммирования и разделения времени в операционных системах (реферат).

4. Лекция: Особенности ОС для различных классов компьютерных систем. ОС реального времени. ОС для облачных вычислений.

В лекции дан обзор особенностей ОС для различных классов вычислительных устройств (многопроцессорные и распределенные системы, настольные, карманные, мобильные и др.). Рассмотрены ОС реального времени, ОС для облачных вычислений. Проанализирована специфика требований к ОС и архитектур ОС для рассмотренных классов устройств.

Введение

Чтобы лучше понять, каковы особенности ОС для различных классов компьютерных систем и устройств, рассмотрим несколько более подробно особенности этих устройств и обсудим соответствующую специфику их операционных систем.

Особенности ОС для персональных компьютеров

Что же изменилось в самих компьютерах и их операционных системах с появлением персональных компьютеров – настольных и портативных, которые ныне являются самыми распространенными компьютерными системами?

Персональные компьютеры предназначены, как правило, для одного пользователя. Тем не менее, ОС для персональных компьютеров должна предусматривать режим мультипрограммирования (многозадачности), так как пользователям подчас удобнее выполнять несколько заданий параллельно – например, набирать некоторый текст в редакторе, принимать электронную почту и одновременно печатать на принтере какие-либо документы. Кроме того, при работе в локальной сети возможен удаленный вход на компьютер других пользователей. То есть, ОС для персональных компьютеров должна поддерживать также режим разделения времени.

Персональные компьютеры имеют разнообразный набор устройств ввода-вывода, работу с которыми должна поддерживать операционная система с помощью **драйверов –** низкоуровневых системных программ для управления этими устройствами. Для пользователя удобнее всего, если все необходимые драйверы встроены в операционную систему. Однако ситуация осложняется тем что драйверы устройств разрабатывает обычно фирма-разработчик соответствующего устройства - в англоязычной терминологии, **Original Equipment Manufacturer (OEM),** а не фирма-разработчик ОС. Поэтому при выпуске и установке на компьютер новой ОС могут возникнуть проблемы с драйверами – какое-либо устройство новая ОС "не понимает". На практике, должно пройти не менее двух-трех лет эксплуатации новой ОС, прежде чем для нее появятся драйверы для всех используемых внешних устройств, хотя в последнее время в этом отношении ситуация значительно улучшилась – новые ОС становятся все более "понятливыми" и имеют в своем составе огромные наборы драйверов.

Персональный компьютер имеет традиционные клавиатуру и мышь, обычно подключаемые через USB-порт, либо беспроводные клавиатуру и мышь, блок управления которых также подключается через USB-порт. Портативный компьютер может иметь также встроенный манипулятор типа trackball (шарик для перемещения курсора мыши) или touchpad (плоская пластинка для этой же цели). К компьютеру подключен монитор: для настольного компьютера – к порту VGA, для портативного – монитор встроен в компьютерную систему, но дополнительно может подключаться через порт VGA внешний монитор или мультимедийный проектор. К традиционным дополнительным внешним устройствам относится также принтер (подключается через порт USB, более старые модели – через так называемый **параллельный порт,** или **LPT – аббревиатура от Line PrinTer**. Реже используется **сканер** – устройство для оцифровки бумажных изображений, например, подписанных или рукописных документов. Сканер может также подключаться через порт USB, однако некоторые модели сканеров подключаются через другой интерфейс – **SCSI**, используемый и для жестких дисков (название произносится "скАзи"; о нем – чуть позже). Имеется внутренний жесткий диск (hard drive) емкостью 250 GB – 1 TB и более,

подключаемый через интерфейс IDE (более старый) или SATA. Могут подключаться через порт USB также внешние накопители - **flash-память**, или "флэшки", имеющие миниатюрный размер и объем памяти до 128 гигабайт и более; **ZIV drives** и другие разновидности **внешних жестких дисков**, имеющие в настоящее время емкость до 1 терабайта. Операционная система должна обеспечивать их использование как части компьютерной системы (например, на внешний ZIV-диск может быть даже установлено программное обеспечение, в том числе - другая операционная система). Для настольного компьютера в комплект входит устройство чтения и записи компакт-дисков в различных форматах - CD-ROM, CD-RW (с возможностью записи на CD); DVD-ROM/DVD-RW; DVD-RAM (последнее означает устройство с режимом непосредственной записи на компакт-диск, как в память); BluRay – более современный формат компакт-дисков емкостью до 25 или 50 GB и др. Для ноутбука DVD-ROM, из соображений экономии веса и размеров, может отсутствовать и должен подключаться, по шутливому выражению автора, "на веревочке" – через USB-порт. Весьма важным внешним устройством, особенно для портативного компьютера, является порт для подключения цифровой видеокамеры (IEEE 1394, или FireWire), более миниатюрный, чем USB. Он имеет дуплексный режим работы, так что, например, перемотка видеоленты на видеокамере может запускаться программным путем с компьютера. Об адаптерах для беспроводной связи – Wi-Fi, Bluetooth, IrDA – мы уже говорили.

Наиболее важными свойствами ОС для персонального компьютера должны быть, конечно, простота и удобство в использовании, дружественность к пользователю. Это достигается прежде всего, удобным и современным аппаратным и программным пользовательским интерфейсом, например, интерфейсом типа multi-touch (с доступом непосредственно к экрану), ноутбуками типа Tablet PC (с возможностью поворота экрана и ввода информации прикосновением к экрану).

При разработке ОС для ПК используются те же технологии, которые применяются и в "больших" ОС (для mainframe-компьютеров). Однако, поскольку пользователь имеет персональный доступ к компьютеру, он часто не нуждается в каких-либо системных программах для оптимизации работы процессора или в улучшенных средствах защиты (последней, однако, не следует пренебрегать и отключать ее, так как на компьютер возможны сетевые атаки).

На одном и том же персональном компьютере могут быть установлены, при необходимости, две или более операционных системы - такой компьютер носит название **double bootable system**, и при его включении пользователю выдается начальное меню для уточнения, какую именно ОС требуется запустить – **boot loader** (загрузчик ОС). Такое использование компьютера рекомендуется, например, для студентов, изучающих ОС и желающих попробовать новую операционную систему, либо изучить другую уже известную, на которую до сих пор не хватало времени, - например, установить на одном компьютере Windows и Linux. Для установки второй ОС необходимо воспользоваться специальной утилитой (например, **Partition Magic**) для выделения на диске для инсталляции новой ОС отдельного **раздела (partition)** – смежной области дисковой памяти, имеющей определенное обозначение, чаще всего – в виде латинской буквы.

Персональные компьютеры имеют **сетевые адаптеры (сетевые карты) –** устройства для подключения к локальной сети. Соответственно, ОС для персональных компьютеров имеют в своем составе драйверы сетевых адаптеров и пользовательский интерфейс для настройки подключения компьютера к локальной сети (об этом подробнее – в специальном разделе данного курса, посвященном сетям).

Параллельные компьютерные системы и особенности их ОС.

Параллельные компьютерные системы – это мультипроцессорные системы с несколькими непосредственно взаимодействующими процессорами. Классические примеры: из зарубежных компьютеров - CRAY, из отечественных – "Эльбрус"; из более современных – компьютеры серии СКИФ. В настоящее время выпускаются мультипроцессорные рабочие станции - например, купив или получив в подарок настольный компьютер, Вы можете

обнаружить в его составе два или даже четыре процессора. Соответственно, ОС должна обеспечивать реконфигурацию такой системы, подключение новых процессоров или удаление процессоров из системы, распараллеливание решения задачи на нескольких процессорах и синхронизацию решающих ее параллельных процессов.

Среди параллельных компьютеров выделяются **тесно связанные (tightly coupled) системы**, в которых процессоры разделяют общую память и таймер (такты); взаимодействие между ними происходит через общую память.

Многоядерные (multi-core) компьютеры – компьютерные системы, основанные на тесно связанных друг с другом процессорах (**ядрах**), находящихся в одном кристалле, разделяющих ассоциативную память (кэш) второго уровня и работающих на общей памяти.

Преимущества параллельной компьютерной системы:

1. **Улучшенная производительность** (**throughput**) – очевидно, что распараллеливание алгоритма решения задачи может позволить уменьшить суммарное время ее решения;
2. **Экономичность** – в параллельной системе ОС может поручить часть работы другому процессору или ядру;
3. **Повышенная надежность** – при сбое или отказе одного из процессоров ОС может переключить вычисления на другой процессор;
4. **"Дружественное" к пользователю снижение производительности (graceful degradation)** – если один из процессоров отказал и выведен из конфигурации, пользователь, при правильной организации компьютера и ОС, может даже не почувствовать замедления вычислений
5. **Устойчивость к ошибкам (fail-soft system)** – стабильная работа многопроцессорной системы при ошибке в аппаратуре или в программе.

Симметричные и асимметричные мультипроцессорные системы

Симметричная мультипроцессорная система - symmetric multiprocessing (SMP) – это многопроцессорная компьютерная система, все процессоры которой равноправны и используют одну и ту же копию ОС. Операционная система при этом может выполняться на **любом** процессоре. В такой системе любому свободному процессору может быть поручено любое задание. Все процессоры используют общую память и общие дисковые ресурсы. Несколько процессов (или потоков) могут исполняться одновременно без существенного нарушения производительности. Большинство современных ОС поддерживают архитектуру SMP. После инсталляции ОС (например, Linux) на симметричную мультипроцессорную систему пользователь может заметить в меню boot loader, что фактически на его компьютер установилась не одна, а две версии ОС – с поддержкой SMP и без нее.

Асимметричная мультипроцессорная система (asymmetric multiprocessing) – это многопроцессорная компьютерная система, в которой процессоры специализированы по своим функциям. Каждому процессору дается специфическое задание; **главный процессор (master processor)** планирует работу **подчиненных процессов (slave processors).** В такой системе ОС, как правило, выполняется на одном определенном, закрепленном за ней, центральном процессоре. Подобная архитектура более типична для очень больших систем. Пример – система "Эльбрус", которая имела в своем составе, в зависимости от конфигурации, от одного до 10 центральных процессоров, от одного до четырех специализированных **процессоров ввода вывода (ПВВ),** от одного до четырех **процессоров передачи данных (ППД).**

Схема организации SMP-архитектуры компьютеров приведена на рис. 3.1.

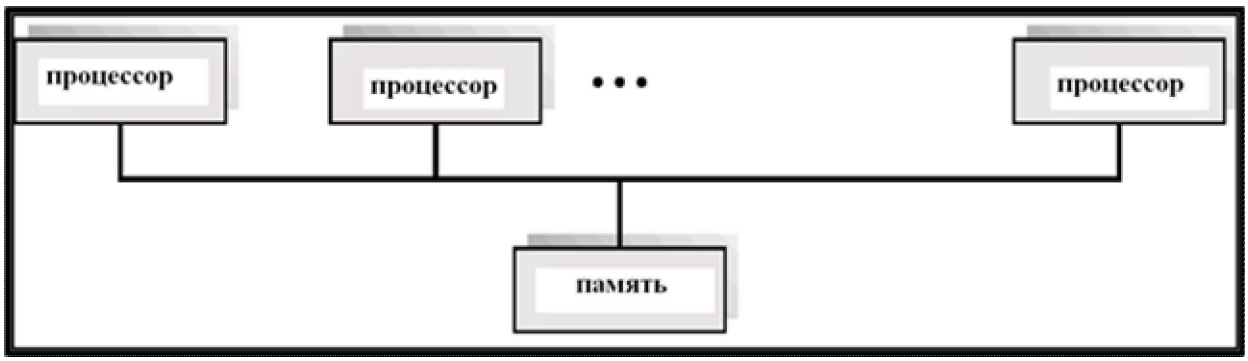

Рис. 3.1. Схема организации SMP-архитектуры компьютеров

Распределенные компьютерные системы и особенности их ОС

В **распределенной системе (distributed system)** вычисления распределены между несколькими физическими процессорами (компьютерами), объединенными между собой в сеть.

Слабо связанная система (loosely coupled system) – распределенная компьютерная система, в которой каждый процессор имеет свою локальную память, а различные процессоры взаимодействуют между собой через **линии связи –** высокоскоростные шины, телефонные линии, беспроводную связь (Wi-Fi, EVDO, Wi-Max и др.).

Преимущества распределенных систем:

1. **Разделение (совместное использование) ресурсов**: в распределенной системе различные ресурсы могут храниться на разных компьютерах. Нет необходимости дублировать программы или данные, храня их копии на нескольких компьютерах.
2. **Совместная загрузка (load sharing**): каждому компьютеру в распределенной системе может быть поручено определенное задание, которое он выполняет параллельно с выполнением другими компьютерами своих заданий.
3. **Надежность**: при отказе или сбое одного из компьютеров распределенной системы его задание может быть перераспределено другому компьютеру, чтобы сбой в минимальной степени повлиял или вовсе не повлиял на итоговый результат.
4. **Связь**: в распределенной системе все компьютеры связаны друг с другом, так что, например, при необходимости возможен удаленный вход с одного компьютера на другой с целью использования ресурсов более мощного компьютера.

В распределенной системе компьютеры связаны в сетевую инфраструктуру, которая может быть:

1. локальной сетью (local area network - LAN);
2. глобальной или региональной сетью (wide area network - WAN).

По своей организации распределенные системы могут быть клиент-серверными (client-server) или одноранговыми (peer-to-peer) системами. В **клиент-серверной** системе определенные компьютеры играют роль серверов, а остальные – роль клиентов, пользующихся их услугами. Подобная организация распределенных систем наиболее распространена, и мы рассмотрим ее подробнее. В **одноранговой** распределенной системе все компьютеры равноправны.

Структура клиент-серверной системы изображена на рис. 3.2.

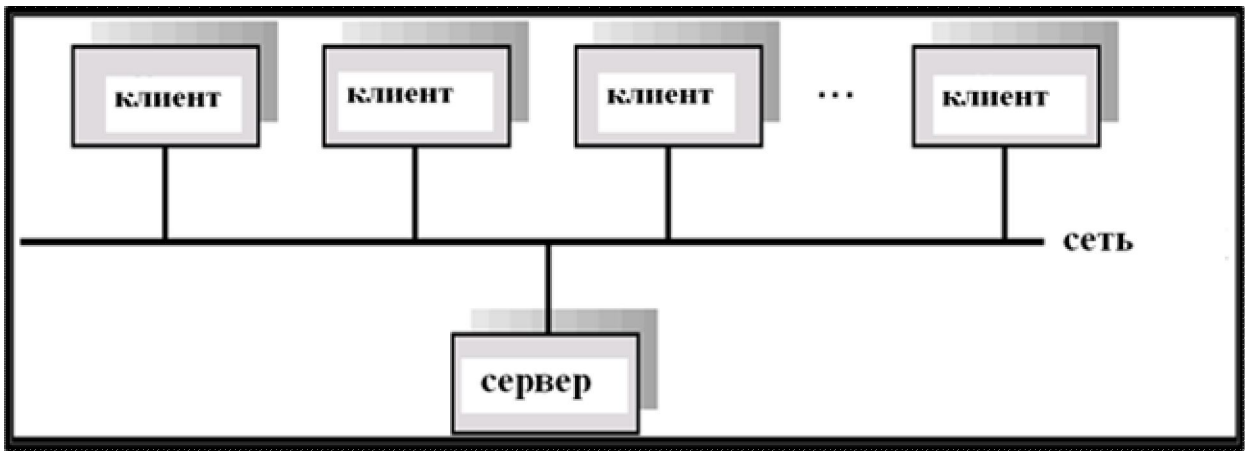

Рис. 3.2. Структура клиент-серверной системы

Виды серверов в клиент-серверных компьютерных системах

Клиент-серверная архитектура распределенных систем весьма широко распространена и поддержана операционными системами. Поэтому очень важно знать, какие виды и функции серверов предлагают современные распределенные системы.

Файл-сервер (file server) – компьютер и программное обеспечение, предоставляющие доступ к подмножеству файловых систем, расположенных на дисках компьютера-сервера, другим компьютерам локальной сети (LAN). Пример – серверное программное обеспечение **SAMBA (SMB –** сокращение от **Server Message Block)** для ОС типа UNIX (Linux, FreeBSD, Solaris и т.д.), обеспечивающее доступ с Windows-компьютеров локальной сети к файловым системам UNIX-машин. Samba также реализована для платформы Macintosh / MacOS.

Сервер приложений (application server) – компьютер и программное обеспечение, предоставляющее вычислительные ресурсы (память и процессор) и необходимое окружение для удаленного запуска определенных классов (как правило, больших) приложений с других компьютеров локальной сети. Примеры серверов приложений - WebSphere (IBM), WebLogic (BEA) – наилучшие из известных серверов приложений, работающих в Java Enterprise Edition (JEE).

Сервер баз данных (database server) – компьютер и программное обеспечение, предоставляющее доступ другим компьютерам сети к базам данных, расположенным на компьютере-сервере. Пример: серверное программное обеспечение для доступа к базам данных Microsoft SQL Server.

Веб-сервер (Web server) – компьютер и программное обеспечение, предоставляющее доступ клиентам через WWW к Web-страницам, расположенным на компьютере-сервере. Пример: свободно распространяемый Web-сервер Apache.

Прокси-сервер – компьютер и программное обеспечение, являющиеся частью локальной сети и поддерживающие эффективное обращение компьютеров локальной сети к Интернету, фильтрацию трафика, защиту от внешних атак. Proxy-сервер обычно встроен в операционную систему.

Сервер электронной почты – компьютер и программное обеспечение, выполняющие отправку, получение и "раскладку" электронной почты для компьютеров некоторой локальной сети. Могут обеспечивать также **криптование** почты (email encryption) – шифрование электронных писем перед отправкой адресатам из определенного сетевого домена (как правило, заказчику) и их дешифровку после получения от заказчика.

Серверный бэк-энд (Server back-end) – группа (пул) связанных в локальную сеть серверных компьютеров, используемых вместо одного сервера, в целях большей надежности и предоставления большего объема ресурсов. Другой термин, близкий к этому, - **центр обработки данных (data center).** Эти понятия особенно актуальны в связи со все более

широким распространением облачных вычислений, являющихся, с этой точки зрения, наиболее современной реализацией клиент-серверной схемы взаимодействия.

Кластерные вычислительные системы и их ОС

Компьютерные кластеры весьма популярны для научных вычислений. Компьютеры в кластере, как правило, связаны между собой через быструю локальную сеть. Кластеризация позволяет двум или более системам использовать общую память. Кластеризация обеспечивает высокую надежность. Различают компьютерные кластеры двух видов:

- **асимметричная кластеризация (asymmetric clustering) –** организация компьютерного кластера, при которой один компьютер выполняет приложение, а остальные простаивают;
- **симметричная кластеризация (symmetric clustering) -** организация компьютерного кластера, при которой все машины кластера исполняют одновременно различные части одного большого приложения.

Различают также:

- **кластеры с высокоскоростным доступом (high-availability clusters) –** компьютерные кластеры, обеспечивающие оптимальный доступ к ресурсам, предоставляемым компьютерами кластера, например, к базам данных;
- **кластеры с балансировкой загрузки (load-balancing clusters)** – компьютерные кластеры, которые имеют несколько входных компьютеров, балансирующих запросы (front-ends), распределяющих задания между компьютерами серверного back-end'а (серверной фермы).

Кластеры часто используются в университетах (например, установлены на нескольких факультетах СПбГУ) и в исследовательских центрах (например, CERN, Швейцария). Операционные системы для кластеров: Windows 2003 for clusters; Windows 2008 High-Performance Computing.

Системы и ОС реального времени

Системы реального времени часто используются как управляющие устройства для специальных приложений, - например, для научных экспериментов; в медицинских системах, связанных с изображениями; системах управления в промышленности; системах отображения (display); системах управления космическими полетами, АЭС и др. Для таких систем характерно наличие и выполнение четко определенные временные ограничения (время реакции – response time; время наработки на отказ и др.).

Различаются системы реального времени видов **hard real-time** и **soft real-time**.

Hard real-time – системы – системы реального времени, в которых при нарушении временных ограничений может возникнуть критическая ошибка (отказ) управляемого ею объекта. Примеры: система управления двигателем автомобиля; система управления кардиостимулятором. В таких системах вторичная память ограничена или отсутствует; данные хранятся в оперативной памяти (RAM) или постоянном запоминающем устройстве (ПЗУ, ROM). При использовании таких систем возможны конфликты с системами разделения времени, не имеющие места для ОС общего назначения. Выражаясь более простым языком, при работе подобных систем не допускаются прерывания; все необходимые данные для основного цикла работы системы должны предварительно быть загружены в память; процесс, выполняющий код такой системы, не должен подвергаться откачке на диск. ОС для таких систем обычно упрощены, вместо виртуальной памяти выделяется физическая, все другие виды виртуализации ресурсов исключены. Популярной практикой разработки ОС реального времени является практика разработки таких ОС на основе открытых исходных кодов ОС общего назначения путем "отсечения всего лишнего". Однако при этом следует соблюдать осторожность. Автору приходилось консультировать разработчиков системы реального времени для "Эльбруса", которые использовали для своей системы низкоуровневую процедуру выделения физической памяти, но не учли ее возможных конфликтов с общей системой виртуальной памяти ОС Эльбрус; в результате выделяемая память иногда

"портилась" … в результате изменения связующей информации в списке областей свободной памяти, который использовался механизмом виртуальной памяти "Эльбруса".

Soft real-time – системы – системы реального времени, в которых нарушение временных ограничений не приводит к отказу управляемого ею объекта. Обычно это системы управления несколькими взаимосвязанными системами с постоянно изменяющейся ситуацией. Пример - система планирования рейсов на коммерческих авиалиниях**.** В случае какой-либо задержки в работе такой системы, в худшем случае, пассажирам некоторых рейсов придется немного подождать в аэропорту, но никаких фатальных последствий не будет. Подобные системы имеют ограниченную полезность для промышленных систем управления и в роботике. Они также полезны в современных приложениях (например, для мультимедиа и виртуальной реальности), требующих развитых возможностей ОС.

Карманные компьютеры (handhelds) и их ОС

К данному классу устройств, как уже отмечалось, относятся карманные персональные компьютеры (КПК), или Personal Digital Assistants (PDA), и мобильные телефоны. Особенности и проблемы данного класса компьютеров следующие:

- ограниченный объем памяти;
- относительно медленные процессоры: для мобильного устройства типично ожидание выполнения простейшей команды в течение нескольких секунд, что неудобно;
- маленький размер экрана мониторов (дисплеев), отсюда – необходимость в специализированном программном обеспечении для поддержки GUI; например, в Java Micro Edition (JME) – версии Java для мобильных устройств – невозможно использовать удобные общие пакеты AWT и Swing для разработки GUI; вместо них разработчику в JME предлагаются специализированные пакеты вида **javax.microelectronics…lcdui,** несовместимые со стандартным изданием Java (JSE), что делает код зависимым от типа устройства, а иногда – и от конкретных моделей мобильных телефонов, которые имеют разные размеры экранов;
- невысокая скорость связи через Интернет: например, GPRS-модем мобильного телефона обеспечивает связь примерно со скоростью dial-up – 3-5 килобайт в секунду;
- связь для передачи данных осуществляется через Bluetooth или IrDA (причем последний часто отсутствует); имеются не все необходимые порты: например, часто в мобильных устройствах отсутствует порт USB, т.е. для них нельзя использовать "флэшки", что весьма неудобно, и приходится использовать специальные сверхминиатюрные диски типа SmartMedia, для которых в настольных компьютерах не всегда имеются адаптеры для чтения.

Тем не менее, современные средства коммуникации (например, Wi-Fi) и совместимые с персональными компьютерами порты и внешние модули памяти начинают использоваться и на карманных и мобильных устройствах.

В операционных системах и другом системном программном обеспечении для карманных и мобильных устройств приходится учитывать все эти ограничения, в частности, ограниченный объем памяти. В связи с этим целый ряд удобных повседневных программистских возможностей приходится для мобильных устройств запрещать (например, в JME нет вещественной арифметики).

Развитие концепций и возможностей ОС представлено на рис. 3.3.

На схеме хорошо видны аналогичные "волны" ("витки") развития ОС - сначала для mainframe-компьютеров, затем – для миникомпьютеров, для персональных и для карманных компьютеров. Каждая волна проходит в своем развитии определенные этапы. ОС развиваются от резидентных мониторов до поддержки пакетного режима (для ранних моделей компьютеров), затем – режима разделения времени, многопользовательских и сетевых возможностей.

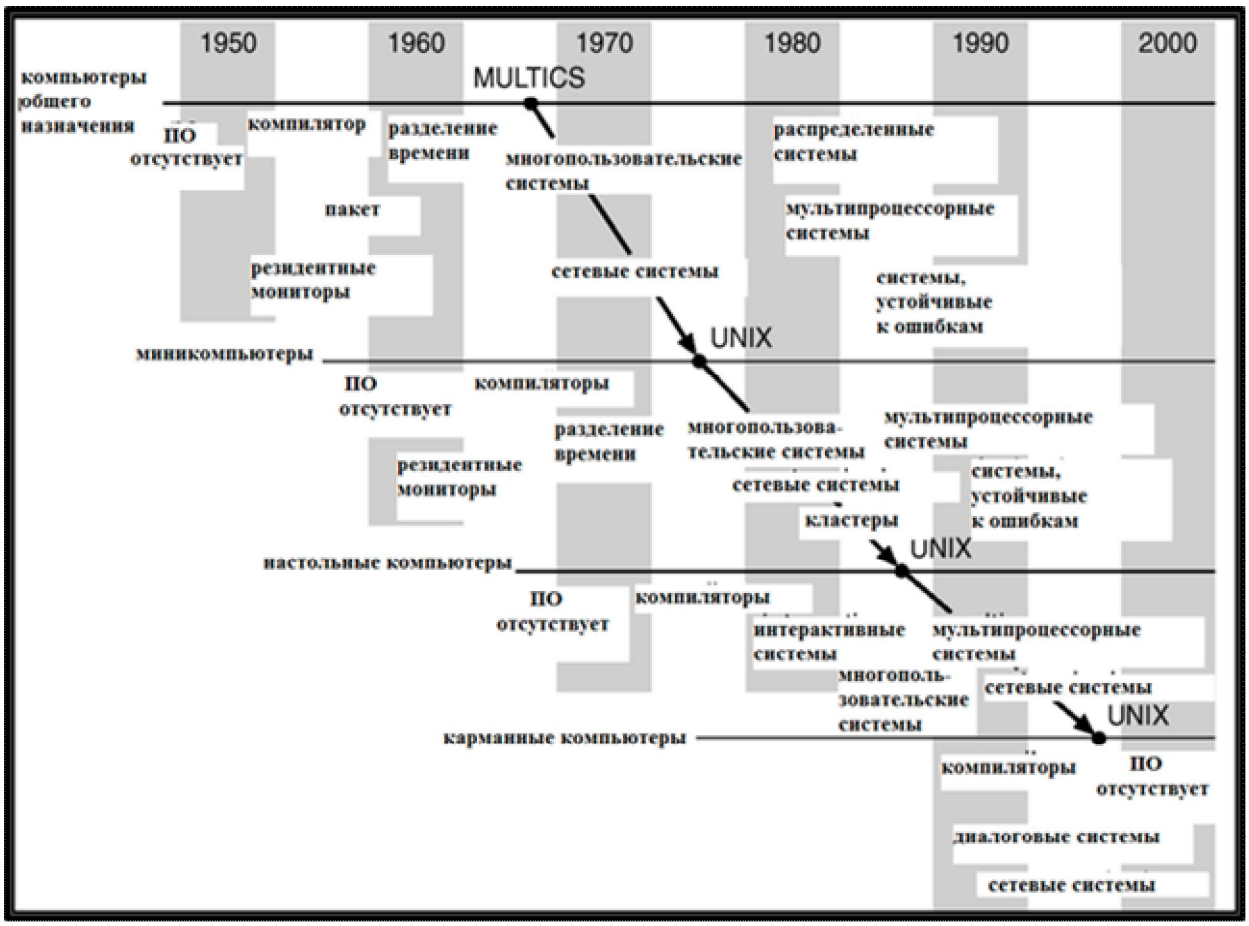

Рис. 3.3. Развитие концепций и возможностей ОС.

Вычислительные среды

В современном мире ИТ имеет место тенденция к интеграции описанных выше устройств и их локальных сетей в **вычислительные среды** – интегрированные распределенные компьютерные системы для решения задач в различных проблемных областях. Вычислительные среды подразделяются на следующие виды:

- **традиционные вычислительные среды** – локальные и региональные сети, используемые в течение нескольких десятков лет;
- **Web-ориентированные вычислительные среды –** вычислительные среды на основе Web-сервисов, характерные для настоящего времени, начиная с 1990-х гг.; к этому классу относятся и среды для облачных вычислений;
- **встроенные (embedded) вычислительные среды –** вычислительные среды для специализированных устройств, например, сети микропроцессоров, встроенных в элементы линии электропередач.

Все эти виды вычислительных сред должны адекватно обслуживаться операционными системами, в чем и состоят ближайшие задачи их разработки.

Облачные вычисления и ОС для облачных вычислений

Облачные вычисления (**cloud computing**) являются одним из наиболее популярных направлений развития ИТ. **"Облако" (cloud) –** это уже десятки лет используемая метафора для изображения сервисов, предоставляемых через Интернет или другую коммуникационную сеть (например, через ATM-сеть). **Облачные вычисления** – модель вычислений, основанная на **динамически масштабируемых (scalable)** и **виртуализованных** ресурсах (данных, приложениях, ОС и др.), которые доступны и используются как **сервисы** через Интернет и реализуются с помощью высокопроизводительных **центров обработки данных (data centers)**

С точки зрения пользователей, существует совокупность "облаков" (общедоступные, корпоративных, частных и др.), предоставляемых различными компаниями, для

использования мощных вычислительных ресурсов, которых нет у индивидуального пользователя. Как правило, "облачные" сервисы платные. Из бесплатных назовем Windows Live.

Недостаток облачных вычислений в том, что пользователь оказывается полностью зависимым от используемого им "облака" (в котором доступны используемые им данные и программы) и не может управлять не только работой "облачных" компьютеров, но даже резервным копированием своих данных. В связи с этим возникает целый ряд важных вопросов о безопасности облачных вычислений, сохранении конфиденциальности пользовательских данных и т.д.; далеко не все из них на данный момент решены.

Серьезной проблемой организации облачных вычислений с точки зрения аппаратуры центров обработки данных является экономия электроэнергии и проблема распределения загрузки, так как облачные вычисления в каждом центре обработки данных имеют (или в ближайшем будущем будут иметь) **миллионы** удаленных пользователей.

Наиболее популярная "облачная" платформа – Microsoft Windows Azure (облачная ОС) и Microsoft Azure Services Platform (реализованная на основе Microsoft.NET). Windows Azure можно рассматривать как "ОС в облаке". Пользователю нет необходимости беспокоиться о ее инсталляции на его компьютере, который может не иметь для этого необходимых ресурсов. Все, что требуется, это иметь Web-браузер и минимальный пакет надстроек (plug-ins) для запуска и использования через браузер облачных сервисов.

В настоящее время все крупные компании (Microsoft, IBM, HP, Dell, Oracle и др.) разрабатывают свои системы облачных вычислений; имеется тенденция к интеграции этих корпоративных систем в единое доступное пользователю "облако".

Ключевые термины

Boot loader - загрузчик одной из нескольких ОС, установленных на некотором компьютере, управляемый специальным меню при включении компьютера.

Double bootable system - компьютер, на котором установлены две (или более) операционных системы, при включении которого пользователю выдается начальное меню для уточнения, какую именно ОС требуется запустить.

Hard real-time – система реального времени, в которой при нарушении временных ограничений может возникнуть критическая ошибка (отказ) управляемого ею объекта.

Original Equipment Manufacturer (OEM) - фирма-разработчик какого-либо внешнего устройства, обычно разрабатывающая и **драйвер** к нему.

Soft real-time – система реального времени, в которой нарушение временных ограничений не приводит к отказу управляемого ею объекта.

Асимметричная кластеризация (asymmetric clustering) – организация компьютерного кластера, при которой один компьютер выполняет приложение, а остальные простаивают.

Асимметричная мультипроцессорная система (asymmetric multiprocessing) – многопроцессорная компьютерная система, в которой процессоры специализированы по своим функциям; имеется главный процессор, планирующий работу подчиненных процессоров.

Веб-сервер (Web server) – компьютер и программное обеспечение, предоставляющее доступ клиентам через WWW к Web-страницам, расположенным на компьютере-сервере.

Вычислительная среда – интегрированная распределенная компьютерная система для решения задач в каких-либо проблемных областях.

Драйвер – низкоуровневая системная программа для управления каким-либо внешним устройством (например, жестким диском).

Кластеры с балансировкой загрузки (load-balancing clusters) – компьютерные кластеры, которые имеют несколько входных компьютеров, балансирующих запросы (front-ends), распределяющих задания между компьютерами серверного бэк-энда.

Кластеры с высокоскоростным доступом (high-availability clusters, HAC) – компьютерные кластеры, обеспечивающие оптимальный доступ к ресурсам, предоставляемым компьютерами кластера, например, к базам данных.

Клиент-серверная система – распределенная компьютерная система, в которой определенные компьютеры играют роль специализированных серверов, а остальные – роль клиентов, пользующихся их услугами.

Многоядерный (multi-core) компьютер – компьютерная система, основанная на тесно связанных друг с другом процессорах (**ядрах**), находящихся в одном кристалле, разделяющих ассоциативную память (кэш) второго уровня и работающих на общей памяти.

Облачные вычисления – модель вычислений, основанная на **динамически масштабируемых (scalable)** и **виртуализованных** ресурсах (данных, приложениях, ОС и др.), которые доступны и используются как **сервисы** через Интернет и реализуются с помощью высокопроизводительных **центров обработки данных (data centers).**

Параллельная компьютерная система – мультипроцессорная система, состоящая из нескольких непосредственно взаимодействующих процессоров.

Параллельный порт, или **LPT** (**аббревиатура от Line PrinTer**) – порт для подключения устаревших моделей принтеров. Для подключения принтера через данный порт требуется предварительно отключить и принтер, и компьютер.

Прокси-сервер – компьютер и программное обеспечение, являющиеся частью локальной сети и поддерживающие эффективное обращение компьютеров локальной сети к Интернету, фильтрацию трафика, защиту от внешних атак.

Распределенная система (distributed system) – компьютерная система, в которой вычисления распределены между несколькими физическими процессорами (компьютерами), объединенными между собой в сеть.

Сервер баз данных (database server) – компьютер и программное обеспечение, предоставляющее доступ другим компьютерам сети к базам данных, расположенным на компьютере-сервере локальной сети.

Серверный бэк-энд (Server back-end) – группа (пул) связанных в локальную сеть серверных компьютеров, используемых вместо одного сервера, в целях большей надежности и предоставления большего объема ресурсов.

Сервер приложений (application server) – компьютер и программное обеспечение, предоставляющее вычислительные ресурсы (память и процессор) и необходимое окружение для удаленного запуска определенных классов (как правило, больших) приложений с других компьютеров локальной сети.

Сервер электронной почты – компьютер и программное обеспечение, выполняющие отправку, получение и "раскладку" электронной почты для компьютеров некоторой локальной сети. Может обеспечивать также криптование почты (email encryption).

Сетевой адаптер (сетевая карта) – устройство для подключения компьютера к локальной сети.

Симметричная кластеризация (symmetric clustering) - организация компьютерного кластера, при которой все машины кластера исполняют одновременно различные части одного большого приложения.

Симметричная мультипроцессорная система (symmetric multiprocessing - SMP) – многопроцессорная компьютерная система, все процессоры которой равноправны и используют одну и ту же копию ОС; операционная система при этом может выполняться на любом процессоре.

Слабо связанная система (loosely coupled system) – распределенная компьютерная система, в которой каждый процессор имеет свою локальную память, а различные процессоры взаимодействуют между собой через линии связи.

Сканер – устройство для оцифровки бумажных изображений, например, подписанных или рукописных документов.

Тесно связанная (tightly coupled) система – параллелльная компьютерная система, в которой процессоры разделяют общую память и таймер (такты); взаимодействие между ними происходит через общую память.

Файл-сервер (file server) – компьютер и программное обеспечение, предоставляющие доступ к подмножеству файловых систем, расположенных на дисках компьютера-сервера, другим компьютерам локальной сети.

Краткие итоги

Операционные системы для персональных компьютеров, предназначенных для одного пользователя, поддерживают многозадачный режим работы, взаимодействие с широким набором внешних устройств, возможности сетевого взаимодействия и имеют удобный дружественный пользовательский интерфейс. На одном ПК могут быть установлены несколько ОС.

Параллельные компьютерные системы могут быть тесно связанными (VLIW, многоядерными и др.) и слабо связанными, или распределенными. Многопроцессорные системы подразделяются на симметричные (SMP) и асимметричные. В распределенных системах компьютеры соединены через сеть. Преимущества параллельных систем – улучшенная производительность, более высокая надежность, устойчивость к ошибкам. Сетевые системы могут быть одноранговыми или клиент-серверными. Сети подразделяются на локальные, региональные и глобальные.

В клиент-серверных системах клиентам предоставляются файл-серверы, серверы приложений, серверы баз данных, веб-серверы, прокси-серверы, серверы электронной почты, бэк-энды (пулы, фермы) серверов.

Кластерные вычислительные системы используются для научных вычислений и подразделяются на симметричные, асимметричные, кластеры с балансировкой загрузки, кластеры с высокоскоростным доступом.

Системы реального времени служат для управления различными объектами и подразделяются на hard real-time (в которых нарушение временных ограничений приводит к отказу объекта) и soft real-time (в которых нарушение временных ограничений не имеет столь фатальных последствий).

Карманные и мобильные компьютерные системы широко используются, хотя и имеют целый ряд недостатков – малый объем памяти, медленные процессоры, невысокая скорость взаимодействия через Интернет, маленькие экраны, неудобство ввода информации, отсутствие традиционных портов. ОС для этих устройств должны учитывать все эти особенности. Для разработки программного обеспечения таких устройств чаще всего используется Java Micro Edition (JME).

Развитие ОС для различных типов компьютеров (mainframe, миникомпьютеров, ПК, настольных и карманных компьютеров) происходит аналогичными волнами (витками), от резидентных однозадачных мониторов - к поддержке разделения времени, многозадачности и работы в сети.

Интегрированные вычислительные среды подразделяются на традиционные, веб-ориентированные и встроенные.

Облачные вычисления обеспечивают клиентам доступ к веб-сервисам центров обработки данных через веб и браузер, без необходимости инсталляции ПО и хранения данных на компьютере клиента. Они являются наиболее популярной моделью вычислений в настоящее время. Недостаток – полная зависимость клиента от используемого им облака. Наиболее распространенная среда и ОС для облачных вычислений – Microsoft Windows Azure.

5. Лекция: Архитектура компьютерной системы

В лекции подробно рассмотрена архитектура компьютерной системы: управление прерываниями, памятью, вводом-выводом, иерархия памяти, ассоциативная память (кэширование), защита памяти, аппаратная защита памяти в системах с теговой архитектурой.

Введение

В данной лекции рассмотрим более подробно архитектуру компьютерной системы. Будут рассмотрены следующие вопросы:

1. функционирование компьютерной системы
2. архитектура ввода-вывода
3. структура памяти
4. иерархия памяти
5. аппаратная защита памяти
6. общая архитектура системы.

Архитектура компьютерной системы

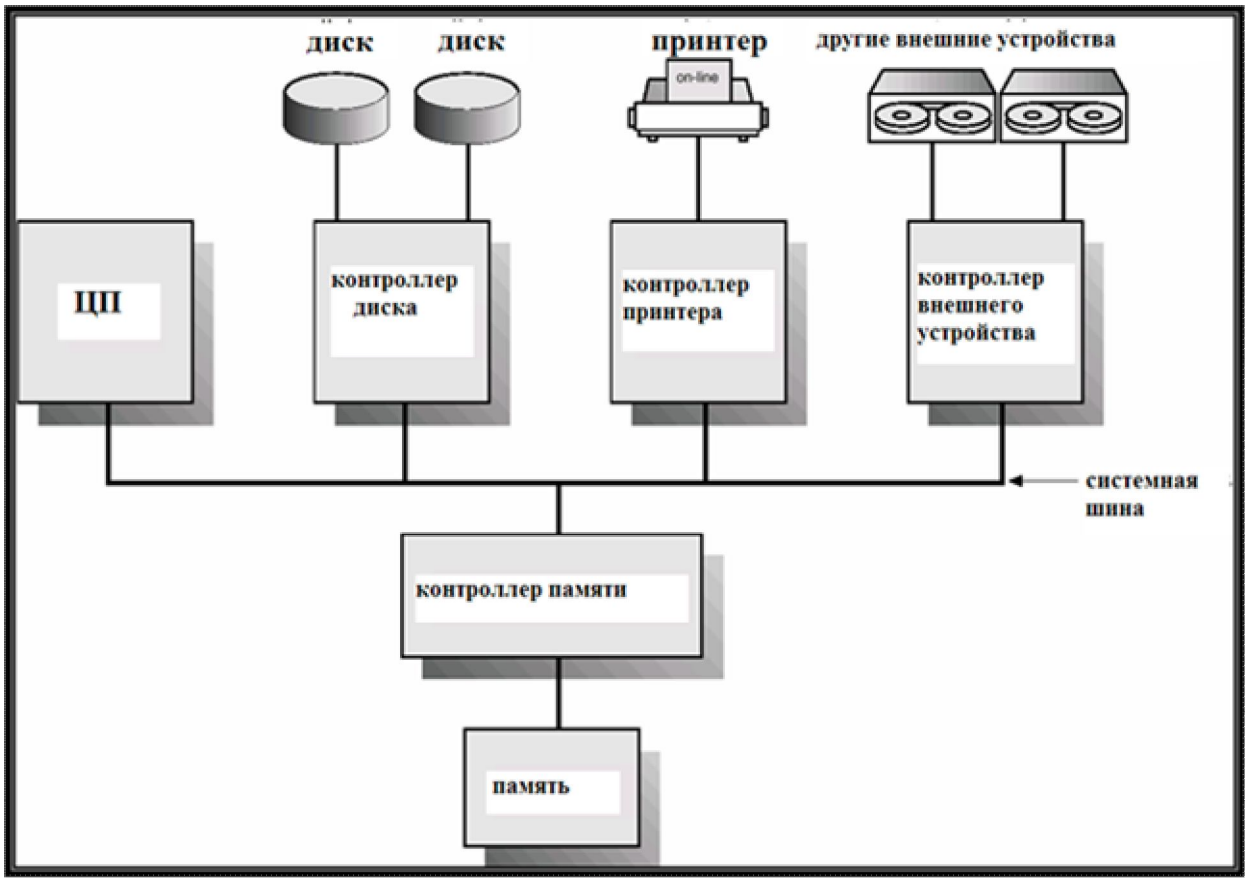

Рис. 4.1. Архитектура компьютерной системы.

Компьютерная система имеет модульную структуру. Для каждого устройства (память, внешние устройства) в системе имеется специальное устройство управления (иначе говоря, специальный процессор), называемый **контроллером устройства**. Все модули (центральный процессор, память и контроллер памяти, внешние устройства и их контроллеры) соединены между собой **системной шиной (system bus),** через которую они обмениваются сигналами. Как мы уже знаем работой каждого контроллера управляет **драйвер -** специализированная низкоуровневая программа, являющаяся частью ОС.

Вот типичная структура современной настольной или портативной компьютерной системы, с указанием наиболее распространенных типов устройств и их характеристик.

Центральный процессор – устройство, выполняющее **команды (instructions)** компьютерной системы. В современных компьютерах, как правило, он является **многоядерным**, т.е. имеет в своем составе от 2 до 32 ядер (копий) процессора, параллельно работающих на общей памяти, либо **гибридным**, состоящим из центрального и графического процессоров. Производительность каждого ядра – 3 – 3.2 GHz. Заметим, что под производительностью понимается в данном случае **тактовая частота процессора (ядра)** – время выполнения им одной самой простой машинной команды. Однако есть и другие важные

факторы, определяющие общую производительность системы, - тактовая частота памяти и системной шины. Фактически итоговую производительность системы можно оценить по самой медленной из этих частей системы (обычно это системная шина). Эти характеристики необходимо принимать во внимание при выборе и покупке компьютера.

Оперативная (основная) память, или просто **память** – устройство, хранящее обрабатываемые данные. Объем памяти – 1 – 16 гигабайт и более; меньший объем памяти использовать не рекомендуется, так как это может привести к значительному замедлению системы. Тактовая частота памяти – 667 MHz – 1.5 GHz.

Системная шина – устройство, к которому подсоединены все модули компьютера и через которое они обмениваются сигналами, например, о прерываниях. Тактовая частота шины – 1 – 1.5 GHz (это и есть фактически некая суммарная производительность системы). Обычно используется шина типа **PCI (Personal Computer Interface).** К ней могут быть подсоединены процессор, память, диски, принтер, модем и другие внешние устройства.

Порты – устройства с разъемами для подключения к компьютеру внешних устройств. Каждый порт имеет свой контроллер (и, соответственно, свой драйвер).

Чаще всего используется **порт USB (Universal Serial Bus),** с характерным плоским разъемом, размером порядка 1 см, с изображением трезубца. К портам USB могут подключаться большинство видов устройств, причем для этого не требуется предварительно отключать компьютер и подключаемое устройство, что очень удобно. Имеется несколько стандартов USB с различным быстродействием. Наиболее распространен ныне стандарт USB 2.0, обеспечивающий быстродействие порта 240 – 260 мегабит в секунду. Для сравнения, предыдущий стандарт – USB 1.0 – обеспечивал лишь 10 – 12 мегабит в секунду (как говорится, почувствуйте разницу). Распознать тип USB-порта на Вашем компьютере можно, если вывести информацию об устройствах; в Windows: **Мой компьютер / (правая кнопка мыши) Свойства / Оборудование / Диспетчер устройств / Устройства USB**. При этом контроллер порта USB 2.0 будет обозначен как **расширенный (enhanced).** Если это не так, Вам необходимо модернизировать порты USB или сам компьютер, иначе при переписи на флэшку Вам придется ждать в 20 раз дольше (!). Существуют также "переходники" USB 1.0 -> USB 2.0. Новейший стандарт USB 3.0, реализация которого только началась, обеспечит быстродействие не менее 1 гигабита в секунду. К порту USB можно подключать клавиатуру, мышь, принтеры, сканеры, внешние жесткие диски, флэшки и даже **TV-тюнеры** - устройства для приема телевизионного сигнала с антенны и показа телевизионного изображения на компьютере. Рекомендуется каждое устройство подключать всегда к одному и тому же порту USB, иначе для некоторых устройств (например, того же TV-тюнера) могут возникнуть проблемы.

Порты COM (communication ports) – порты для подключения различных коммуникационных устройств, например, **модемов** – устройств для выхода в Интернет и передачи информации по аналоговой или цифровой телефонной линии. Более старое название стандарта COM-порта – **RS-232**. В компьютерах 10-15 – летней давности к COM-порту часто подключалась мышка (сейчас она, разумеется, подключается через USB). Разъемы COM-портов имеют два формата – "большой" (с 25 контактами - **pins**) и "малый" (с 9 контактами). В современных компьютерах часто разъемы COM-порты отсутствуют, но операционная система, по традиции, имитирует наличие в системе **виртуальных COM-портов** – воображаемых COM-портов, которые ОС как бы инсталлирует в систему при установке, например, драйверов для взаимодействия через Bluetooth или через кабель компьютера с мобильным устройством. При этом физически мобильный телефон или органайзер может быть подключен к порту USB (или соединен с компьютером беспр оводной связью), но все равно для взаимодействия с ним ОС использует виртуальный COM-порт, обычно с большим номером (например, 10 или 15). COM-порт иначе называют **последовательным портом (serial port),** так как, с точки зрения ОС и драйверов, COM-порт – это символьное устройство последовательного действия.

Порт LPT (от line printer), или **параллельный порт** – это ныне уже устаревший вид порта для подключения принтера или сканера, с толстым в сечении кабелем и большим разъемом. Все новые модели принтеров и сканеров работают через USB-порты. Однако иногда приходится решать задачу подключения к новому компьютеру старого принтера. Если на компьютере нет LPT-порта, приходится покупать специальный переходник, подключаемый к USB или другим портам. Однако и здесь возможен сюрприз (по личному опыту автора) – разъем LPT-порта имеет несколько не совместимых друг с другом модификаций. Лучше всего иметь в домашнем "вычислительном центре" один старый компьютер с LPT-портом и через него и подключать старые принтеры, обеспечивая обращение к ним с других компьютеров через домашнюю локальную сеть. Неудобство LPT-порта в том, что он требует предварительно выгрузить ОС и выключить принтер, и только после этого выполнять подсоединение к компьютеру, иначе возможен выход из строя принтера или компьютера. LPT-порт может, как правило, работать и для ввода информации, например, со сканером, но для этого требуется в низкоуровневой утилите Setup, запустив ее при загрузке ОС (обычно – клавишей Del), установить для LPT-порта специальный режим работы: **EPP – Extended Parallel Port**.

Порты SCSI и SCSI-устройства. **SCSI (Small Computer System Interface**; произносится "**скАзи",** с ударением на первом слоге) – интерфейс, адаптеры и порты для подключения широкого спектра внешних устройств – жестких дисков, CD-ROM / DVD-ROM, сканеров и др. Стандарт SCSI был предложен в начале 1980-х гг. и получил широкое распространение, благодаря фирме Sun, которая широко использовала его в своих рабочих станциях. Характерной удобной возможностью SCSI является возможность подключения к одному SCSI-порту **гирлянды (цепочки) SCSI-устройств** (до 10), каждый из которых имеет уникальный для данного соединения **SCSI ID** – число от 0 до 9, устанавливаемое обычно на задней панели SCSI-устройства. Например, по традиции, SCSI ID сканера обычно равен 4. На одном из концов цепочки – SCSI-порт с контроллером, на другом – **терминатор –** переключатель на задней панели устройства, устанавливаемый в определенное положение как признак конца SCSI-цепочки. Каждое устройство, кроме последнего, соединено со следующим SCSI-устройством специальным кабелем. SCSI-разъем напоминает разъем порта LPT, однако имеет по бокам специальные металлические захваты ("лапки") для большей надежности подключения. Преимущество SCSI, кроме возможности использования гирлянд устройств, в его быстродействии, а также надежности. Ранние модели SCSI имели скорость обмена информацией до 10-12 мегабит в секунду, сейчас – 240-250 мегабит в секунду. Имеется несколько стандартов SCSI (в том числе – Wide SCSI, Ultra Wide SCSI), к сожалению, не совместимых по разъемам. Автор до сих пор использует SCSI-сканер 10-летней давности, подключенный к старому компьютеру, и имеет большой положительный опыт использования SCSI-устройств для рабочих станций SPARC.

Порт **VGA (Video Graphic Adapter)** используется для подключения **монитора (дисплея),** управляемого **графическим контроллером (процессором)**.

IEEE 1394 (FireWire) – порты для подключения цифровых видеокамер или фотоаппаратов. Характерная особенность – небольшой блестящий плоский разъем шириной 3-5 мм (имеются два его стандарта). Порт работает в дуплексном режиме, т.е. позволяет управлять не только вводом информации с камеры в компьютер, но и установками самой камеры (например, перемоткой ленты) с помощью компьютерной программы (например, Windows Movie Maker). С помощью такого же порта может быть подключен также телевизор, имеющий интерфейс FireWire. Характерной особенностью современных компьютеров является то, что FireWire-порты монтируются прямо на **материнской плате (motherboard) –** основной печатной плате компьютера, на которой смонтированы процессор и память, - столь большое значение придают производители компьютеров портам для обмена мультимедийной информацией. В таких случаях в технических характеристиках компьютера обычно указывается: **"FireWire on board (на борту)".** Рекомендуется не путать **FireWire** с **Wi-Fi** –

стандартом быстрой беспроводной связи; эти сокращения забавно напоминают друг друга из-за привязанности американцев к аббревиатурам в "детском стиле".

HDMI (High Definition Multimedia Interface) – интерфейс и порт. позволяющий подключить к компьютеру телевизор или другое видеооборудование, обеспечивающее наилучшее качество воспроизведения (HD – High Definition). Разъем HDMI напоминает разъем USB. HDMI-порт входит в комплектацию всех современных портативных компьютеров.

Bluetooth – устройства для беспроводного подключения (с помощью радиосвязи) к компьютеру мобильных телефонов, органайзеров, а также наушников, плейеров и многих других полезных устройств. Удобство Bluetooth в том, что компьютер и телефон остаются соединенными, даже если отойти от компьютера с телефоном на некоторое расстояние (например, в другую комнату), не более 10-15 метров (Bluetooth 2.0). Новый стандарт Bluetooth 3.0 обеспечивает взаимодействие на расстоянии 200-250 м. Обычно портативные компьютеры комплектуются встроенными адаптерами Bluetooth, либо можно приобрести адаптер Bluetooth, подключаемый через USB. Недостаток Bluetooth – относительно маленькая суммарная скорость передачи информации. Например, при пересылке на компьютер через Bluetooth с мобильного телефона Nokia 3230 цифровой фотографии объемом 500 килобайт требуется ждать порядка 10 – 15 секунд.

Инфракрасный порт (IrDA) – порт для подключения ноутбука к мобильному телефону (или двух ноутбуков друг к другу) через инфракрасную связь. Неудобство портов IrDA – необходимость установки двух соединяемых устройств рядом, на расстоянии 20-30 см друг от друга, без физических препятствий между ними. Скорость передачи информации – 10-12 мегабит в секунду. Современные ноутбуки уже **не** комплектуются портами IrDA.

Имеются также **сетевые устройства – порты и адаптеры –** для подключения компьютера к локальной сети.

Функционирование компьютерной системы

Преимущество описанного модульного подхода к аппаратуре в том, что центральный процессор, память и внешние устройства могут функционировать параллельно. Работой каждого устройства управляет специальный контроллер. При необходимости выполнения ввода-вывода центральный процессор генерирует прерывание, в результате которого вызывается операционная система, в свою очередь, в качестве реакции на прерывание запускающая драйвер устройства, соответственно, активизирующий его контроллер. Каждый контроллер устройства имеет локальный буфер – специализированную память для обмена информацией между компьютером и устройством. Для того, чтобы контроллер мог начать вывод на устройство, предварительно центральный процессор (точнее, драйвер устройства, запущенный на нем) должен переслать информацию из заданной области оперативной памяти в буфер устройства. Далее контроллер устройства уже выполняет вывод информации из буфера на само устройство (например, записывает ее в заданную область жесткого диска). По окончании обмена информацией, контроллер генерирует сигнал о **прерывании (interrupt)** по системной шине, этим информируя процессор об окончании операции. Для того, чтобы избежать повторных пересылок больших объемов информации, в современных компьютерах применяют **DMA (Direct Memory Access) – контроллеры** – контроллеры с прямым доступом к оперативной памяти. Такие контроллеры используют при обмене с устройством не свою специализированную память, а напрямую область оперативной памяти, в которой и размещается буфер обмена.

Обработка прерываний

Операционную систему можно рассматривать как **программу, управляемую прерываниями (interrupt-driven program).** Прерывание центрального процессора передает управление подпрограмме обработки данного вида прерываний, являющейся частью ОС. В большинстве компьютеров этот механизм реализован через **вектор прерываний (interrupt vector)** – резидентный массив в оперативной памяти, в котором хранятся доступные по номерам прерываний адреса подпрограмм-обработчиков прерываний (модулей ОС). При

обработке прерывания аппаратура и ОС сохраняют **адрес прерванной команды.** При возобновлении вычислений будет вновь повторено выполнение прерванной команды.

Очевидно, что при обработке прерывания, в свою очередь, может возникнуть другое прерывание. В этом случае новое входящее прерывание **задерживается (disabled),** и информация о нем запоминается в **очереди прерываний** – системной структуре ОС, обеспечивающей поочередную обработку всех возникших прерываний.

Кроме прерываний, генерируемых аппаратурой неявно при вычислениях (например, отсутствие страницы в оперативной памяти), возможно также **программируемое прерывание (trap**; дословно – **ловушка**) с помощью специальной команды процессора, - например, при обнаруженной ошибке в программе. В случае такого прерывания также работает общий механизм запуска обработчика прерывания – части ОС. Таким образом, с упрощенной точки зрения, ОС можно рассматривать как набор обработчиков прерываний.

При прерывании ОС сохраняет **состояние процессора –** значения регистров и значение **счетчика команд (program counter – PC)** – адреса прерванной команды**.** Обработчик прерывания в ОС определяет по содержимому сегмента объектного кода, какого вида прерывание возникло и какие действия по его обработке следует предпринять. Среди возможных видов прерываний, кроме фиксации различных ошибок, имеются также **прерывания по таймеру** – периодические прерывания через определенный квант времени, предназначенные для **опроса устройств (polling)** – действий операционной системы по периодической проверке состояния всех портов и внешних устройств, которое может меняться с течением времени: например, к USB-порту была подключена флэшка; принтер закончил печать и освободился, и т.д. ОС выполняет реконфигурацию системы и корректирует системные таблицы, хранящие информацию об устройствах.

Архитектура ввода-вывода

На рис. 4.2 изображена временная диаграмма прерываний процессора, выполняющего ввод-вывод.

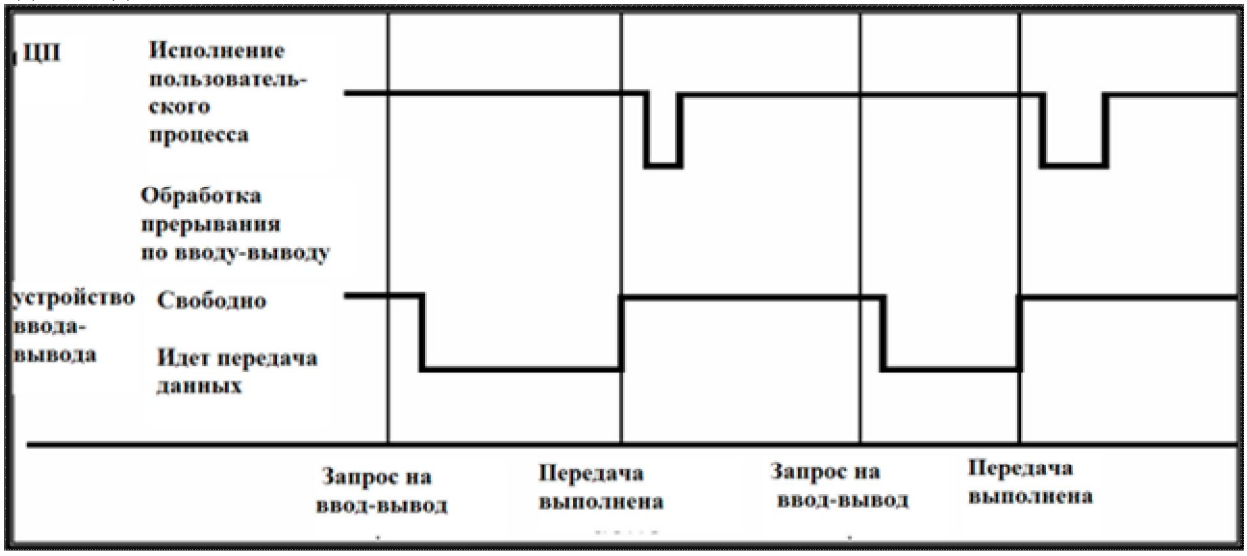

Рис. 4.2. Временная диаграмма прерываний процессора при вводе-выводе.

На диаграмме видны моменты смены состояний процессора и устройства ввода-вывода: прерывание по запросу на ввод-вывод, обработка этого прерывания и пересылка информации из памяти в буфер устройства, вызов драйвера и контроллера, окончание обмена и прерывание контроллера, продолжение вычислений.

Имеются две разновидности режима ввода-вывода – **синхронный** и **асинхронный** .

Синхронный ввод-вывод – это ввод-вывод, выполнение которого приводит к переходу программы в состояние ожидания, до тех пор, пока операция ввода-вывода не будет полностью завершена. На аппаратном уровне – команда ввода-вывода переводит процессор в состояние ожидания (idle) до следующего прерывания. При данном режиме в каждый момент

выполняется не более одного запроса на ввод-вывод; одновременный ввод-вывод отсутствует. Синхронный вывод выполняют всем программистам привычные операторы вида **println(x).** При их использовании в программах мы не задумывается над тем, что используем достаточно неэффективный вариант синхронного ввода-вывода. Однако до сих пор мышление большинства программистов – последовательное, в том смысле, что о своей программе они мыслят как о чисто последовательно выполняемой, и вообще не думают о возможности какого-либо распараллеливания. При отладке программы, либо если размер выводимой информации невелик, это обычно вполне допустимо.

Асинхронный ввод-вывод – ввод-вывод, выполняемый параллельно с выполнением основной программы. После того. как начинается асинхронный ввод-вывод, управление возвращается пользовательской программе, без ожидания завершения ввода-вывода (последнее может быть выполнено специальной явной операцией). Таким образом, операция асинхронного обмена как бы разбивается на две: **начать ввод-вывод** и **закончить ввод-вывод**. Последняя выполняется для того, чтобы в этом месте программа все же ожидала завершения обмена, когда его результат необходим для дальнейших вычислений. Такой подход к реализации обмена более труден для понимания программистами и может привести к ошибкам (например, использована только операция начала ввода-вывода, а вызов операции его окончания забыт).

Таблица состояния устройств

На системном уровне, при обмене происходит следующее. Выполняется **системный вызов (system call)** – запрос к ОС путем вызова системной подпрограммы, в данном случае – чтобы позволить пользователю ожидать завершения ввода-вывода. Операционная система хранит **таблицу состояния устройств**, в которой каждому устройству соответствует элемент, содержащий тип устройства, его адрес и состояние. ОС индексирует таблицу устройств, с целью определения состояния устройства и модификации элемента таблицы для включения в нее информацию о прерывании.

Архитектура синхронного (a) и асинхронного (b) ввода-вывода иллюстрируется на рис. 4.3.

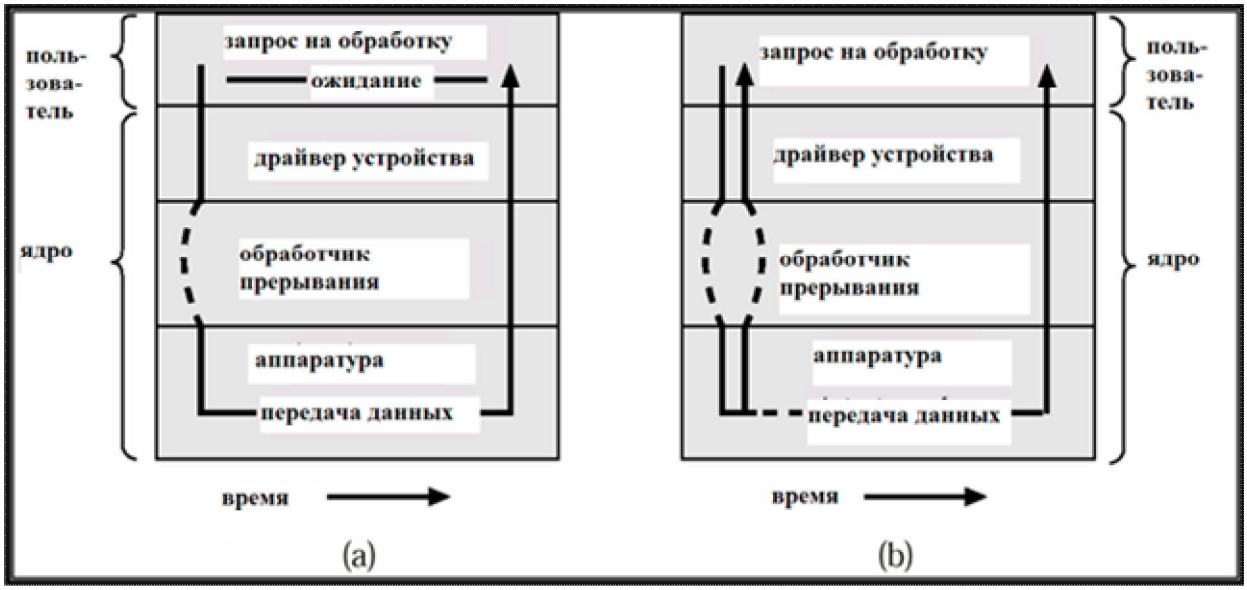

Рис. 4.3. Архитектура синхронного и асинхронного ввода-вывода

На схеме видно, что отличительной чертой синхронного обмена является переход процессора в состояние ожидания до окончания операции ввода-вывода.

На рис. 4.4 показан пример состояния таблицы устройств ввода-вывода, хранимой операционной системой. Для каждого устройства хранится информация о его имени, состоянии, а для занятых устройств – адрес начала и длина порции информации, подлежащей обмену. Если для некоторого устройства (в примере – **диск3**) имеется несколько запросов на

ввод-вывод, все они организуются в очередь и обслуживаются по очереди, по мере освобождения устройства.

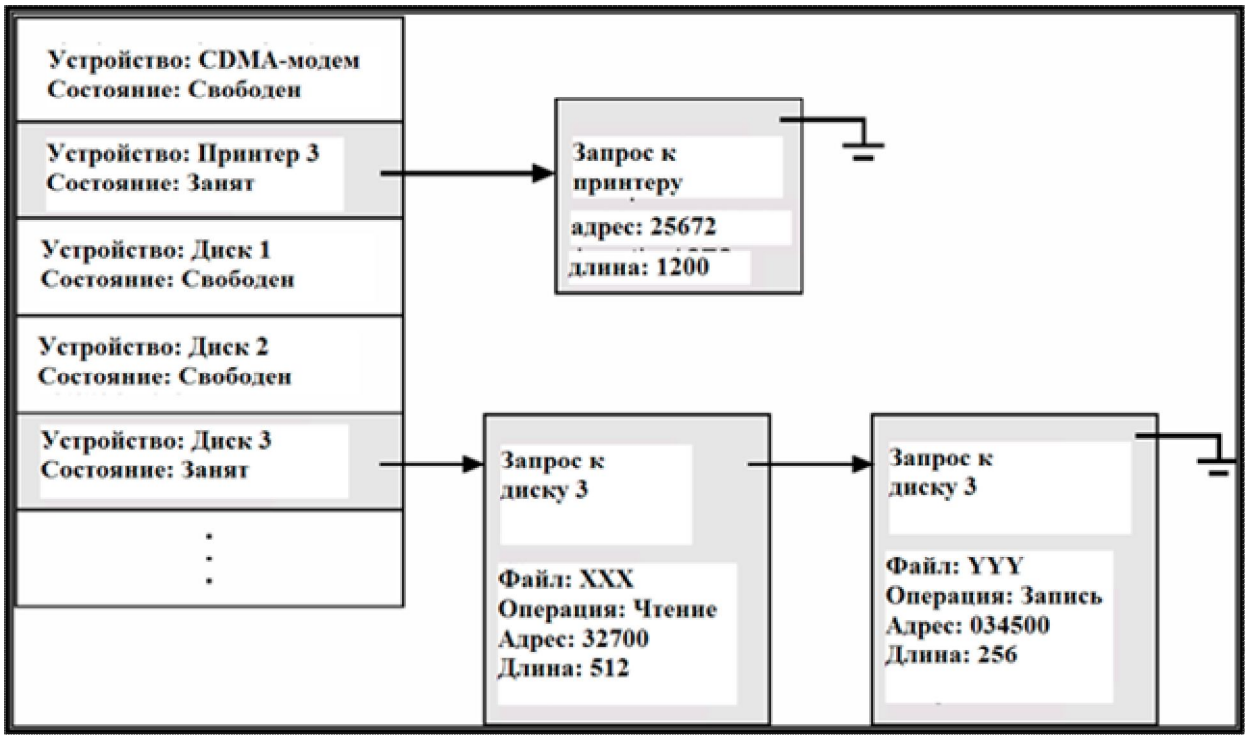

Рис. 4.4. Пример состояния таблицы внешних устройств ОС

6. Лекция: Архитектура компьютерной системы. Прямой доступ к памяти

Прямой доступ к памяти (Direct Memory Access – DMA) – более эффективный метод работы контроллеров устройств ввода-вывода, используемый для работы высокоскоростных устройств, способных передавать информацию со скоростью, близкой к скорости работы памяти

DMA-контроллер передает блок данных из буферной памяти непосредственно в основную память, без участия процессора. Преимущество подобного широко применяемого подхода – не только в том, чтобы избежать лишней пересылки данных из одной области памяти в другую, но также в том, что прерывание в этом случае генерируется на каждый блок пересылаемых данных (хранящийся в буфере), но не на каждый пересылаемый байт, как при более традиционном способе обмена.

Структура памяти

Основная (оперативная) память – единственная крупная часть памяти, к которой процессор имеет непосредственный доступ. Как известно, содержимое основной памяти не сохраняется после перезагрузки системы или после выключения компьютера. **Внешняя (вторичная) память** – расширение основной памяти, обеспечивающее функциональность устойчивой (сохраняемой) памяти большого объема.

В качестве вторичной памяти чаще всего используются **жесткие диски (hard disks).** Физически они состоят из твердых пластин из металла или стекла, покрытых магнитным слоем для записи. Поверхность диска логически делится на **дорожки (tracks),** которые, в свою очередь, делятся на **секторы.** Контроллер диска определяет логику взаимодействия между устройством и компьютером.

Устройство жесткого диска показано на рис. 4.5.

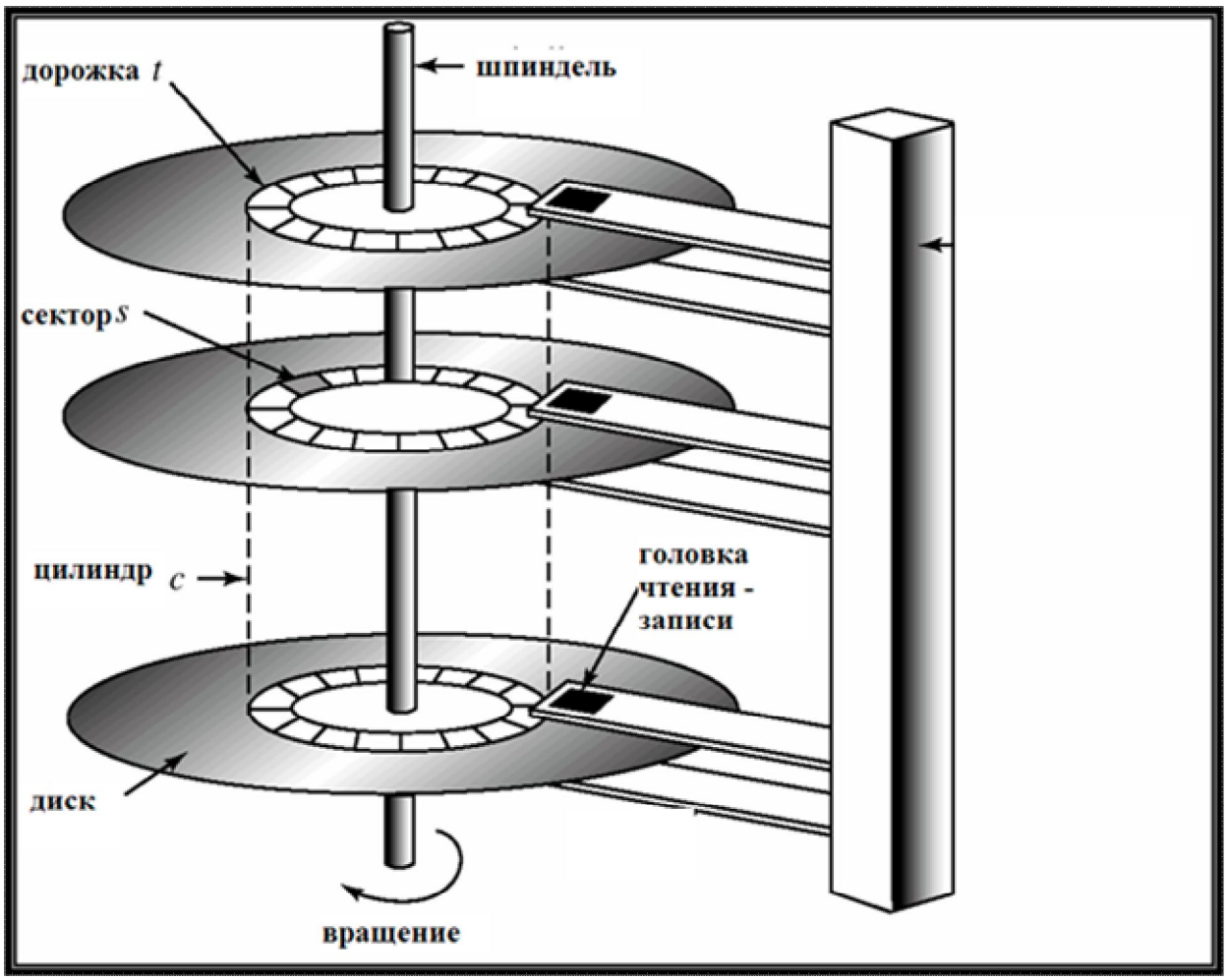

Рис. 4.5. Устройство жесткого диска

Как видно из рисунка, **цилиндр** - это группа вертикально расположенных друг под другом секторов различных магнитных дисков с одним и тем же номером дорожки.

Системы памяти организованы в **иерархию**, исходя из их быстродействия, стоимости и возможности сохранения информации (устойчивости). Для оптимизации работы памяти любого вида используется **ассоциативная память (кэш – cache),** размещаемая в более быстродействующих системах памяти и хранящая наиболее часто используемые элементы более медленной памяти. С этой точки зрения, оперативную память можно рассматривать как кэш для внешней памяти. Кэш-память – это, по сути дела, ассоциативный список пар **(Адрес, Значение),** причем аппаратный поиск в ней происходит по адресу как по ключу. Таким образом, перед обращением к медленной внешней памяти сначала происходит поиск по заданному адресу в кэш-памяти, и только если он не привел к успеху, выполняется стандартное обращение к внешней памяти. Принцип кэширования очень важен и позволяет существенно ускорить работу со внешней памятью. Однако он требует реализации специальной политики управления кэш-памятью, так как кэширование вводит дополнительный уровень в иерархии памяти и требует согласованности данных, хранимых одновременно на разных уровнях памяти. Аппаратура и ОС поддерживают **кэш команд, кэш данных, кэш жесткого диска** и т.д. – для всех видов памяти.

Иерархия устройств памяти (в упрощенном виде) показана на рис. 4.6

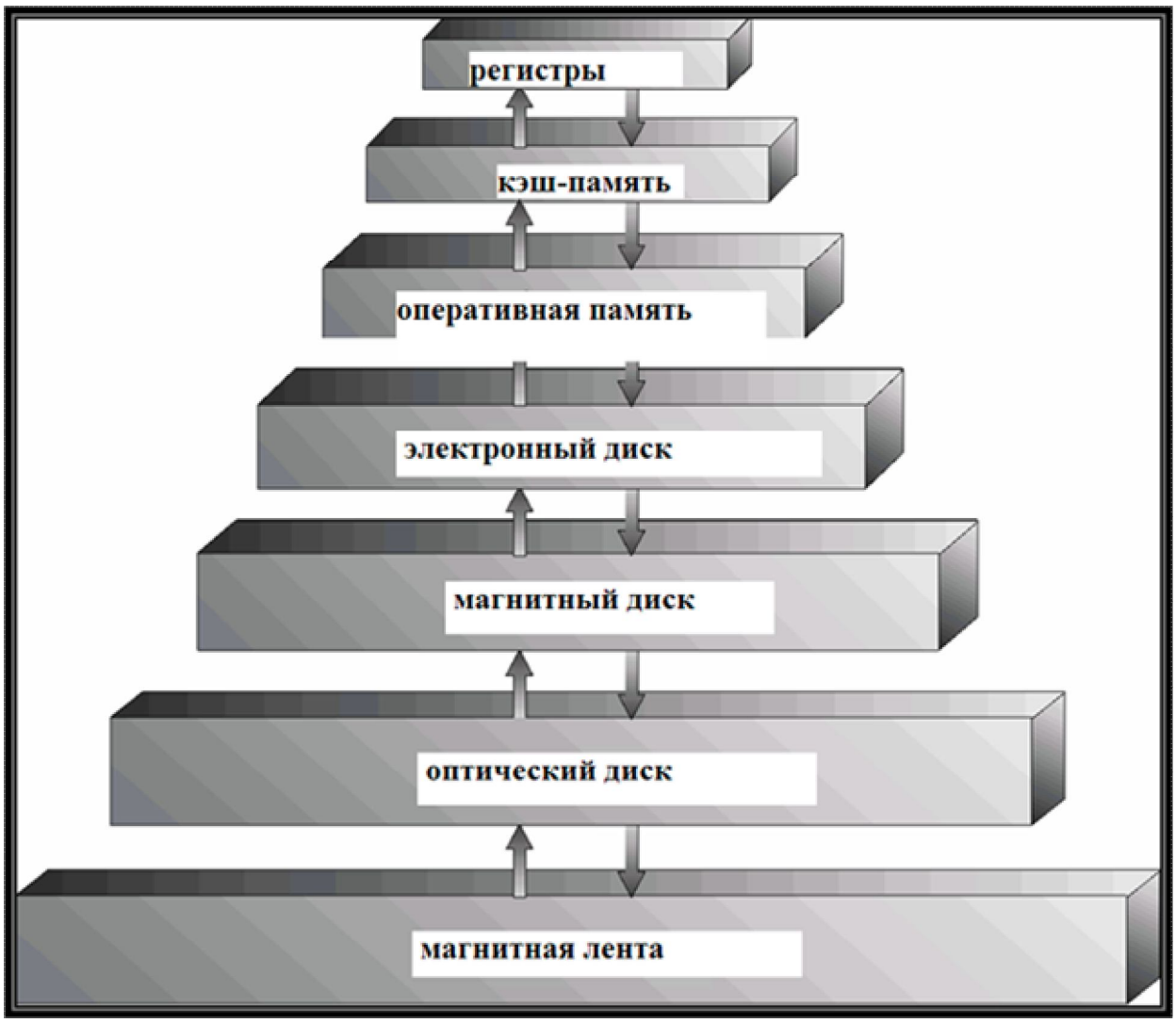

Рис. 4.6. Иерархия устройств памяти

Более быстрые виды памяти на схеме расположены выше, более медленные – ниже. Схема особых комментариев не требует. Некоторые часто используемые виды внешней памяти:

- **флэш-память (флэшка)** – внешняя память компактного размера, модуль которой подключаются через USB-порт. Параметры: объем - до 128 гигабайт и более; скорость обмена через порт USB 2.0: 240 – 260 мегабит в секунду;
- **внешний жесткий диск (ZIV drive** и другие) – объем до 1 терабайта; работает также через порт USB;
- **BluRay – диски** – новая разновидность компакт-дисков большой емкости (односторонние – 25 гигабайт, двусторонние – 50 гигабайт). Для сравнения, стандартная емкость диска DVD составляет 4.7 гигабайт.

Аппаратная защита памяти и процессора

В целях совместного использования системных ресурсов (памяти, процессора, внешних устройств) несколькими программами, требуется, чтобы аппаратура и операционная система обеспечили невозможность влияния некорректно исполняемой программы на другие пользовательские программы. Для этого необходима аппаратная поддержка, как минимум, двух режимов исполнения программ – **пользовательского (непривилегированного) режима (user mode) –** для выполнения программ пользователей – и **системного (привилегированного, режима ядра - system mode, monitor mode)** - для модулей операционной системы. Идея двух режимов в том, чтобы выполняемые в привилегированном режиме модули ОС могли выполнять распределение и выделение системных ресурсов, в

частности, формировать новые адреса, а пользовательские программы, в результате ошибок или преднамеренных атак, выполняясь в обычном режиме, не могли бы обратиться в область памяти операционной системы или другой задачи, изменять их и этим нарушать их целостность. Для определения текущего режима выполнения команд в аппаратуре вводится **бит режима**, равный 0 для системного и 1 – для пользовательского режима. При прерывании или сбое аппаратура автоматически переключается в системный режим. Некоторые привилегированные команды, изменяющие системные ресурсы и состояние системы (например, регистр состояния процессора), должны выполняться только в системном режиме, что защитит системные ресурсы от случайной или преднамеренной порчи при выполнении этих команд обычной пользовательской программой.

Для **защиты ввода-вывода** все команды ввода-вывода считаются привилегированными. Необходимо гарантировать, чтобы пользовательская программа никогда не получила управление в системном режиме и, в частности, не могла бы записать новый адрес в вектор прерываний, который, как уже отмечалось, содержит адреса подпрограмм обработки прерываний, в частности, связанных со вводом-выводом.

Использование системного вызова для выполнения ввода-вывода иллюстрируется на рис. 4.7.

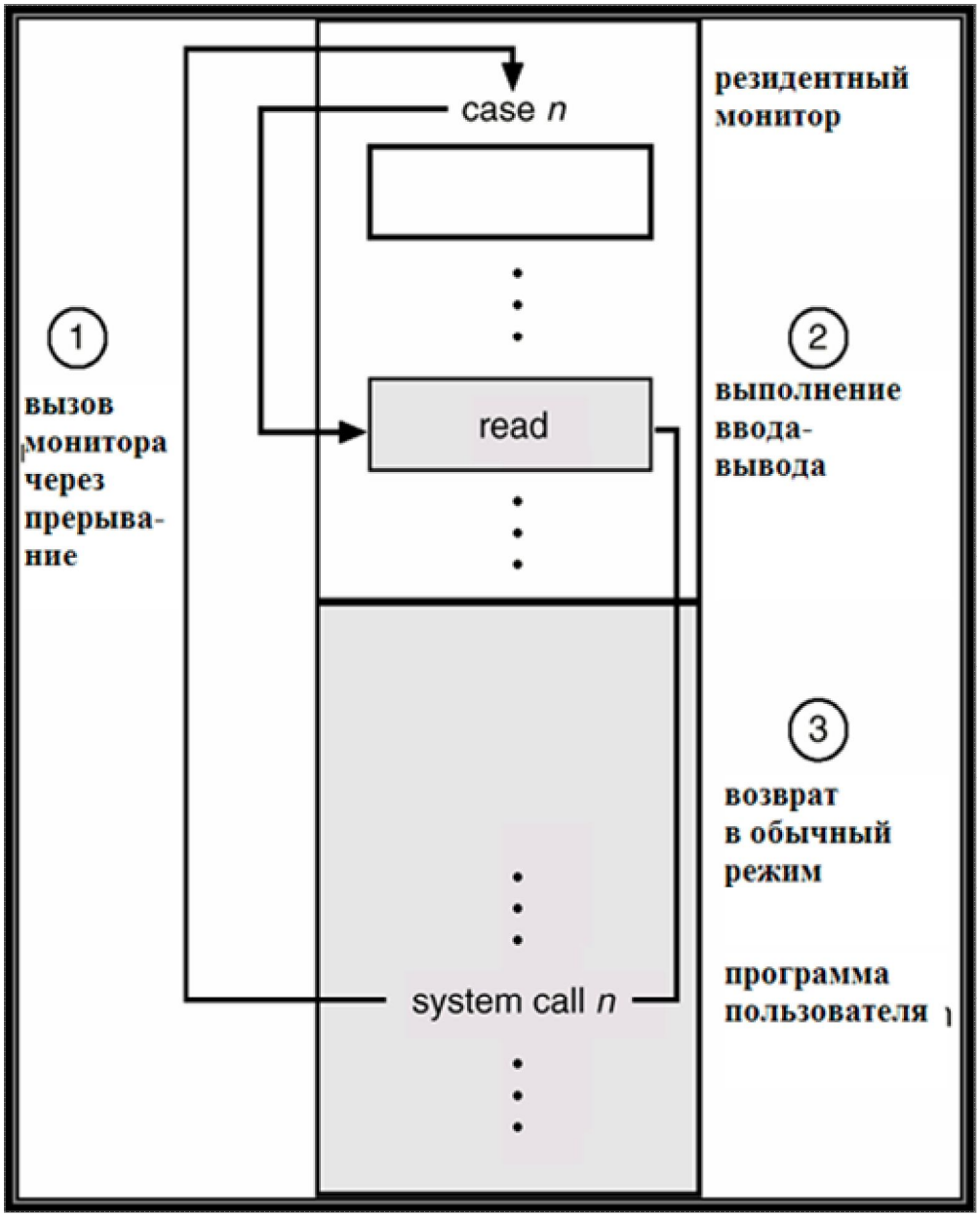

Рис. 4.7. Использование системного вызова для выполнения ввода-вывода.

На схеме системный вызов номер **n** вызывает программируемое прерывание (trap), вызывается ОС в привилегированном режиме, и по номеру системного вызова определяется операция ввода-вывода, которая должна быть выполнена по данному прерыванию. Затем в привилегированном режиме выполняется операция ввода-вывода, после чего происходит прерывание и возврат в пользовательскую программу, выполняемую в обычном режиме.

Для **защиты памяти** необходимо обеспечить защиту, по крайней мере, для вектора прерываний и подпрограмм обслуживания прерываний. Например, недопустимо разрешить пользовательской программе формировать в обычном режиме произвольный адрес и обращаться по нему, так как при этом может быть нарушена целостность системных областей памяти. Чтобы этого избежать, в аппаратуре вводятся два регистра, которые отмечают границы допустимой области памяти, выделенной пользовательской программе. Это **базовый регистр (base register),** содержащий начальный адрес области памяти, выделенной пользовательской программе, и **регистр границы (limit register),** содержащий размер пользовательской области памяти. Память вне отмеченного диапазона считается защищенной, т.е. обращения к ней из пользовательской программы не допускаются (при попытке такого обращения возникает прерывание).

Использование базового регистра и регистра границы иллюстрируется на рис. 4.8.

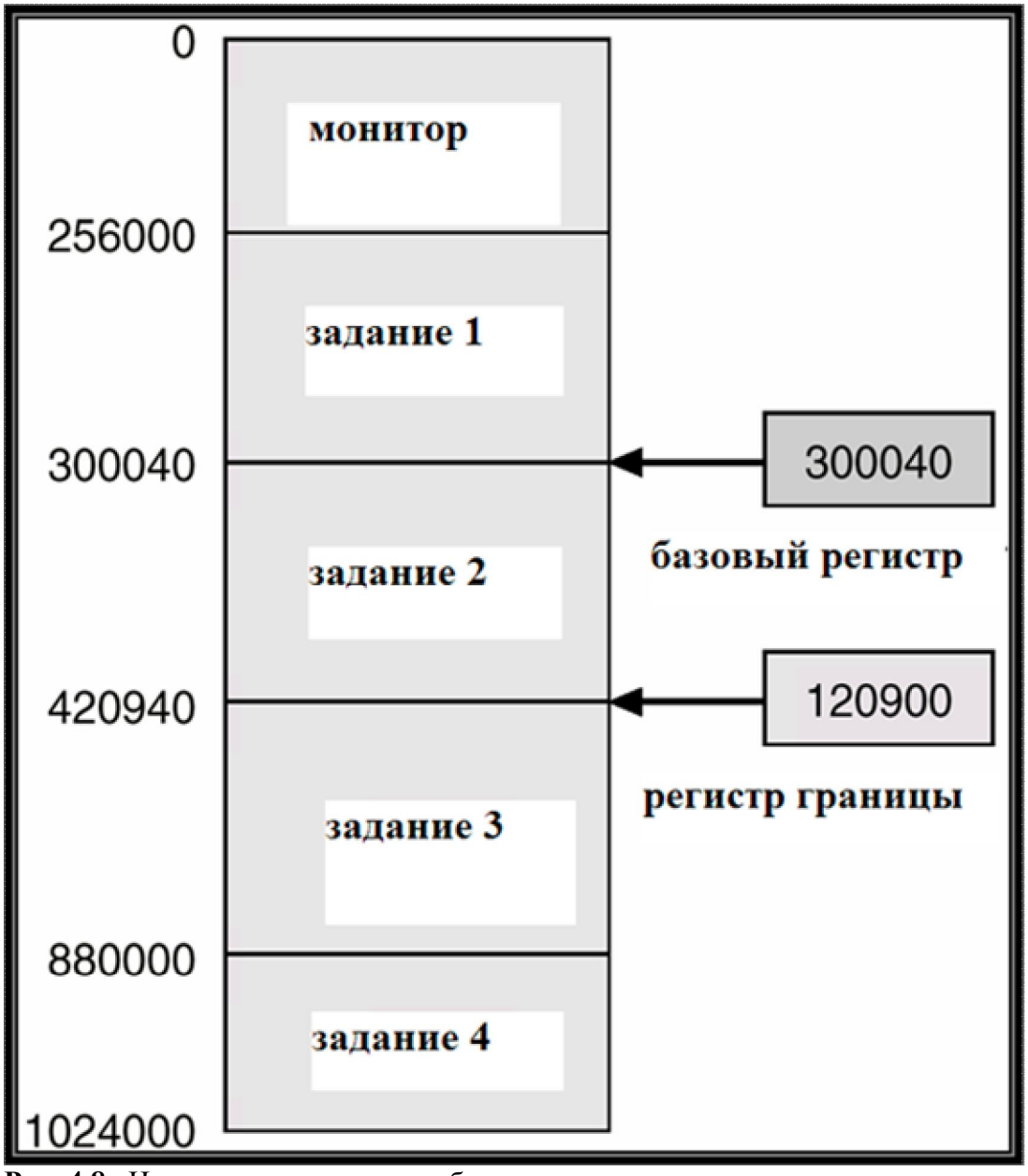

Рис. 4.8. Использование регистра базы и регистра границы для защиты памяти

На схеме заданию 2 выделена область памяти, начиная с адреса 300040 (хранящегося в регистре базы), длиной 120900 (хранящейся в регистре границы), т.е. по адрес 420939 включительно. Обращение, например, по адресу 420940 из программы задания 2 приводит к прерыванию как недопустимое – срабатывает защита памяти.

Схема аппаратной защиты адресов памяти иллюстрируется рис. 4.9.

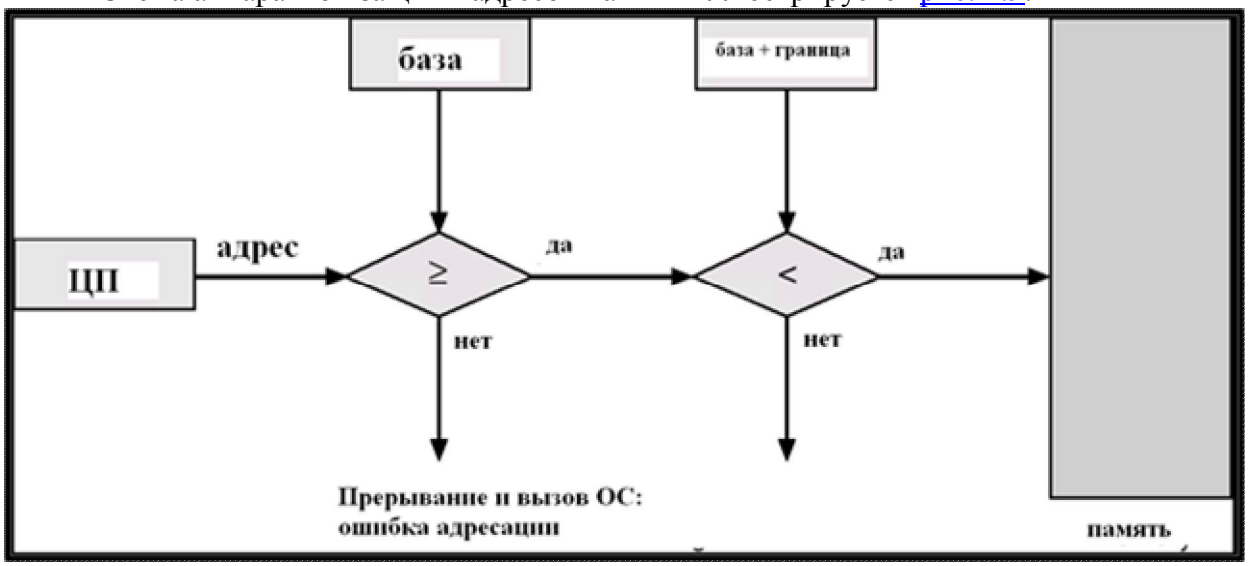

Рис. 4.9. Схема аппаратной защиты адресов памяти

Аппаратная защита адресов памяти в системах с теговой архитектурой

Более радикальные меры для защиты памяти (и не только) предприняты в системах с теговой архитектурой - МВК "Эльбрус", Burroughs 5000/6700/7700 и др.

Как уже пояснялось, в такой компьютерной системе каждое слово памяти имеет **тег** – информацию о типе данных, хранящемся в данном слове. Специальные теги имеют любые данные – например, числа (целые и вещественные), адреса, указатели на процедуры и др. Аппаратура при выполнении команды выполняет динамический контроль типов – проверяет, соответствуют ли теги операндов выполняемой операции. Если не соответствуют – прерывание.

Адрес в системе с теговой архитектурой представлен специальным адресным словом - **дескриптором** (**descriptor**). Кроме тега и собственно адреса начала адресуемого массива в памяти, дескриптор содержит также **длину** массива и 4 бита защиты – **от чтения, от записи, от выполнения** и **от записи адресной информации**. Формирование и изменение дескриптора возможно только средствами ОС в привилегированном режиме. Пользовательская программа не может ни сформировать, ни изменить дескриптор и работает со своей областью памяти как с массивом, защищенным тегом и дескриптором, образовывая от него подмассивы и формируя их дескрипторы (такое действие разрешено). Допустимая операция над массивом - **индексация a[i],** в которой аппаратно проверяется, что индекс **i** не выходит за границы массива **a**. Таким образом, обращение в "чужую" область памяти в такой системе принципиально невозможно. Невозможна также адресная арифметика (в стиле C / C++), так как попытка выполнения арифметической операции над словом с тегом **дескриптор** приводит к немедленному прерыванию.

Кроме дескриптора, имеется также **косвенное слово (indirect word)** – тегированный адрес для обращения к элементу данных одной командой, непосредственно по адресу (без индексации). Для косвенных слов фактически выполняются те же аппаратные проверки, что и для дескрипторов.

Подобная система защиты, с одной стороны, совершенна и стопроцентна, с другой, разумеется, требует больших накладных расходов на аппаратную проверку тегов, которую

отключить невозможно, даже в случаях, если из кода программы очевидно, что никаких ошибок при работе с адресной информации нет.

Организация аппаратной защиты памяти и процессора

Прерывания по таймеру

При исполнении в привилегированном режиме ОС имеет неограниченный доступ как к памяти монитора, так и к памяти пользователя. Команды записи значений в регистры **base** и **limit** являются привилегированными.

В системах с теговой архитектурой только привилегированная команда может сформировать новый дескриптор на область памяти, либо изменить поле в дескрипторе (например, адрес начала или длину).

Для организации периодических прерываний в системе имеется **таймер –** системный регистр, содержащий некоторое установленное специальной командой значение времени, которое уменьшается через каждый квант (такт) процессорного времени. Когда значение таймера становится равным нулю, происходит прерывание. Прерывание по таймеру используется для организации периодического опроса устройств, для реализации режима разделения времени (для отачки неактивных задач по истечении некоторого временного интервала) и для вычисления текущего времени.

Команда записи значения в таймер является привилегированной.

Ключевые термины

Bluetooth – интерфейс для беспроводного подключения (с помощью радиосвязи) к компьютеру мобильных телефонов, органайзеров, наушников, плейеров и многих других видов устройств.

BluRay – диск – разновидность компакт-дисков большой емкости (25 – 50 ГБайт).

COM (communication port, serial port, последовательный порт) – порт для подключения к компьютеру различных коммуникационных устройств, например, модемов.

DMA (Direct Memory Access) – контроллеры с прямым доступом к оперативной памяти, минуя использование специализированной памяти устройства.

EPP (Extended Parallel Port) – двунаправленный режим работы порта **LPT**, в котором он может работать не только для вывода, но и для ввода информации.

HDMI (High Definition Multimedia Interface) – интерфейс и порт. позволяющий подключить к компьютеру телевизор или другое видеооборудование, обеспечивающее наилучшее качество воспроизведения (HD – High Definition).

IEEE 1394 (FireWire) – порт для подключения к компьютеру цифровой видеокамеры или фотоаппарата.

LPT (от line printer), или **параллельный порт** –устаревший вид порта для подключения принтера или сканера, с толстым в сечении кабелем и большим разъемом, требующий предварительного выключения компьютера и устройства для их безопасного соединения.

PCI (Personal Computer Interface) – наиболее распространенный тип системной шины, к которой подсоединены процессор, память, диски, принтер, модем и другие внешние устройства компьютерной системы.

RS-232 - другое (более старое) название порта **COM**.

SCSI (Small Computer System Interface) – интерфейс, адаптеры и порты для подключения широкого спектра внешних устройств – жестких дисков, сканеров и др., с возможностью обслуживания **гирлянды устройств**, подключенных к одному SCSI-порту и имеющих различные номера (**SCSI IDs**).

SCSI ID – номер устройства (от 0 до 9), являющегося частью **гирлянды SCSI-устройств,** подключенных к одному SCSI-порту.

TV-тюнер - устройство для приема телевизионного сигнала с антенны и показа телевизионного изображения на компьютере.

USB (Universal Serial Bus) – наиболее распространенный универсальный порт компьютера (с характерным плоским разъемом, размером порядка 1 см, с изображением

трезубца), к которому могут подключаться клавиатура, мышь, внешний диск, принтер, сканер и другие внешние устройства.

Асинхронный ввод-вывод – ввод-вывод, выполняемый параллельно с выполнением программы пользователя.

Ассоциативная память (кэш – cache) – область памяти, размещаемая в более быстродействующей системе памяти и хранящая наиболее часто используемые элементы более медленной памяти вместе с их адресами, с целью оптимизации обращений к ним.

Базовый регистр (base register) – системный регистр, используемый для защиты памяти и содержащий начальный адрес области памяти, выделенной пользовательской программе.

Бит режима – бит, хранящийся в системном регистре и задающий текущий режим выполнения команд: равен 0 для системного режима и 1 – для пользовательского режима.

Вектор прерываний (interrupt vector) – резидентный массив в оперативной памяти, в котором хранятся доступные по номерам прерываний адреса подпрограмм-обработчиков прерываний (модулей ОС).

Виртуальный COM-порт – воображаемый COM-порт (в действительности не существующий и не имеющий разъема), который ОС как бы инсталлирует в систему при установке, например, драйвера для взаимодействия через Bluetooth или через кабель компьютера с мобильным устройством. Обычно имеет большой номер, например, 18.

Внешняя (вторичная) память – расширение основной памяти, обеспечивающее функциональность устойчивой (сохраняемой) памяти большого объема.

Гирлянда SCSI-устройств – цепочка устройств, подключенных к одному **SCSI-порту** и имеющих различные **SCSI IDs** (номера).

Дескриптор (**descriptor**) – адресное слово в системах с теговой архитектурой; содержит тег дескриптор, **адрес** начала адресуемого массива в памяти, **длину** массива и 4 бита зашиты – **от чтения, от записи, от выполнения** и **от записи адресной информации**.

Дорожка (track) – часть **жесткого диска**, расположенная между двумя концентрическими окружностями на одном из составляющих его магнитных дисков.

Жесткий диск (hard disk) - разновидность **внешней памяти,** физически состоящая из твердых пластин из металла или стекла, покрытых магнитным слоем для записи, шпинделя и головок считывания – записи.

Инфракрасный порт (IrDA) – порт для подключения ноутбука к мобильному телефону (или двух ноутбуков друг к другу) через инфракрасную связь.

Контроллер устройства – специализированный процессор (устройство управления) для какого-либо устройства компьютерной системы - основной памяти или внешнего устройства.

Материнская плата (motherboard) – основная печатная плата компьютера, на которой смонтированы процессор и память.

Модем (аббревиатура от **модулятор – демодулятор**) – устройство для выхода в Интернет и передачи информации по аналоговой или цифровой телефонной линии.

Опрос устройств (polling) – действия операционной системы по периодической проверке состояния всех портов и внешних устройств, которое может меняться с течением времени.

Основная (оперативная) память – быстродействующая память, к которой процессор имеет непосредственный доступ во время выполнения программы, хранящая программы и данные, информация в которой не сохраняется после выключения компьютера или перезапуска системы.

Очередь прерываний – системная структура ОС, обеспечивающая поочередную обработку всех возникших прерываний.

Пользовательский (непривилегированный) режим (user mode) – стандартный режим выполнения программ, в котором исполняются программы пользователей. В данном

режиме запрещены некоторые привилегированные операции (например, изменение системных областей памяти и регистров).

Порт – устройство с разъемом и контроллером для подключения к компьютеру внешних устройств.

Прерывания по таймеру – периодические прерывания через определенный квант времени, предназначенные для **опроса устройств** и других необходимых периодических действий ОС.

Программа, управляемая прерываниями (interrupt-driven program) – программа, запускаемая автоматически при возникновении прерывания центрального процессора (например, операционная система).

Программируемое прерывание (trap; дословно – ловушка) – прерывание, явно генерируемое с помощью специальной команды процессора (обычно для обработки ошибки в программе).

Регистр границы (limit register) – системный регистр, используемый для защиты памяти и содержащий длину области памяти, выделенной пользовательской программе.

Сектор – часть **жесткого диска**, ограниченная **дорожкой** и двумя радиусами.

Синхронный ввод-вывод – операция ввода-вывода, выполнение которой приводит к переходу программы в состояние ожидания, до тех пор, пока операция ввода-вывода не будет полностью завершена.

Системная шина (system bus) – коммуникационное устройство, соединяющее между собой все модули компьютерной системы - центральный процессор, память и контроллер памяти, внешние устройства и их контроллеры, - которые через системную шину обмениваются сигналами.

Системный вызов (system call) – явный запрос пользовательской программы к ОС путем вызова системной подпрограммы.

Системный (привилегированный) режим (system mode, kernel mode, monitor mode) – особый режим выполнения команд, в котором исполняются модули ядра ОС, допускающий выполнения ряда привилегированных операций, например, изменение системных областей памяти и регистров.

Состояние процессора – значения регистров и значение **счетчика команд** .

Счетчик команд – адрес текущей выполняемой или прерванной команды процессора.

Таблица состояния устройств – таблица, хранимая и используемая операционной системой, в которой каждому устройству соответствует элемент, содержащий тип устройства, его адрес и состояние, а для занятого устройства – ссылку на очередь обрабатываемых запросов к нему.

Таймер – системный регистр, содержащий некоторое установленное специальной командой значение времени, которое уменьшается через каждый квант (такт) процессорного времени. Когда значение таймера становится равным нулю, происходит прерывание.

Флэш-память (флэшка) – модуль внешней памяти компактного размера (как правило, – 5 см), подключаемый через USB-порт и имеющий емкость до 128 ГБайт.

Цилиндр – часть **жесткого диска**, представляющая собой совокупность **дорожек** одного диаметра, находящихся на всех его параллельно расположенных магнитных дисках.

Краткие итоги

Компьютерная система состоит из модулей – процессора, памяти и внешних устройств, каждое из которых управляется своим контроллером, соединенных между собой системной шиной. В современных компьютерных системах имеются такие модули, как процессор, память, общая шина PCI, порты – USB, COM, IEEE 1394, SCSI, HDMI и другие,. SCSI-порт допускает подключение к нему гирлянды устройств. Инфракрасный порт (IrDA) неудобен и фактически устарел. Беспроводной интерфейс Bluetooth используется для связи компьютера с мобильным устройством, наушниками, плеером.

Модули компьютерной системы – процессор, память и внешние устройства с их контроллерами – функционируют параллельно. Контроллер имеет локальный буфер, через

который осуществляется обмен с устройством. Оптимизация – режим DMA, при котором роль буферной памяти играет часть оперативной памяти. При необходимости выполнения ввода-вывода процессор информирует систему об этом через прерывание. По окончании операции контроллер также генерирует прерывание.

Обработка прерываний осуществляется через резидентный вектор прерываний, содержащий адреса подпрограмм обработки прерываний – модулей ОС. ОС – это фактически программа, управляемая прерываниями. Она вызывается либо по прерыванию, либо по программируемому прерыванию (ловушке), либо системным вызовом подпрограммы ОС из программы пользователя. В системе имеется очередь прерываний, с помощью которой обрабатывается последовательно вся цепочка возникающих прерываний. При прерывании ОС сохраняет состояние процессора, обработчик прерывания определяет, какого типа прерывание произошло и какие действия следует предпринять по его обработке. Возможны прерывания по таймеру с целью периодического опроса устройств.

Ввод-вывод может быть синхронным и асинхронным. Для обработки ввода-вывода ОС хранит и использует таблицу состояния устройств.

Устройства памяти имеют свою иерархию, от самых быстрых к наиболее медленным. Для оптимизации обращения к более медленной памяти используется ассоциативная память (кэш), организуемый в более быстрой памяти. Наиболее распространенные виды внешней памяти – жесткие диски, а также флэш-память, CD, DVD и BluRay – диски.

Для защиты памяти и всей системы вводятся два режима исполнения – привилегированный (для ядра ОС) и непривилегированный (для обычных программ). Для защиты памяти используются два регистра – базы и границы, задающие границы области памяти, выделенной пользовательской программе.Все команды ввода-вывода – привилегированные. Бит режима задает текущий режим выполнения. Ввод-вывод реализуется с помощью системных вызовов.

В системах с теговой архитектурой защита памяти осуществляется в помощью адресных слов со специальными тегами – дескрипторов. Дескриптор содержит адрес начала массива, длину и признаки защиты.

Прерывания по таймеру организуются системой для опроса устройств и для реализации режима разделения времени.

7. Лекция: Архитектура ОС. Управление процессами: Основные понятия. Семафоры и мониторы .

В лекции рассматриваются: архитектура ОС и ее функциональность; управление процессами как основная функция ОС; обзор базовых механизмов синхронизации процессов - семафоров и мониторов.

Введение

В данной и следующей лекциях рассмотрена архитектура ОС. Будут рассмотрены следующие вопросы:

1. Компоненты системы
2. Сервисы (службы) системы
3. Системные вызовы
4. Системные программы
5. Структура системы
6. Виртуальные машины
7. Проектирование и реализация системы
8. Генерация системы.

Основные компоненты ОС

Операционная система – весьма сложная по архитектуре программная система, в которой можно выделить следующие основные компоненты:

1. Управление процессами

2. Управление основной памятью
3. Управление файлами
4. Управление системой ввода-вывода
5. Управление внешней памятью
6. Поддержка сетей (networking)
7. Система защиты (protection)
8. Система поддержки командного интерпретатора.
9. Графическая оболочка.

Рассмотрим эти компоненты подробнее.

Управление процессами. Процесс – это программа пользователя в ходе ее выполнения в компьютерной системе. ОС управляет работой процессов, их распределением по процессорам и ядрам системы, порядком их выполнения и размещения в памяти, их синхронизацией при параллельном решении частей одной и той же задачи разными процессами.

Управление основной памятью. Основная (оперативная) память может рассматриваться как большой массив. Операционная система распределяет ресурсы памяти между процессами, выделяет память по запросу, освобождает ее при явном запросе или по окончании процесса, хранит списки занятой и свободной памяти в системе.

Управление файлами. Файл – это логическая единица размещения информации на внешнем устройстве, например, на диске. ОС организует работу пользовательских программ с файлами, создает файлы, выполняет их открытие и закрытие и операции над ними (чтение и запись), хранит ссылки на файлы в директориях (папках) и обеспечивает их поиск по символьным именам.

Управление системой ввода-вывода. Как уже отмечалось, в компьютерной системе имеется большое число внешних устройств (принтеры, сканеры, устройства управления компакт-дисками и др.), управляемых специальными контроллерами (спецпроцессорами) и драйверами – низкоуровневыми программами управления устройствами, выполняемыми в привилегированном режиме. ОС управляет всеми этими аппаратными и программными компонентами, обеспечивая надежность работы внешних устройств, эффективность их использования, диагностику и реконфигурацию в случае их сбоев и отказов. Для этого ОС хранит и использует таблицу состояния устройств.

Управление внешней памятью. Как уже говорилось, внешняя (вторичная) память – это расширение оперативной памяти процессора более медленными, но более емкими и постоянно хранящими информацию видами памяти (диски, ленты и др.). При управлении внешней памятью ОС решает задачи, аналогичные задачам управления основной памятью, - выделение памяти по запросу, освобождение памяти, хранение списков свободной и занятой памяти и др. ОС поддерживает также использование ассоциативной памяти (кэш-памяти) для оптимизации обращения ко внешней памяти.

Поддержка сетей. Как неоднократно подчеркивалось, любая современная компьютерная система постоянно или временно находится в различных локальных и глобальных сетях. Операционная система обеспечивает использование сетевого оборудования (сетевых карт, или адаптеров), вызов соответствующих драйверов, поддержку удаленного взаимодействия с файловыми системами, находящимися на компьютерах сети, удаленный вход на другие компьютеры сети и использование их вычислительных ресурсов, отправку и получение сообщений по сети, защиту от сетевых атак.

Система защиты. Согласно современным принципам надежных и безопасных вычислений, при работе ОС должны быть обеспечены надежность и безопасность, т.е. защита от внешних атак, конфиденциальность личной и корпоративной информации, диагностика и исправления ошибок и неисправностей и др. ОС обеспечивает защиту компонент компьютерной системы, данных и программ, поддерживает фильтрацию сетевых пакетов, обнаружение и предотвращение внешних атак, хранит информацию обо всех действиях над системными структурами, полезную для анализа атак и борьбы с ними.

Система поддержки командного интерпретатора. Любая операционная система поддерживает командный язык (или набор командных языков), состоящих из пользовательских команд, выполняемых с пользовательского терминала (из пользовательской консоли). Типичные команды – это получение информации об окружении, установка и смена текущей рабочей директории, пересылка файлов, компиляция и выполнение программ, получение информации о состоянии системы и выполнении своих процессов и др. В системе Windows для выполнения команд по традиции используется окно пользовательской консоли MS DOS (MS DOS Prompt), в системе Linux – специальное окно "Терминал" (Start / System Tools / Terminal). Наиболее мощные командные процессоры имеются в системах типа UNIX (UNIX, Solaris, Linux и др.). Их командные языки позволяют писать **скрипты** – командные файлы, содержащие часто используемые последовательности команд ОС. В UNIX это наиболее удобно. Можно назвать такие командные языки UNIX, как **sh (Bourne Shell), csh (C shell), ksh (Korn shell), bash.** Каждый UNIX-программист имеет смой излюбленный командный язык и привыкает постоянно использовать скрипты и длинные нетривиальные последовательности команд, которые он выполняет с терминала. Что касается Windows, сравнительно недавно в ней появился мощный командный интерпретатор **PowerShell,** который и рекомендуется к использованию. Кроме того, для Windows имеется система **CygWin,** позволяющая выполнять команды и командные файлы UNIX в среде Windows. Типичная последовательность команд в стиле UNIX: ps –a | grep saf, которая выводит в стандартный вывод информацию об активных процессах, причем только принадлежащих пользователю **saf.** Вертикальная черта (p1 | p2) обозначает операцию **конвейер (pipe),** позволяющую использовать стандартный вывод процесса p1 как стандартный ввод процесса p2, что и используется операцией grep (фильтрация строк, содержащих заданную последовательность). Подробнее о UNIX (Linux) можно прочитать в книге [16].

Графическая оболочка – подсистема ОС, реализующая графический пользовательский интерфейс пользователей и системных администраторов с операционной системой. Разумеется, использование одного лишь командного языка и системных вызовов неудобно, поэтому простой и наглядный графический пользовательский интерфейс с ОС необходим. Имеется много известных графических оболочек для операционных систем, причем их возможности очень похожи друг на друга - настолько, что подчас не вполне понятно, какая именно ОС используется. Среди графических оболочек, используемых в системах типа UNIX, можно назвать CDE, KDE, GNOME. ОС Windows и MacOS имеют собственные, весьма удобные графические оболочки.

Управление процессами

Процесс (process) - это пользовательская программа при ее исполнении в компьютерной системе. Для выполнения процесса требуется ряд **ресурсов**, включая время процессора, память, файлы, устройства ввода-вывода, сетевые устройства и др.

В классической схеме UNIX, при создании процесса для него создается новое пространство виртуальной памяти, т.е. таблица страниц для отображения виртуальных адресов в физические, своя для каждого нового процесса. При этом расходуются значительные ресурсы. Если учесть, что в UNIX каждая команда пользователя (например, ls – вывод содержимого текущей директории) запускается как **отдельный процесс**, то становится понятным, насколько "дорога" операция создания процесса в классическом смысле. Поэтому еще в 1980-х гг. появилась концепция **облегченного процесса** (lightweight process) – выполняемого в том же пространстве виртуальной памяти, что и процесс-родитель. При создании нового облегченного процесса ОС создает для него только **стек** – системный резидентный массив в памяти, предназначенный для поддержки выполнения процедур процесса и хранящий их локальные данные и связующую информацию между ними.

ОС отвечает за следующие действия, связанные с управлением процессами:

Создание и удаление процессов. При создании процесса необходимо создать в памяти соответствующие системные структуры (таблицу страниц, стек и др.). При удалении процесса память, занимаемая ими, освобождается, а также выполняется закрытие всех файлов и

освобождение всех других ресурсов, которые использовал процесс, если последний не сделал этого явно.

Приостановка и возобновление процессов. Выполнение процесса приостанавливается при выполнении синхронного ввода-вывода, а также системного вызова или команды (типа **suspend**). Сразу отметим, что использовать подобные операции явной приостановки процессов следует с осторожностью, так как приостанавливаемый процесс может находиться в своей **критической секции** – выполнять обработку общего ресурса, к которому каждому процессу предоставляется монопольный доступ, так что при его приостановке возникает ситуация **тупика (deadlock**) – приостановленный процесс не может освободить ресурс, а конкурирующий процесс не может его получить. При приостановке процесса ОС сохраняет состояние его выполнения, а при возобновлении – восстанавливает.

Синхронизация процессов. Процессы работают параллельно и при этом конкурируют за общие ресурсы, а также должны в некоторые моменты вычислений ожидать наступления некоторых событий. Для предотвращения возможных конфликтов и несогласованностей, например, **race condition** - несогласованного доступа к общим данным, при котором один процесс читает старые данные, а другой их в этот же момент обновляет, - ОС предоставляет средства **синхронизации** (например, **семафоры** и **мониторы**, рассмотренные в следующем разделе).

Взаимодействие процессов. При своей параллельной работе процессам необходимо взаимодействие, с целью согласованного решения различных частей одной и той же задачи. Процессы могут взаимодействовать с помощью передачи **сообщений** друг другу, а также с помощью так называемых **условных переменных** и **рандеву** (все эти виды взаимодействия рассмотрены позже). ОС предоставляет все эти средства, в виде системных вызовов, для организации адекватного и удобного взаимодействия процессов.

Семафоры. В 1966 г. в работе [17] проф. Эдсгер Дейкстра предложил новый способ синхронизации процессов, ставший классическим, - семафоры.

Двоичный семафор (binary semaphore) – переменная S, которая может находиться в двух состояниях: "открыт" и "закрыт"; над S определены две операции ("семафорные скобки"): P(S) – закрыть, V(S) – открыть. При попытке закрыть уже закрытый семафор происходит прерывание, и ОС добавляет текущий процесс в очередь к закрытому семафору. Операция V(S) активизирует первый стоящий в очереди к S процесс, который успешно завершает операцию P(S). Если семафор S уже открыт, операция V(S) не имеет никакого эффекта.

Таким образом, если предположить, что аппаратура и ОС поддерживают подобную концепцию семафора, то она является удобным инструментом для синхронизации по ресурсам. Назовем **критической секцией** код, который может выполняться несколькими процессами параллельно и осуществляет доступ к некоторому общему для всех процессов ресурсу – глобальной области памяти, общему файлу и т.д. Обозначим код критической секции **critical_section.** Если допустить, что данный код может выполняться параллельно в нескольких процессах напрямую, то может возникнуть уже известная нам ситуация **race condition (**конкуренция за общие данные): один процесс может изменять ресурс, а второй в этот момент считывать его (некорректное) состояние, либо два процесса одновременно будут пытаться изменять однин и тот же ресурс, что приведет к нарушению его целостности. Таким образом, для критических секций необходимо решить задачу **взаимного исключения (mutual exclusion)** – в каждый момент времени не более чем один из параллельных процессов может выполнять критическую секцию. С помощью семафоров Дейкстры эта задача решается легко и изящно: код критической секции должен иметь вид

P(S); critical_section; V(S);

В самом деле, предположим, что несколько процессов выполняют данный код. Первый из них, который начал выполнять операцию P(S), закрывает семафор S и получает доступ к критической секции. Все остальные процессы, которые пытаются выполнить операцию P(S) над закрытым семафором S, прерываются и попадают в очередь к закрытому семафору. Когда

первый процесс закончил работу с ресурсом, он открывает семафор S операцией V(S) для первого процесса из очереди, который, выполнив P(S), вновь закрывает семафор, и т.д.

Очень важное свойство операций P и V в следующем: они **атомарны (atomic)** для других процессов, т.е. если процесс начал выполнять операцию P(S) или V(S), то никакой другой процесс до ее завершения не может также начать выполнять аналогичную операцию.

Подведем итог: для синхронизации процессов по общему ресурсу необходимы взаимное исключение выполнения критических секций и атомарность операций синхронизации.

Однако следует заметить, что использование семафоров – далеко не идеальный способ синхронизации, с точки зрения надежности. При их неаккуратном использовании возможна ситуация **тупика (взаимной блокировки, deadlock**), при которой образуется цепочка процессов, бесконечно ждущих друг друга. Простейший способ создать deadlock – использовать **два** семафора S1 и S2, так, что первый параллельный процесс пытается выполнить код **P(S1); P(S2),** а второй – код **P(S2); P(S1).** Очевидно, что при любом соотношении времен выполнения операций будут закрыты оба семафора, на которых и будут "висеть" оба процесса, не в состоянии двинуться дальше. Как же избежать подобных ситуаций? Ведь ни компилятор, ни операционная система не подскажут программисту правильный способ использования семафоров. Очень легко также "забыть" вызов V(S) и, тем самым, сделать общий ресурс "навеки" недоступным для других процессов. Один из способов решения этой задачи заключается в том, чтобы использовать специальные инструменты и технологии, автоматически обеспечивающие "правильную" последовательность применения операций над семафорами. Один из таких инструментов – **аспектно-ориентированное программирование** [19].

Мониторы – еще один, более надежный способ синхронизации, предложенный в 1974 г. одним из классиков компьютерных наук профессором Чарльзом Хоаром [18].

Монитор – многовходовый модуль M, в котором определены общие для процессов данные D (скрытые) и (абстрактные) операции P1, … PN над этими данными (в виде процедур).

В каждый момент не более чем один из параллельных процессов может вызвать какую-либо из операций: M.Pi (X, Y, …)

Вызов каждой операции монитора – атомарен (как и операции над семафором).

Монитор – еще один удобный механизм синхронизации процессов по ресурсам. Он более надежен, чем семафоры, поскольку вызов операции монитора автоматически обеспечивает разблокировку ресурса после завершения вызова.

Мониторы включены Ч. Хоаром в разработанный им язык Concurrent Pascal для параллельного программирования и разработки операционных систем.

Подробнее о семафорах и мониторах – в специальных разделах курса, посвященных управлению процессами и синхронизации процессов.

Ключевые термины

Race condition - несогласованный доступ из параллельных процессов к общим данным.

Атомарная (atomic) операция – операция, такая, что, если один из параллельных процессов начал ее выполнять, никакой другой процесс до ее завершения не может также начать выполнять эту же операцию над теми же данными.

Графическая оболочка – подсистема ОС, реализующая графический пользовательский интерфейс пользователей и системных администраторов с операционной системой.

Взаимное исключение (mutual exclusion) – режим выполнения **критической секции**, в котором в каждый момент времени ее может выполнять не более чем один из параллельных процессов.

Двоичный семафор (binary semaphore) – системная переменная, над которой определены операции открытия и закрытия, обеспечивающая в закрытом состоянии

прерывание процесса, пытающегося ее закрыть, и добавление его к очереди к закрытому семафору; используется для синхронизации процессов по общим ресурсам.

Конвейер (pipe) – конструкция командных языков (shell) системы UNIX, позволяющая использовать стандартный вывод процесса – первого аргумента как стандартный ввод процесса – второго аргумента.

Критическая секция - код, который может выполняться несколькими процессами параллельно и осуществляет доступ к некоторому общему для всех процессов ресурсу – например, глобальной области памяти или общему файлу.

Монитор (как средство синхронизации) – многовходовый модуль в котором определены общие для параллельных процесов данные и набор операций (в виде процедур) над ними, таких, что в каждый момент времени не более чем один из параллельных процессов может выполнять какую-либо операцию монитора.

Облегченный процесс (lightweight process) – процесс, выполняемый в том же пространстве виртуальной памяти, что и процесс-родитель.

Процесс (process) - пользовательская программа при ее исполнении в компьютерной системе.

Скрипт (script) – командный файл, содержащий часто используемые последовательности команд ОС.

Стек – системный резидентный массив в памяти, создаваемый операционной системой для поддержки выполнения процедур некоторого процесса и хранящий их локальные данные и связующую информацию между ними.

Тупик (взаимная блокировка, deadlock) – ситуация, при которой образуется циклическая цепочка блокированных процессов, бесконечно ждущих друг друга.

Краткие итоги

В данной и следующей лекциях рассмотрена архитектура операционных систем, включая следующие вопросы: компоненты системы; сервисы (службы) системы; системные вызовы; системные программы; структура системы; виртуальные машины; проектирование и реализация системы; генерация системы.

Основные компоненты ОС следующие: управление процессами; управление основной памятью; управление файлами; управление системой ввода-вывода; управление внешней памятью; поддержка сетей; система защиты; система поддержки командного интерпретатора; графическая оболочка.

Процесс – программа пользователя при ее исполнении. ОС поддерживает средства создания, удаления, синхронизации, приостановки и возобновления, взаимодействия процессов. Облегченный процесс исполняется в том же пространстве виртуальной памяти, что и процесс-родитель.

Классические средства синхронизации процессов – семафоры и мониторы. Семафор может находиться в открытом и закрытом состояниях и в закрытом состоянии блокирует все, кроме одного, процессы, которым требуется доступ к общему ресурсу. Монитор содержит описание общих для процессов данных и операций над ними, таких, что в каждый момент не более чем один параллельный процесс может выполнять какую-либо операцию монитора.

8. Лекция: Обзор функций ОС: управление памятью, файлами, процессами, сетями, командными интерпретаторами, сервисы ОС, системные вызовы. Уровни абстракции ОС. Архитектура UNIX и MS-DOS

В лекции рассмотрены: обзор функциональности ОС: управление памятью, файлами, процессами, сетями, командными интерпретаторами, сервисы ОС, системные вызовы; организация ОС по принципу уровней абстракции; особенности архитектуры UNIX и MS-DOS.

Введение

В данной лекции мы продолжаем обзор основной функциональности операционной системы. Рассмотрены также архитектура MS-DOS и UNIX и подход к разработке операционных систем на основе уровней абстракции.

Управление основной памятью

Основную (оперативную) память компьютерной системы можно рассматривать как большой массив слов или байтов, каждый из которых имеет свой адрес. Память - это хранилище данных с быстрым доступом, совместно используемое процессором и устройствами ввода-вывода.

Следует иметь в виду важную особенность основной памяти. В компьютерных архитектурах имеется два различных способа нумерации байтов в слове. По традиции будем представлять себе память как линейный массив, расположенный "слева направо", такой, что адреса слов, находящихся левее, меньше, чем адреса слов, находящихся правее. Каждое слово делится на байты, имеющие в слове свои номера – 0, 1 и т.д.. Например, в 64-разрядных системах в слове 8 байтов, с номерами от 0 до 7, в более старых 16-разрядных (x86) – два байта, с номерами 0 и 1. Если нумерация байтов в слове начинается слева, т.е. начиная со старших битов, то такую архитектуру принято называть **big endian**, если же справа, т.е. начиная с младших битов, то **little endian.** Например, при big endian – архитектуре 32-разрядного процессора байты двух соседних слов памяти нумеруются так: **0, 1, 2, 3, 0, 1, 2, 3.** При little endian же архитектуре нумерация будет иной: **3, 2, 1, 0, 3, 2, 1, 0.** Представим теперь, что мы хотим рассматривать эти же два слова как массив байтов длиной 8 и записать туда байт за байтом символы строки: **"ЭТОТЕКСТ"** (всего – 8 символов). Такая операция при обеих архитектурах будет выполнена одинаково, т.е. последовательные байты получат именно эти значения. Затем рассмотрим результат снова, но уже как последовательность их двух слов. Каково будет содержимое этих слов? При big endian – архитектуре сюрпризов не будет: первое слово – "**ЭТОТ",** второе **"ЕКСТ".** Однако при little endian – архитектуре результат будет совсем иным: первое слово – **"ТОТЭ",** второе – "**ТСКЕ" !** Не забудем, что при обработке целого слова в little endian – архитектуре байты как бы "переставляются" в обратном порядке. Разумеется, это неудобно. С подобной проблемой автор столкнулся при переносе написанного им компилятора с архитектуры SPARC (big endian) на архитектуру Intel x86 (little endian), используя типы **byte** и **word** на Турбо-Паскале. Подобная операция типична для системных программ, например, таблица идентификаторов в компиляторе должна содержать как символы идентификатора (последовательность байтов), так и другую информацию о нем (длину, ссылки в различные таблицы и т.д.). Поэтому при little endian – архитектуре приходится хранить и обрабатывать байтовые массивы и массивы слов отдельно, и нельзя изменять точку зрения на одну и ту же область памяти и рассматривать ее то как массив байтов, то как массив слов.

Пример little endian – архитектуры – x86. Пример big endian – архитектуры – SPARC. При программировании на языках высокого уровня разработчику, как правило, не приходится учитывать это различие. Однако если при реализации распределения памяти требуется одну и ту же область памяти рассматривать то как массив слов, то как массив байтов, то для little endian – архитектур могут быть "сюрпризы", связанные с тем, что при записи в память как в массив слов байты как бы переставляются.

Основная память – это **неустойчивое** (**volatile**) устройство памяти. Ее содержимое теряется при сбое системы или при выключении питания. Для организации устойчивой памяти используются другие, более медленные технологии.

ОС отвечает за следующие действия, связанные с управлением памятью:

- **Отслеживание того, какие части памяти в данный момент используются и какими процессами**. Как правило, ОС организует для каждого процесса свою **виртуальную память** – расширение основной памяти путем хранения ее образа на диске и организации подкачки в основную память фрагментов (страниц или сегментов) виртуальной памяти процесса и ее откачки по мере необходимости.

- **Стратегия загрузки процессов в основную память, по мере ее освобождения**.

При активизации процесса и его запуске или продолжении его выполнения процесс должен быть загружен в основную память, что и осуществляется операционной системой. При этом, возможно, какие-либо не активные в данный момент процессы приходится откачивать на диск.

- **Выделение и освобождение памяти по мере необходимости**. ОС обслуживает запросы вида "выделить область основной памяти длиной **n** байтов" и "освободить область памяти, начинающуюся с заданного адреса, длиной **m** байтов". Длина участков выделяемой и освобождаемой памяти может быть различной. ОС хранит список занятой и свободной памяти. При интенсивном использовании памяти может возникнуть ее **фрагментация** – дробление на мелкие свободные части, вследствие того, что при запросах на выделение памяти длина найденного сегмента оказывается немного больше, чем требуется, и остаток сохраняется в списке свободной памяти как область небольшого размера (подчас всего 1 – 2 слова). В курсе рассмотрены различные стратегии управления памятью и борьбы с фрагментацией. При исчерпании основной памяти ОС выполняет **сборку мусора** – поиск не используемых фрагментов, на которые потеряны ссылки, и **уплотнение (компактировку)** памяти – сдвиг всех используемых фрагментов по меньшим адресам, с корректировкой всех адресов.

Управление файлами

Файл (file) – совокупность логически взаимосвязанной информации, расположенная во внешней памяти. Как правило, файлы представляют программы (в виде исходного текста или в двоичной форме) или данные.

Другой термин, использованный для обозначения файлов фирмой IBM в ее операционной системе – IBM 360/370, - **набор данных (data set).**

ОС отвечает за следующие действия, связанные с управлением файлами.

Создание и удаление файлов. **Отображение файлов на внешнюю память.** ОС выделяет внешнюю память при создании нового файла. Файл в большинстве файловых систем состоит из **заголовка** и **памяти**. В заголовке хранятся **атрибуты** файла, например, его длина, тип, ссылка на элементы файла во внешней памяти. Память файла может быть организована по-разному – как список смежных областей (**блоков** или **записей),** одна смежная область, список индексных узлов, ссылающихся на блоки файла и т.д. В курсе подробно рассмотрены некоторые файловые системы. Кроме создания и удаления файла, основные операции над ним – **открытие** и **закрытие**. Открытие файла – это считывание в основную память его заголовка и, возможно, одного или нескольких соседних блоков. Оно должно быть выполнено перед выполнением операций чтения из файла или записи в файл. Закрытие файла – это обратная операция: сброс всех копий блоков на внешнюю память и освобождение областей основной памяти, занятых открытым файлом. ОС закрывает файлы процесса при его завершении, если процесс не сделал этого явно (последнее рекомендуется). При отображении файлов на внешнюю память возникают проблемы, аналогичные проблемам распределения основной памяти, - фрагментация, возможность исчерпания внешней памяти или ее **раздела (partition)** – смежной области внешней памяти, имеющей определенное символьное обозначение.

Создание и удаление директорий. Поддержка примитивов (пользовательских команд и библиотечных вызовов) для управления файлами и директориями.
Директория (directory) – это каталог (справочник) ссылок на группу файлов или других директорий, каждый (каждая) из которых имеет в данной директории свое уникальное символьное **имя**. Иерархия директорий позволяет организовать **поиск** файла по его символьному **пути** (**path),** например, в Windows: **c:\doc\plan.txt** – текстовый документ, содержащий план моих текущих действий, ссылка на который находится на диске C: , в директории **doc**. ОС управляет созданием и удалением директорий и поиском в них файлов по их путям. Следует иметь в виду, что на файл возможно несколько ссылок из разных директорий (хотя это и не рекомендуется), так что удаление элемента директории не означает и удаления файла – сам файл сохраняется, пока на него есть хотя бы одна ссылка. Более того,

в некоторых файловых системах (например, FAT в Windows) ошибочно удаленный файл можно восстановить, хотя и под другим именем. В других же файловых системах (например, в UNIX, где используются индексные блоки, хранящие адреса блоков файла) удаление файла – фатальная операция, от ошибок в которой может спасти только вовремя сделанная резервная копия файловой системы на диске или флэшке.

Сброс, или **резервное копирование (backup)** файлов на устойчивые носители (флэш-память, компакт-диск, ленточный стример и др.), с целью их последующего восстановления при сбое или при ошибке пользователя. Значение резервного копирования для пользователей ОС трудно переоценить. Все наиболее важные документы, директории, файловые системы должны регулярно копироваться на внешнюю память (желательно делать не одну, а несколько копий на разные носители). Это должно стать непреложным правилом для каждого пользователя. Трудно даже вспомнить, сколько раз автору приходилось выслушивать сетования и жалобы студентов, аспирантов, сотрудников на то, что у них в самый ответственный момент "полетел винчестер", из-за чего они не могут показать свою программу или отчет. Рецепт очень простой: **необходимо регулярно копировать важную информацию на устойчивые носители.** Если Вы работаете в локальной сети фирмы, исследовательской лаборатории и т.д., то в ней должен быть системный администратор, который должен заботиться о регулярном резервном копировании всех важных файловых систем. Возможности ОС позволяют выполнять такое копирование автоматически, в определенное время, - например, ночью, когда в офисе никого нет, но компьютеры локальной сети работают.

В некоторых ОС реализованы файловые системы с **криптованием** данных при записи в файл (например, система ZFS в Solaris). Такой подход позволяет решить проблему **сохранения конфиденциальности информации (privacy)** .

Управление вторичной памятью

Поскольку размер основной памяти недостаточен для постоянного хранения всех программ и данных, в компьютерной системе должна быть предусмотрена вторичная (внешняя) память для откачки (back up, swapping) части содержимого основной памяти.

В большинстве компьютерных систем в качестве главной вторичной памяти для хранения программ и данных используются диски.

ОС отвечает за выполнение следующих действий, связанных с управлением дисками:

- **Управление свободной дисковой памятью**;
- **Выделение дисковой памяти**;
- **Диспетчеризация дисков (disk scheduling).**

При управлении вторичной памятью возникают проблемы, аналогичные проблемам распределения основной памяти. Всякая память, даже самая большая по объему, рано или поздно может исчерпаться, либо фрагментироваться на множество мелких областей свободной памяти. О методах управления основной и внешней памятью речь пойдет подробно ниже в специальных разделах курса.

Управление сетевыми (распределенными) системами. Как уже было сказано в более ранних лекциях, распределенная система – это совокупность процессоров, которые не используют общую память или часы (такты процессора). Каждый процессор имеет собственную локальную память. Процессоры в такой системе соединены в сеть. Сетевое взаимодействие выполняется по определенному **протоколу** (интерфейсу, набору операций). Наиболее распространенный сетевой протокол – **TCP/IP**, основанный на **IP-адресах** машин (hosts); например, 190.100.125.1.

В распределенной системе ОС обеспечивает доступ пользователей к различным общим сетевым **ресурсам** – например, файловым системам или принтерам. Каждому общему ресурсу ОС присваивает определенное сетевое имя и управляет возможностью доступа к нему с различных компьютеров сети. ОС обеспечивает также **удаленный запуск программ** на другом компьютере сети – возможность входа на другой компьютер и работы на нем, с использованием памяти, процессора и диска удаленной, как правило, более мощной машины,

и использованием клиентского компьютера в качестве терминала. В Windows такая возможность называется **удаленный рабочий стол (remote desktop connection),** в UNIX, LINUX, Solaris – **rsh (remote shell)** и **rlogin (remote login).**

Доступ к общим ресурсам (shared resource) в распределенной системе позволяет:

- Ускорить вычисления;
- Расширить границы доступа к данным;
- Обеспечить более высокую надежность.

Система защиты (ptotection)

Термин **защита (protection)** используется для обозначения механизма управления доступом программ, процессов и пользователей к системным и пользовательским ресурсам.

Механизм защиты в ОС должен обеспечивать следующие возможности:

Различать **авторизованный,** или **санкционированный (authorized)** и **несанкционированный (unauthorized)** доступ. Под **авторизацией** понимается предоставление операционной системой пользователю или программе какого-либо определенного набора **полномочий (permissions),** например, возможности чтения или изменения файлов в файловой системе с общим доступом.

Описывать предназначенные для защиты элементы управления (конфигурации). Например, в UNIX используются специальные текстовые конфигурационные файлы для представления информации о файловых системах, к которым возможен сетевой доступ, с указанием списка машин (хостов), с которых возможен доступ, и набора действий, которые могут быть выполнены.

Обеспечивать средства выполнения необходимых для защиты действий (сигналы, исключения, блокировка и др.). Например, система защиты ОС должна фильтровать сетевые пакеты, получаемые извне локальной сети, выбирать и отсеивать "неблагонадежные" (получаемые с подозрительных IP-адресов), сообщать пользователю об обнаруженных и ликвидированных попытках сетевых атак с целью "взлома" Вашего компьютера (что и происходит на практике, например, при работе в Windows, когда Вы выходите в Интернет с Вашего компьютера). Если Вы нарушили условия защиты (например, Ваша программа попыталась обратиться к файлу, работать с которым у Вас нет полномочий), ОС должна выдать понятное сообщение и прекратить работу программы. В современных системах это делается с помощью генерации **исключений (exceptions),** например, **SecurityException**.

Система поддержки командного интерпретатора

Большинство команд для ОС задаются с помощью специальных управляющих операторов, предназначенных для выполнения следующих основных функций:

- **создания процессов и управления процессами**; например, в UNIX команда ps –a выводит в стандартный вывод процесса информацию обо всех активных процессах в системе, с указанием их номеров (PID);
- **выполнения ввода-вывода**; например, в системе MS DOS команда type file_name выполняет вывод на терминал содержимого заданного текстового файла;
- **управления вторичной памятью**; например, в UNIX команда share /mydir добавляет директорию **/mydir** к списку совместно используемых в локальной сети файловых систем;
- **управления основной памятью**; например, команда **swap** в ОС Solaris позволяет управлять размером пространства дисковой памяти для реализации виртуальной памяти (swap) и выводить информацию о его состоянии;
- **доступа к файловой системе**; например, в большинстве ОС команда cd new_dir устанавливает заданную директорию в качестве текущей (рабочей);
- **защиты**; например, в системе UNIX команда chmod 700 my_home_dir защитит Вашу домашнюю директорию от непрошенных любопытных глаз – "лазутчик" не сможет даже выполнить команду cd для этой директории и, тем более, читать в ней какие-либо файлы;

- **управления сетью**; например, команды telnet host_name и rlogin host_name (последняя доступна в системе UNIX) служат для удаленного входа на другой компьютер сети.

Программа, которая читает и интерпретирует операторы управления, называется **командным интерпретатором**. В Windows это интерпретатор **command.com**, доступный для выполнения команд в окне **MS DOS prompt**. В UNIX, Linux, Solaris это уже упоминавшиеся всевозможные "шеллы": **sh, csh, ksh, bash** – процессоры для интерпретации мощных командных языков. Функция командного процессора состоит в том, чтобы прочесть и исполнить очередной управляющий оператор (команду).

Сервисы (службы) ОС

Операционная система предоставляет для пользователей целый ряд сервисных возможностей, или, коротко, **сервисов (служб**):

Исполнение программ – загрузка программы в память и ее выполнение; например, в Windows при запуске программы ОС находит в файле ее двоичного кода (.exe) так называемую **заглушку для исполнения (execution stub),** содержащую ссылку на код головного метода main, и запускает его. В среде .NET этот же execution stub в файле двоичного кода используется системой для вызова не непосредственно исполняемой программы, а общего окружения времени выполнения – Common Language Runtime (CLR), которое обеспечивает особый режим (managed execution) выполнения программы.

Поддержка ввода-вывода – обеспечение интерфейса для работы программ с устройствами ввода-вывода. Например, в UNIX у каждой программы есть свой стандартный ввод и стандартный вывод (по умолчанию это терминал). В более старых ОС, например, IBM 360, привязку программы к устройствам ввода-вывода требовалось специфицировать с помощью громоздких DD (Data Definition) – предложений на специальном языке управления заданиями.

Работа с файловой системой – предоставление программам интерфейса для создания, именования, удаления файлов. Об этом уже много говорилось выше.

Коммуникация – обмен информацией между процессами, выполняемыми на одном компьютере или на других системах, связанных в сеть. В операционных системах реализуется с помощью общей памяти (**shared memory)** или передачи сообщений.

Обнаружение ошибок в работе процессора, памяти, устройств ввода-вывода и программах пользователей.

Дополнительные функции ОС

Данные функции реализованы не непосредственно для удобства пользователя, а для обеспечения выполнения операций системы. Это следующие возможности.

Распределение ресурсов между пользователями, программами и процессами, работающими одновременно.

Ведение **статистики** использования ресурсов, с целью выставления пользователям счетов (например, за сетевой трафик) или для анализа эффективности работы системы.

Защита – обеспечение того, чтобы доступ к любым ресурсам был контролируемым.

Системные вызовы (system calls) являются интерфейсом между выполняемой программой и операционной системой

Обычно системные вызовы доступны как специальные ассемблерные команды, например, в IBM 360 ассемблерная команда **svc 10** выполняет вызов супервизора (управляющей программы ОС) с номером системной функции 10.

Некоторые языки (C, C++, Java и др.) позволяют выполнять системные вызовы непосредственно, не "опускаясь" до ассемблерного уровня, с помощью вызовов специальных библиотечных функций (методов) типа **System("cd my_dir").**

При системном вызове ОС из программы пользователя возникает проблема передачи параметров. Используются три основных способа передачи параметров исполняемой программой операционной системе:

- **Передача параметров в регистрах**; например, в IBM 360 системная макрокоманда **GETMAIN** выделения области основной памяти ожидает, что ей в регистре номер 1 передана длина требуемой области памяти, а сама макрокоманда в результате своего выполнения записывает также в первый регистр адрес выделенной области основной памяти. Очевидно, что подобный интерфейс не вполне надежен – слишком много не очевидных умолчаний. А вдруг программист по ошибке запишет длину области памяти не в первый, а во **второй** регистр? Об этой ошибке никто ему не подскажет, и результат будет бессмысленным.
- **Запись параметров в таблицу**, расположенную в памяти, и передача адреса этой таблицы в регистре. Этот способ немного лучше, но все равно он зависит от джентльменского соглашения между автором программы и авторами ОС относительно передачи через регистр, соблюдение которого фактически никем не проверяется.
- **Запись (проталкивание) параметров в стек** программой и чтение (выталкивание) их из стека операционной системой. Такой способ гораздо лучше, так как он является стандартным способом передачи параметров при любом вызове процедуры или метода.

На рис. 6.1 изображен способ передачи параметров при системном вызове через таблицу, адрес которой передается в регистре.

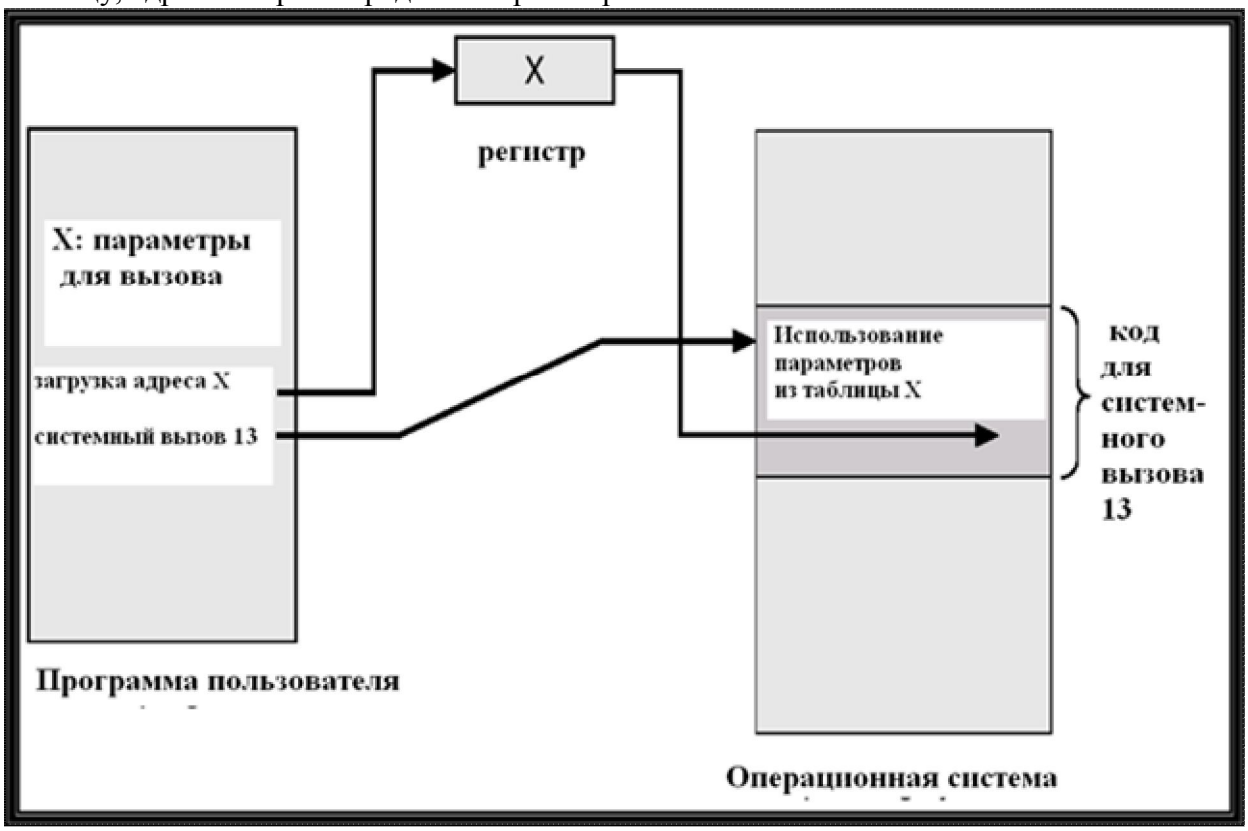

Рис. 6.1. Передача параметров системного вызова в таблице.

Различаются следующие основные виды системных вызовов:

- **Управление процессами**; например, в UNIX системный вызов fork создает новый параллельный процесс с новым пространством виртуальных адресов.
- **Управление файлами**; например, в UNIX системный вызов open (f, "rw") осуществляет открытие заданного файла для чтения и записи.
- **Управление устройствами**; например, системный вызов rewind осуществляет перемотку (позиционирование) магнитной ленты на начало.
- **Сопровождающая информация**; например, системный вызов env выдает в стандартный вывод информацию о значениях **переменных окружения –** переменных с символьными значениями, например, **PATH**, задающими окружение исполняемого процесса;

- **Коммуникации**; например, системный вызов CreateSocket создает новый **сокет** – системную структуру для обмена информацией клиента с сервером через TCP/IP – сеть.

Из примеров нетрудно видеть, что многие из этих возможностей ОС доступны также в виде выполняемых команд.

Исполнение программ в MS DOS

Как уже упоминалось, операционная система MS DOS была разработана в конце 1970-х гг. для 16-разрядных процессоров фирмы Intel (x86). Эта система, по сравнению с предшествующими ей по времени операционными системами для mainframe-компьютеров MULTICS, OS IBM 360 и др., была значительно проще по возможностям, в частности, она была однозадачной. Это сознательное упрощение было вызвано жесткими ограничениями по памяти: объем основной памяти, предоставляемый задаче для выполнения, был равен всего 640 килобайт. Схема распределения памяти при выполнении программ в системе MS DOS изображена на рис. 6.2.

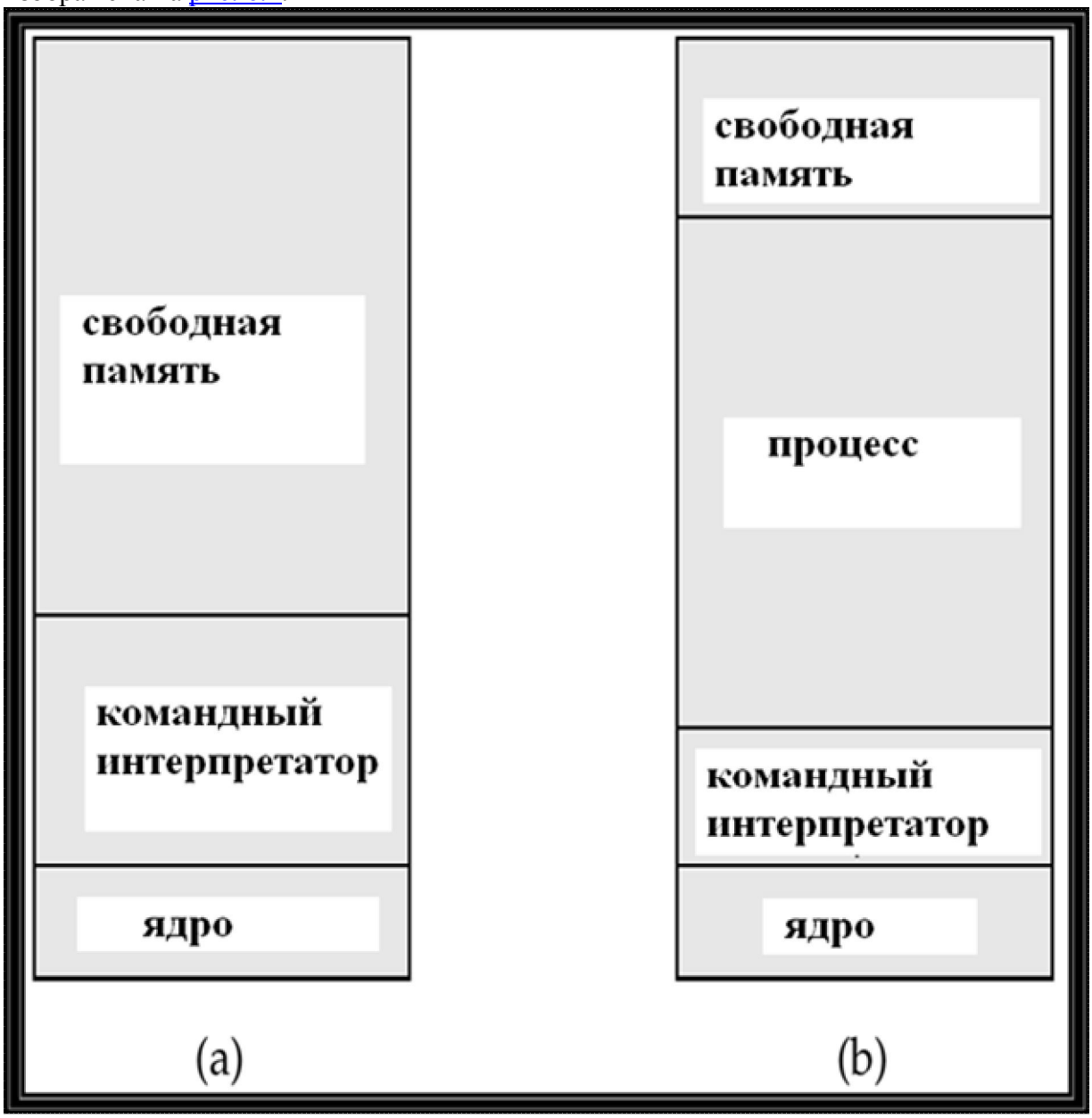

Рис. 6.2. Выполнение программ в MS DOS.

Автор со своей командой в конце 1980-х – начале 1990-х гг. выполнял большие программные разработки для MS DOS в интегрированной среде Турбо Паскаль. Если размер программы или одного ее модуля превышал 640 K, приходилось организовывать **оверлейную структуру (overlay)** – разбиение программы на группы взаимосвязанных модулей, таких, что

различные группы одновременно в памяти не нужны, и поочередно загружать в выделенную для задачи область памяти необходимые группы модулей, к которым происходило обращение. Такой метод был вынужденным и типичным для многих программистов при разработке больших программ в среде MS DOS.

Исполнение нескольких программ в UNIX

Система UNIX, первоначально, как уже говорилось, разработанная в 1970 г. для миникомпьютеров PDP 10, была многозадачной (т .е. поддерживала режим мультипрограммирования) – несмотря на ограниченный объем памяти, система могла одновременно обрабатывать несколько заданий пользователей. Система поддерживала также и режим разделения времени, а впоследствии – и сетевое взаимодействие. В UNIX работали компиляторы с нескольких языков, в том числе – Паскаль. Использовались инструментальные средства, ставшие классическими, - утилита make – для сборки проектов, утилита lex – генератор лексических анализаторов в компиляторах; yacc – генератор синтаксических анализаторов в компиляторах; grep – утилита текстового поиска и фильтрации с помощью регулярных выражений; awk – язык для обработки табличной информации; sed – потоковый редактор текстов. Была реализована, с одной стороны, развитая, с другой - унифицированная файловая система, лишенная надуманных сложностей IBM 360 и "Эльбруса". Схема размещения в памяти одновременно нескольких заданий пользователя и их обработки в режиме мультипрограммирования изображена на рис. 6.3.

Рис. 6.3. Выполнение нескольких пользовательских программ в системе UNIX.

Коммуникационные модели

Одновременно обрабатываемые операционной системой пользовательские процессы должны иметь возможность коммуникации друг с другом. Такая коммуникация реализуется в операционных системах двумя способами – с помощью общей памяти и с помощью передачи сообщений. Обе схемы коммуникации процессов изображены на рис. 6.4.

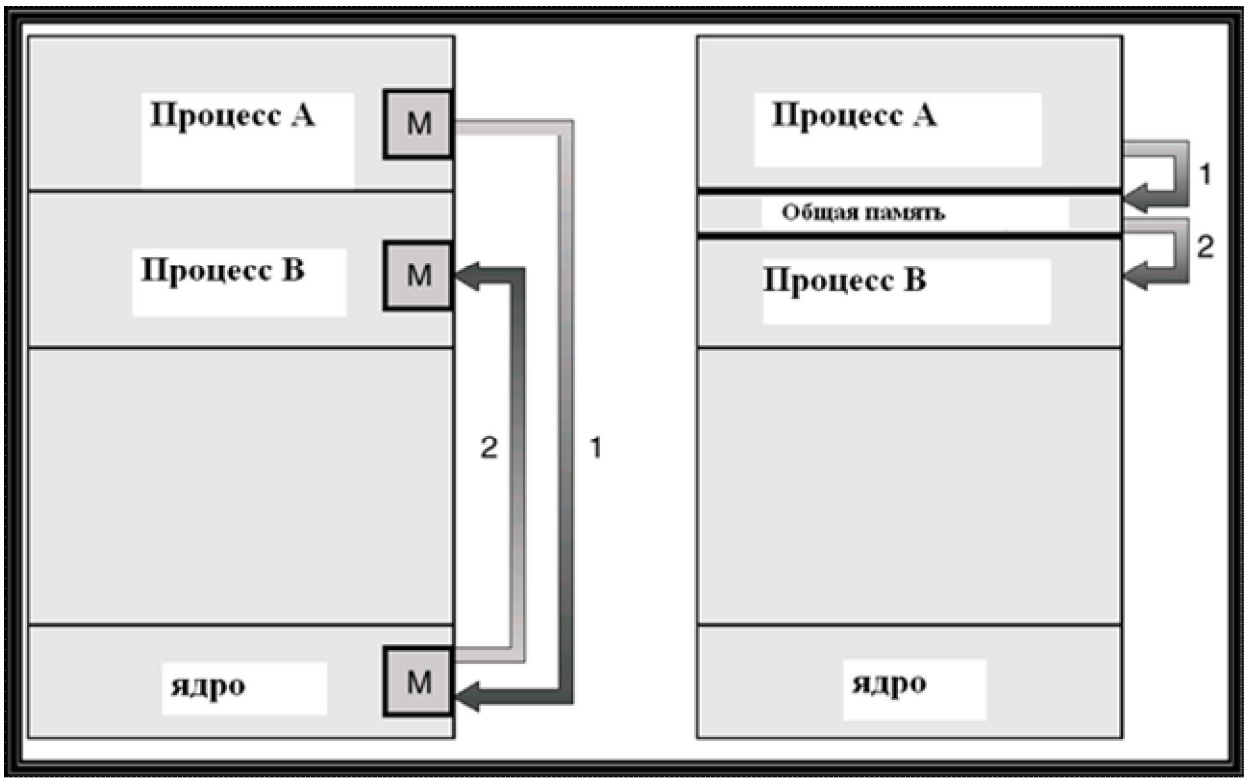

Рис. 6.4. Коммуникация процессов с помощью передачи сообщений и с помощью общей области памяти.

При первом способе, процесс А для передачи сообщения М процессу В выполняет системный вызов - например, **send (B, M),** - т.е. фактически передает это сообщение ядру ОС. Процесс В, в свою очередь, выполняет системный вызов для получения сообщения – например, **M = receive (B)** – т.е. фактически получает сообщение М от ядра ОС, подобно какому-либо ресурсу системы.

При втором способе, оба процесса взаимодействуют через общую область памяти, адрес которой известен им обоим. Такой способ может оказаться более быстрым, но необходимо учесть, что в данном случае процессы должны позаботиться о **синхронизации**, так как общая область памяти – это общий ресурс, при обращении к которому может возникнуть race condition.

Системные программы – разновидность сервисов операционной системы

Системные программы обеспечивают удобное окружение для разработки и исполнения программ. Они подразделяются на программы:

Управления файлами; например, файл-менеджеры типа Norton Commander и Far в MS DOS и Windows Commander – в Windows;

Получения информации о состоянии; например, Task Manager – программа в системе Windows для управления процессами и получения информации об их состоянии, загрузке процессора и используемой памяти. Вызывается комбинацией клавиш **Ctrl – Alt – Del.**

Создания и изменения файлов; например, текстовые редакторы **notepad** и **wordpad** и программа **paint** для создания и редактирования рисунков, поставляемые с ОС Windows.

Поддержки языков программирования; например, компиляторы с языков Си (**cc**) и Java (**javac**), поставляемые с большинством операционных систем;

Загрузки и исполнения программ; например, ld – загрузчик и редактор связей UNIX;

Коммуникации; например, **Windows Messenger –** программа обмена мгновенными сообщениями, часть ОС Windows.

Использование ОС большинством пользователей основано на использовании системных программ, а не системных вызовов. Причина этого в том, что уровень интерфейса

системных программ несколько выше, чем уровень системных вызовов (например, не требует передачи параметров через регистры процессора). Системные программы ближе к сути решаемых задач и поэтому понятнее пользователям, чем системные вызовы.

Структура системы MS DOS

После краткого обзора возможностей ОС проанализируем теперь особенности их структуры и архитектуры и некоторые полезные методы их разработки. Простейшим примером с этой точки зрения является MS DOS, разработанная по принципу: обеспечить максимум функциональности, используя минимум памяти (напомним об ограничении в 640 K на объем памяти для программы в MS-DOS). В MS DOS нет явного разделения на модули. Поэтому, хотя MS-DOS и имеет некоторую архитектуру, уровни функциональности и интерфейсы в ней не отделены четко друг от друга. Уровни абстракции модулей MS DOS изображены на рис. 6.5. Подробнее о концепции уровней абстракции, полезной для разработки ОС, которую мы пока используем интуитивно, - в конце данной лекции.

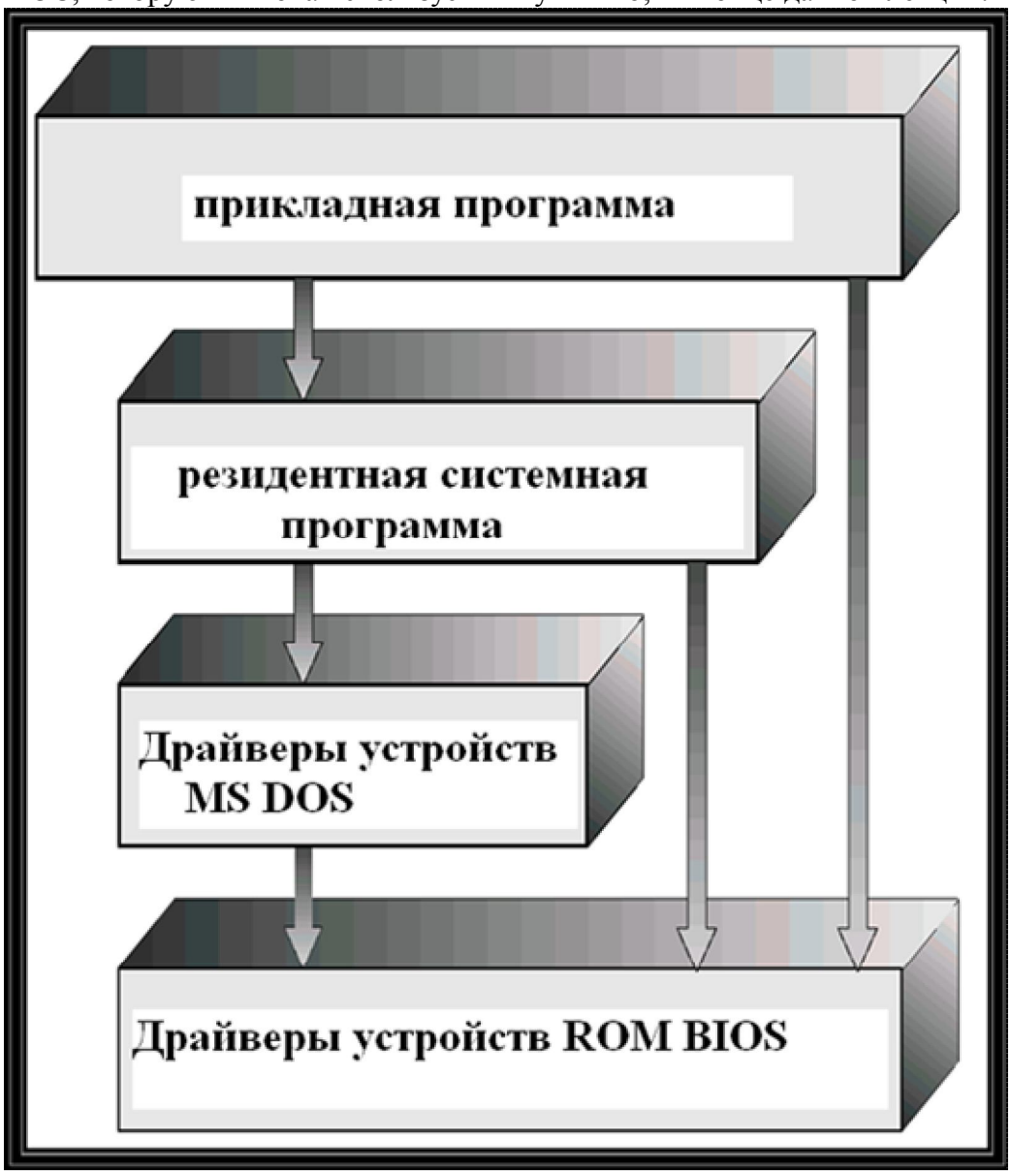

Рис. 6.5. Уровни абстракции модулей MS DOS.

В схеме можно выделить четыре уровня абстракции. Наиболее высокий – уровень пользовательской программы. Более низкий – резидентная системная программа, компонента ядра ОС. Еще ниже – уровень драйверов устройств, являющихся частью ядра MS DOS, и самый низкий – уровень драйверов, хранящихся в **ROM BIOS (Read-Only Memory of the Basic Input-Output System)** – постоянной памяти BIOS, системного модуля компьютера,

которому передается управление непосредственно после его включения. ROM BIOS содержит наиболее важную часть драйверов, например, драйвер материнской платы (motherboard). ОС загружает и использует драйверы других устройств, например, принтера.

Структура системы UNIX

Хотя система UNIX и имеет более модульную структуру, чем MS DOS, ее архитектура ограничена функциональностью аппаратуры, для которой она были первоначально разработана, - миникомпьютеров. Поэтому первые версии UNIX имели ограниченное структурирование.

Система UNIX состоит из двух частей: **системные программы** и **ядро**.

Ядро содержит все модули, уровень абстракции которых ниже системных вызовов, но выше непосредственно аппаратных модулей.

UNIX обеспечивает поддержку файловой системы, диспетчеризацию процессора, управление памятью и другие основные функции ОС.

Архитектура UNIX изображена на рис. 6.6.

Рис. 6.6. Структура системы UNIX

В архитектуре UNIX уже четко прослеживаются три уровня абстракции – пользовательский (системные программы), системных вызовов и низкоуровневых модулей взаимодействия с аппаратурой.

Уровни абстракции

В конце 1960-х гг., при разработке операционной системы THE (название – аббревиатура, означающая "Технический университет Эйндховена"), Э. Дейкстра предложил для своего времени весьма новый и прогрессивный принцип уровней абстракции, полезный при разработке любой сложной программной системы, в том числе – столь сложной, как

операционная система. Согласно этому принципу, ОС (или другая сложная программа) реализуется в виде набора (иерархии) **уровней абстракции (abstraction layers),** каждый из которых реализован на основе предыдущего уровня. **Уровень 0 (layer 0)** образует **аппаратура** (**hardware**); самый высокий уровень N (layer N) является пользовательским интерфейсом с операционной системой. Каждый уровень абстракции N > 0 – это группа модулей, при реализации которого, согласно принципам модульного программирования, используются только модули предшествующего уровня (N-1).

"Перескакивание" через уровень (т.е., например, использование при реализации модуля уровня N вызовов модулей уровня N – 2) не рекомендуется и является нарушением технологии, которое может привести к ошибкам.

Подобный подход позволяет проектировать сложную программную систему шаг за шагом, снизу вверх, причем на каждом шаге (уровне K) используется все более и более удобная система обозначений уровня K-1. Это позволяет абстрагироваться от лишних деталей, что и является объяснением названия данного метода.

Заметим, что, по сути дела, уровни абстракции – движущая сила и принцип развития всего программного обеспечения в целом, а не только операционных систем. Каждая новая программа разрабатывается не с нуля, а на некотором достаточно высоком уровне абстракции, используя другие уже разработанные системы.

Ключевые термины

Big endian – архитектура памяти компьютера, при которой нумерация байтов в каждом слове памяти начинается слева, т.е. начиная со старших битов.

Little endian – архитектура памяти компьютера, при которой нумерация байтов в каждом слове памяти начинается справа, т.е. начиная с младших битов.

ROM BIOS (Read-Only Memory of the Basic Input-Output System) – постоянная память, входящая в состав BIOS, системного модуля компьютера, которому передается управление непосредственно после его включения; содержит часть драйверов для модулей аппаратуры.

Авторизация - предоставление операционной системой пользователю или программе какого-либо определенного набора **полномочий (permissions),** например, возможности чтения или изменения файлов в файловой системе с общим доступом.

Атрибут файла – его характеристика, например, длина и начальный адрес во внешней памяти.

Блок – смежная область внешней памяти файла, как правило, считываемая или записываемая одной операцией ввода-вывода.

Виртуальная память – расширение основной памяти путем хранения ее образа на диске и организации подкачки в основную память фрагментов (страниц или сегментов) памяти процесса и ее откачки на диск по мере необходимости.

Директория (directory) – каталог ссылок на группу файлов или других директорий, каждый (каждая) из которых имеет в данной директории свое уникальное символьное **имя**.

Заглушка для исполнения (execution stub) – область файла исполняемого двоичного кода, содержащая ссылку на код головного метода (процедуры), обычно – **main** .

Заголовок файла – начальная часть файла, в которой хранятся его **атрибуты** .

Закрытие файла – операция, обратная **открытию файла**: сброс всех копий блоков файла на внешнюю память и освобождение всех областей основной памяти, занятых открытым файлом.

Защита (protection) - механизм управления доступом программ, процессов и пользователей к системным и пользовательским ресурсам.

Командный интерпретатор - программа, читающая и интерпретирующая операторы управления операционной системы, задаваемые пользователем с терминала или в виде командного файла.

Набор данных (data set) – то же, что и **файл** (в терминологии фирмы IBM).

Неустойчивое (**volatile**) - устройство памяти, типичное для основной памяти компьютеров, при котором ее содержимое теряется при сбое системы или при выключении питания.

Оверлейная структура (overlay) – разбиение программы на группы взаимосвязанных модулей, таких, что различные группы одновременно в памяти не нужны, с целью их поочередной загрузки в выделенную для программы область памяти.

Открытие файла – считывание в основную память его заголовка и, возможно, одного или нескольких соседних блоков перед выполнением операций ввода-вывода.

Память файла – совокупность его элементов, хранящихся во внешней памяти (например, на диске).

Переменные окружения – набор системных переменных с символьными значениями, например, **PATH**, задающих окружение исполняемого процесса.

Протокол – интерфейс, набор операций (например, для работы в локальной сети).

Путь (**path**) – символьная строка для поиска файла по имени в иерархии директорий.

Раздел (partition) – смежная область внешней памяти, имеющая в ОС определенное символьное обозначение (например, D:).

Сброс, или **резервное копирование (backup)** – копирование файлов на устойчивые носители (флэш-память, компакт-диск, ленточный стример и др.), с целью их последующего восстановления при сбое или при ошибке пользователя.

Сборка мусора – поиск не используемых фрагментов в памяти, на которые потеряны ссылки, и **уплотнение (компактировка)** памяти – сдвиг всех используемых фрагментов по меньшим адресам, с корректировкой всех адресов.

Системный вызов (system call) - интерфейс между выполняемой программой и операционной системой в виде явного вызова процедуры, метода или макроса, являющегося частью ОС.

Сокет (socket) – системная структура для обмена информацией клиента с сервером через TCP/IP – сеть.

Удаленный запуск программ (на другом компьютере сети) – возможность входа на другой компьютер сети и работы на нем, с использованием памяти, процессора и диска удаленной (как правило, более мощной) машины и использованием клиентского компьютера в качестве терминала.

Уплотнение (компактировка) памяти – сдвиг всех используемых фрагментов по меньшим адресам, с корректировкой всех адресов, при **сборке мусора**.

Уровень абстракции (abstraction layer) - группа модулей, при реализации которых используются только модули предшествующего уровня.

Файл (file) – совокупность логически взаимосвязанной информации, расположенная во внешней памяти.

Фрагментация – дробление памяти на мелкие свободные части, вследствие неточного совпадения длин имеющихся свободных и требуемых пользовательскому процессу областей памяти.

Краткие итоги

Основная память – большой массив слов или байтов. Байты в слове могут нумероваться двумя способами, по которому различают big endian- и little endian – архитектуры. Задачи ОС по управлению памятью – отслеживание, какие области памяти используются какими процессами, стратегия загрузки процессов в основную память, выделение и освобождение памяти.

Файл (набор данных) – логически взаимосвязанная совокупность информации во внешней памяти. Функции ОС по управлению файлами – создание и удаление, открытие и закрытие, управление директориями и поиском файлов в них, отображение файлов во внешнюю память, их резервное копирование.

Вторичная память (например, на диске) используется как расширение основной памяти. ОС управляет свободной дисковой памятью, выделяет дисковую память и выполняет диспетчеризацию дисков.

ОС поддерживает работу в распределенной системе (сети) – сетевые протоколы, взаимодействие с общими сетевыми ресурсами, удаленный запуск программ.

Система защиты в ОС – механизм управления доступом программ и пользователей к системным и пользовательским ресурсам.

ОС поддерживает командный интерпретатор, читающий и интерпретирующий управляющие операторы (команды), задаваемые пользователями с терминала или в виде командных файлов. Команды могут выполнять действия над файлами, процессами, основной и вторичной памятью, защиту, управление сетью.

Основные сервисы ОС – исполнение программ, поддержка ввода-вывода, работа с файловой системой, коммуникация, обнаружение ошибок.

Дополнительные функции ОС – распределение ресурсов, ведение статистики, защита.

Системный вызов – интерфейс программ с ОС в виде вызовов процедур, методов или макросов, являющихся частью ОС. Способы передачи параметров системному вызову: через регистр, через таблицу, адрес которой в регистре, или через стек. Системные вызовы выполняют управление процессами, файлами, устройствами, выдают сопровождающую информацию, осуществляют коммуникации.

MS DOS – однозадачная операционная система. Одновременно система обрабатывает только одну задачу, размер памяти которой не может превышать 640 килобайт.

UNIX, в отличие от MS DOS, поддерживает мультипрограммирование, т.е. может одновременно обрабатывать несколько задач и хранить их в памяти.

Коммуникация между процессами может осуществляться с помощью передачи сообщений или через общую область памяти.

Системные программы – более удобный и чаще применяемый пользовательский интерфейс с ОС, чем системные вызовы. Они осуществляют управление файлами, получение информации о состоянии, создание и изменение файлов, поддержку языков программирования, загрузку и исполнение программ, коммуникации.

В архитектуре MS DOS нет явного разделения на модули. Различаются уровень прикладной программы, резидентной системной программы, драйверы устройств MS DOS, драйверы устройств ROM BIOS.

В архитектуре системы UNIX более явно выделяются несколько уровней абстракции. Система состоит из ядра и системных программ. Для обращения к ядру используется интерфейс системных вызовов. На самом верхнем уровне пользователям доступны командные процессоры, компиляторы и интерпретаторы, системные библиотеки.

Э. Дейкстра предложил подход к разработке ОС, основанный на уровнях абстракции. Каждый из них представляет собой группу модулей, в реализации которых используются только модули непосредственно предшествующего уровня. Это дает возможность более удобной разработки и позволяет абстрагироваться от лишних деталей. По сути дела, принцип уровней абстракции – движущая сила всего процесса развития программного обеспечения.

9. Лекция: Методы взаимодействия процессов

В лекции рассматриваются: взаимодействие процессов: проблема ограниченного буфера; проблема "производитель – потребитель"; прямая и косвенная связь процессов; клиент-серверная взаимосвязь; сокетная связь; удаленный вызов процедуры (RPC) и удаленный вызов метода (RMI); выстраивание параметров (marshaling).

Введение

Взаимодействие процессов – основа для распараллеленного, эффективного решения задач с помощью группы процессов, координирующих свои действия друг с другом. В лекции

рассмотрены некоторые классические схемы взаимодействия процессов при решении типовых задач (например, схема производитель – потребитель), а также виды взаимодействия процессов между собой с помощью передачи сообщений, сокетов, удаленных вызовов процедур и методов.

Независимые и взаимодействующие процессы

С точки зрения взаимосвязи, процессы подразделяются на **независимые** и **взаимодействующие.**

Независимый процесс – процесс, никак не связанный с другими процессами, который не может влиять на исполнение других процессов или испытывать их влияние.

Взаимодействующий (совместный) процесс – процесс, который может влиять на исполнение других процессов или испытывать их влияние.

Преимущества взаимодействующих процессов очевидны:

- **Совместное использование данных**; процессы могут работать с общими данными, при условии их синхронизации (рассматриваемой в следующих лекциях);
- **Ускорение вычислений**;
- **Модульность**: организация взаимодействующих процессов – это метод параллельного решения задачи, декомпозируемой на относительно независимые части, части, каждую из которых решает один из взаимодействующих процессов
- **Удобство**.

Виды организации взаимосвязи процессов

С точки зрения видов взаимосвязи родительского и дочернего процессов, процессы подразделяются на **независимые**, **подчиненные** и **сопроцессы**.

Подчиненный процесс – процесс, зависящий от процесса-родителя. Подчиненный процесс уничтожается при уничтожении родительского процесса, как в системах UNIX и ОС "Эльбрус". Процесс-родитель перед своим завершением должен ожидать завершения всех своих подчиненных процессов.

Независимый процесс – дочерний процесс, выполняемый независимо от процесса-родителя. Типичные примеры: **процессы-демоны** в UNIX, запускаемые начальным процессом **init**. Например, **cron –** процесс-демон, организующий вызов заданных в специальной таблице **crontab** действий с заданной периодичностью (автоматическое резервное копирование всех файловых систем на ленту в полночь); **smbd –** процесс-демон, управляющий серверным программным обеспечением SAMBA для сетевого доступа с Windows-машин к файлам UNIX-машины.

Сопроцесс (coprocess, coroutine) – процесс, равноправно взаимодействующий с другими такими же процессами; хранит свое текущее **локальное управление** (program counter); взаимодействует с другим сопроцессом **Q** с помощью операций **resume (Q).** Взаимодействие нескольких сопроцессов друг с другом операторами resume полностью равноправно. Данный механизм взаимодействия принципиально отличается от вызова процедуры. Операция **detach** (**открепить**) переводит сопроцесс в пассивное состояние, в котором могут быть доступны только его глобальные данные, но его программа уже завершена и не подлежит повторному запуску. Сопрограммное взаимодействие реализовано в языке СИМУЛА-67, который, как известно, стал родоначальником и объектно-ориентированного подхода.

Парадигма (шаблон) взаимодействия процессов: производитель – потребитель

Реализация взаимодействия процессов может быть основана на одной из классических парадигм (шаблонов), сложившейся за десятилетия развития программирования. В данном разделе рассмотрим одну из наиболее распространенных из парадигм взаимодействия процессов - производитель – потребитель: процесс-производитель (**producer**) генерирует в некотором буфере информацию, которая используется процессом-потребителем (**consumer**).

При реализации данной парадигмы возможны схемы с неограниченным и ограниченным буфером, используемым для связи двух процессов.

- Схема с **неограниченным буфером (unbounded buffer)** подразумевает, что на размер используемого буфера теоретически нет ограничений.
- Схема с **ограниченным буфером (bounded buffer)** предполагается определенное ограничение размера буфера, например, константой **BUFFER_SIZE.**

При реализации следует учесть, что схема с ограниченным буфером, с точки зрения принципов надежных и безопасных вычислений, представляет опасность **атаки "переполнение буфера"(buffer overrun)** – ошибочного или преднамеренного превышения размера буфера. Чтобы избежать этой уязвимости, при заполнении буфера необходимо проверять его размер.

Реализуем ограниченный буфер следующим образом. Информация хранится в массиве с двумя указателями: in - для считывания и использования очередного элемента информации процессом-потребителем и out - для записи очередного сгенерированного элемента информации процессом-производителем. При считывании из буфера очередной элемент удаляется, и указатель in, соответственно, продвигается. При записи в буфер продвигается указатель out. Для удобства будем считать буфер **циклическим**, т.е. при его заполнении следующим заполняемым элементом будет нулевой (если он освободился), следующим после него – первый и т.д. Таким образом, процесс-производитель должен вычислять индекс в буфере, по которому он записывает следующий элемент, по формуле (out + 1) % BUFFER_SIZE, где "%" операция взятия остатка от деления. Аналогично, процесс-потребитель должен вычислять индекс следующего элемента информации в буфере по формуле (in + 1) % BUFFER_SIZE. Учтем также две возможных ситуации: **переполнение буфера** (при генерации производителем числа элементов, большего длины буфера) и **исчерпание буфера** (в случае, если потребитель взял из буфера последний на данный момент сгенерированный элемент). Чтобы избежать обращения за границы буфера, при переполнении буфера производитель должен будет ждать, пока в буфере не освободится хотя бы один элемент, а при исчерпании буфера должен будет ждать потребитель, пока хотя бы один новый элемент не появится в буфере.

Коммуникация процессов

Рассмотрим теперь возможные механизмы для непосредственной коммуникации процессов и синхронизации их действий.

Наиболее распространенный их них - **система сообщений**; при этом процессы взаимодействуют между собой без обращений к общим переменным (сравните с алгоритмами производителя и потребителя раздела 9.4).

Средства коммуникации между процессами обеспечивают две операции вида:

- **send (message**) – отправка сообщения **message**; размер сообщения может быть постоянным или переменным;
- **receive (message**) – получение сообщения в буфер **message**.

Если процессам **P** и **Q** требуется взаимодействовать между собой, им необходимо:

- Установить **связь (communication link)** друг с другом
- Обменяться сообщениями вида send/receive.

Реализация связи может быть физической (общая память, аппаратная шина) или логической (например, логические свойства).

При реализации коммуникационного механизма между процессами необходимо решить следующие вопросы:

- Как устанавливается связь?
- Можно ли установить связь более чем двух процессов?
- Сколько связей может быть установлено между двумя заданными процессами?
- Какова пропускная способность линии связи?
- Является ли длина сообщения по линии связи постоянной или переменной?

- Является ли связь ненаправленной или двунаправленной (дуплексной)?

Будем использовать данный контрольный список вопросов при анализе различных способов коммуникации процессов.

Непосредственная коммуникация процессов

При **непосредственной коммуникации (direct communication)** процессы именуют друг друга явно – по именам или по адресам (указателям), которые указываются в вызовах коммуникационных примитивов, например:

- **send (P, message**) – послать сообщение процессу **P**;
- **receive (Q, message**) – получить сообщение от процесса Q.

При данном способе коммуникации свойства линии связи, согласно контрольному списку раздела 9.5, следующие:

- Связь устанавливается автоматически (ее реализуют операционная система или отдельные коммуникационные инструменты).
- Связь ассоциируется только с одной парой взаимодействующих процессов.
- Между каждой парой процессов всегда только одна связь.
- Связь может быть ненаправленной, но, как правило, она двунаправленная (процесс может получить сообщение от другого явно заданного процесса и принять от него сообщение).

Косвенная коммуникация процессов

При **косвенной коммуникации** сообщения направляются и получаются через **почтовые ящики** (**mailboxes**), или **порты** (**ports**) **–** системные структуры, предназначенные для приема, хранения и передачи сообщений. Для определенности будем использовать термин **почтовый ящик.**

Каждый почтовый ящик имеет уникальный идентификатор.

Процессы могут взаимодействовать, только если они имеют общий почтовый ящик.

Свойства линии связи, согласно списку раздела 9.4, в этом случае следующие:

- Связь устанавливается, только если процессы имеют общий почтовый ящик.
- Связь одного процесса может быть установлена со многими процессами (которым доступен тот же почтовый ящик).
- Каждая пара процессов может иметь несколько линий связи (так как сообщения могут посылаться через различные почтовые ящики).
- Связь может быть ненаправленной или двунаправленной.

При косвенном способе коммуникации процессы используют набор операций вида:

- Создать новый почтовый ящик.
- Отправить (принять) сообщение через почтовый ящик.
- Удалить почтовый ящик

Основные операции коммуникации принимают вид:

- **send (A, message**) – послать сообщение в почтовый ящик A.
- **receive (A, message**) – получить сообщение из почтового ящика A.

Как мы видим, в данном случае **не** используются адреса или имена процессов-корреспондентов; вместо них задаются имена **почтовых ящиков**.

Чтобы лучше осознать суть и особенности данного метода коммуникации, проведем следующую аналогию. Представьте себе очень привычную для Вас электронную почту, которой Вы пользуетесь каждый день. Вы имеете один или несколько почтовых ящиков (email-адресов) и можете послать через любой из них электронное письмо Вашему корреспонденту. При приеме электронной почты Вы обычно задаете режим типа "**receive all"** (принять все сообщения со всех адресов), т.е. устанавливаете последовательно несколько линий связи с корреспондентами. Аналогичным образом и взаимодействуют процессы при косвенном способе коммуникации. Понятны и возможные проблемы, и способы их решения в обоих случаях: например, если один адрес (почтовый ящик) не работает, можно попытаться послать сообщение через другой. Однако есть и отличия: при отправке электронной почты

через некоторый почтовый ящик Вы все же явно указываете email-адрес получателя. С процессами дело обстоит иначе, из-за чего могут возникнуть проблемы.

Вот возможная проблема, возникающая при использовании общего почтового ящика. Пусть процессы **P1, P2,** и **P3** используют почтовый ящик **A**. **P1**, посылает сообщение; **P2** и **P3** принимают. Возникает вопрос: кто (какой из процессов) получает сообщение? Как решить данную проблему? Вот возможные решения:

- Ограничить связь только двумя процессами;
- Разрешить только одному процессу в каждый момент исполнять операцию получения;
- Разрешить системе произвольным образом определить получателя;
- Отправитель нотифицируется, кто является получателем.

Очевидно, что каждое из решений имеет свои достоинства и недостатки.

При косвенной связи процессов может оказаться необходимой **синхронизация.** Дело в том, что передача сообщений может выполняться с блокировкой (синхронно) или без блокировки (асинхронно). Соответственно, основные операции send и receive могут быть с блокировкой или без блокировки.

Буферизация и очередь сообщений

С коммуникационной линией связывается **очередь сообщений**, реализованная одним из трех возможных способов:

1. **Нулевая емкость очереди сообщений** означает, что сообщения не могут храниться в очереди. Поэтому при использовании данного способа отправитель должен ждать получателя. Такая схема коммуникации называется **рандеву (rendezvous)** и используется, например, в языке Ада.
2. **Ограниченная емкость** очереди сообщений – конечная длина очереди, в которой может храниться максимум **n** сообщений. Данный способ является общеупотребительным, однако в данном случае, как уже отмечалось, необходимо предотвратить опасность атаки "buffer overrun", т.е. в любой операции проверять длину буфера (очереди). Отправитель должен ждать, если очередь заполнена.
3. **Неограниченная емкость очереди сообщений** – (теоретически) она имеет бесконечную длину. В данном случае получатель никогда не ждет.

Клиент-серверная взаимосвязь – один из наиболее распространенных видов коммуникации процессов

Используются, в частности, следующие ее разновидности, которые мы и рассмотрим:

- Сокеты (Sockets)
- Удаленные вызовы процедур (Remote Procedure Calls – RPC)
- Удаленные вызовы методов (Remote Method Invocation – RMI).

Сокеты – наиболее распространенный способ связи клиента и сервера в сети. Впервые они были реализованы в UNIX BSD 4.2. Сокет можно определить как отправную (конечную) точку для коммуникации. Сокет создается клиентом для взаимодействия с сервером. Сокет связан с определенным номером **порта,** через который клиент и сервер обмениваются информацией, используя числовой или символьный последовательный поток. Сервер, со своей стороны, **прослушивает** порт с заданным номером и создает для этого **серверный сокет**. По сути дела, сокет можно представлять как конкатенацию IP-адреса и порта. Например, сокет 161.25.19.8:1625 ссылается на порт 1625 на машине (хосте) 161.25.19.8. Коммуникация осуществляется между парой сокетов – клиентским и серверным. Она изображена на рис. 9.1.

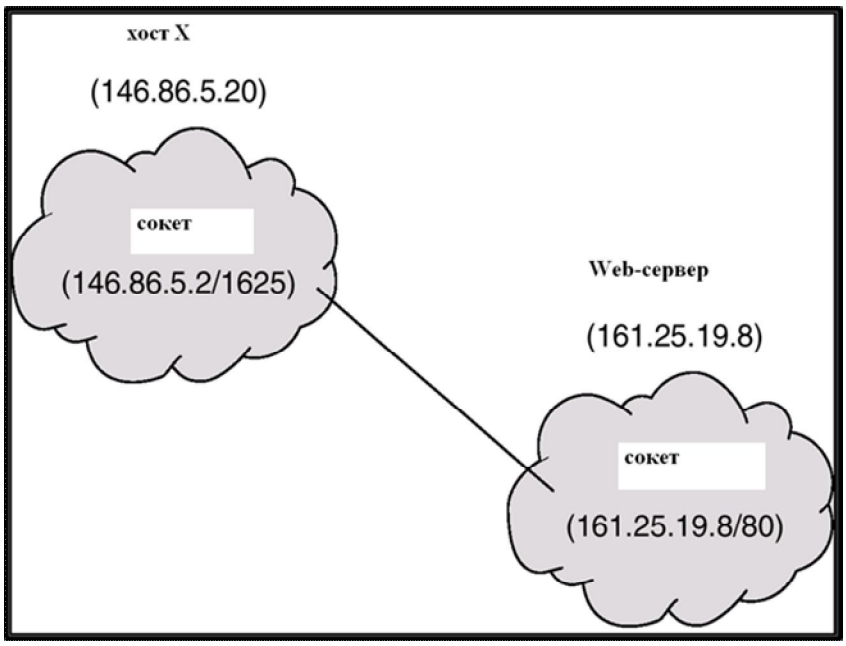

Рис. 9.1. Взаимодействие с помощью сокетов.

Удаленные вызовы процедур (Remote Procedure Calls – RPC) впервые предложены фирмой Sun и реализованы в ОС Solaris.

Удаленный вызов процедуры (RPC) – абстракция вызова процедуры между процессами в сетевых системах. Он основан на следующей идее. В клиентской части создаются **заглушка (proxy, stub)** – локальная процедура, осуществляющая связь с фактической процедурой, находящейся на сервере. Заглушка в клиентской части находит сервер и **выстраивает (marshals)** параметры для их передачи на сервер по сети. Проблема здесь в том, что адресация на клиенте и на сервере различная, и передавать адрес в памяти каких-либо данных с одного хоста на другой не имеет смысла. Поэтому приходится использовать особую форму передачи информации в виде последовательного потока байтов. Заглушка в серверной части принимает сообщение, распаковывает параметры, преобразует их к нормальному виду и выполняет процедуру на сервере.

Схема организации удаленного вызова процедуры изображена на рис. 9.2.

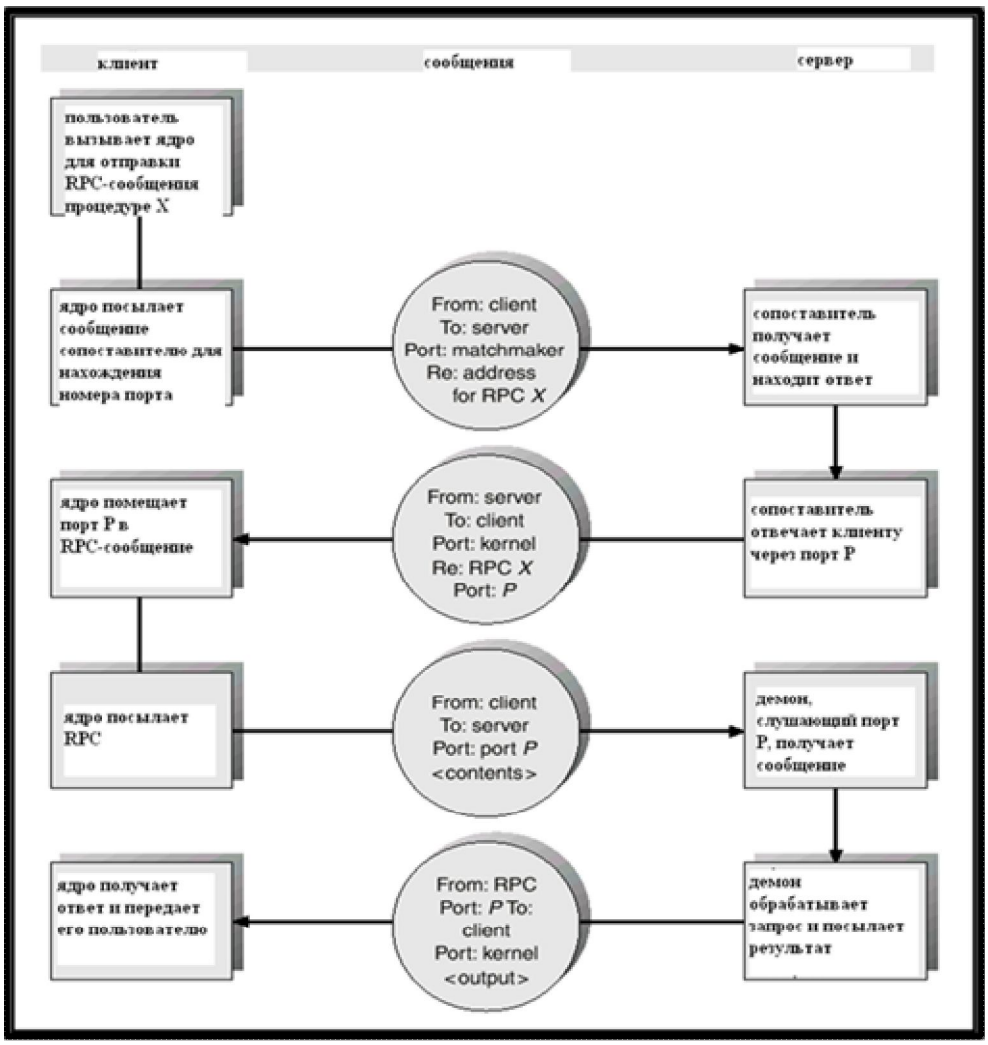

Рис. 9.2. Исполнение RPC.

Удаленный вызов метода (Remote Method Invocation, RMI) – механизм в Java-технологии, аналогичный RPC, но в объектно-ориентированной форме.

RMI позволяет Java-приложению на одной машине вызвать метод удаленного объекта.

Схема RMI изображена на рис. 9.3.

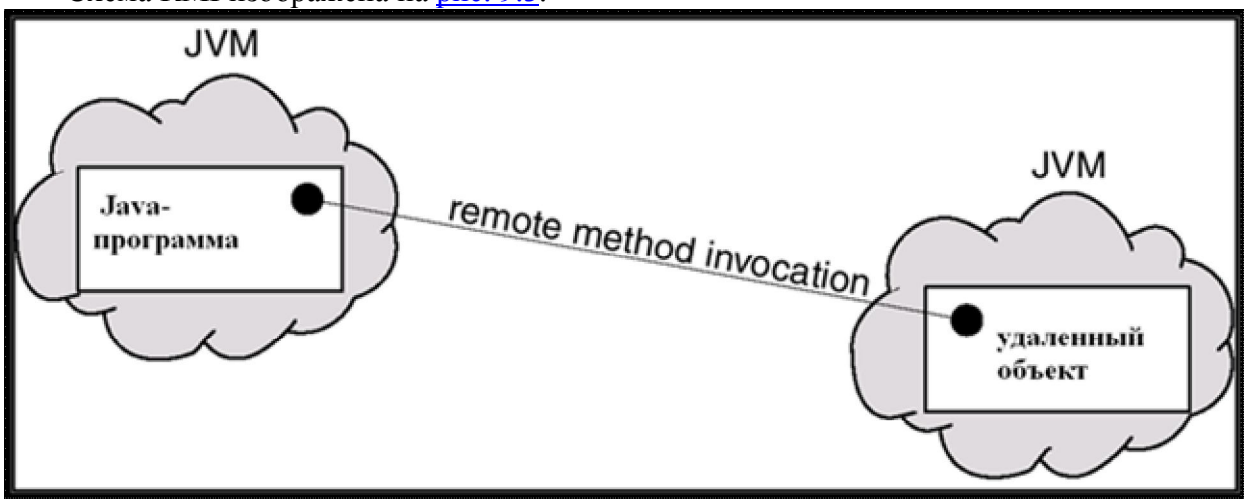

Рис. 9.3. Удаленный вызов метода в Java.

Схема выстраивания параметров и результатов при удаленных вызовах изображена на рис. 9.4.

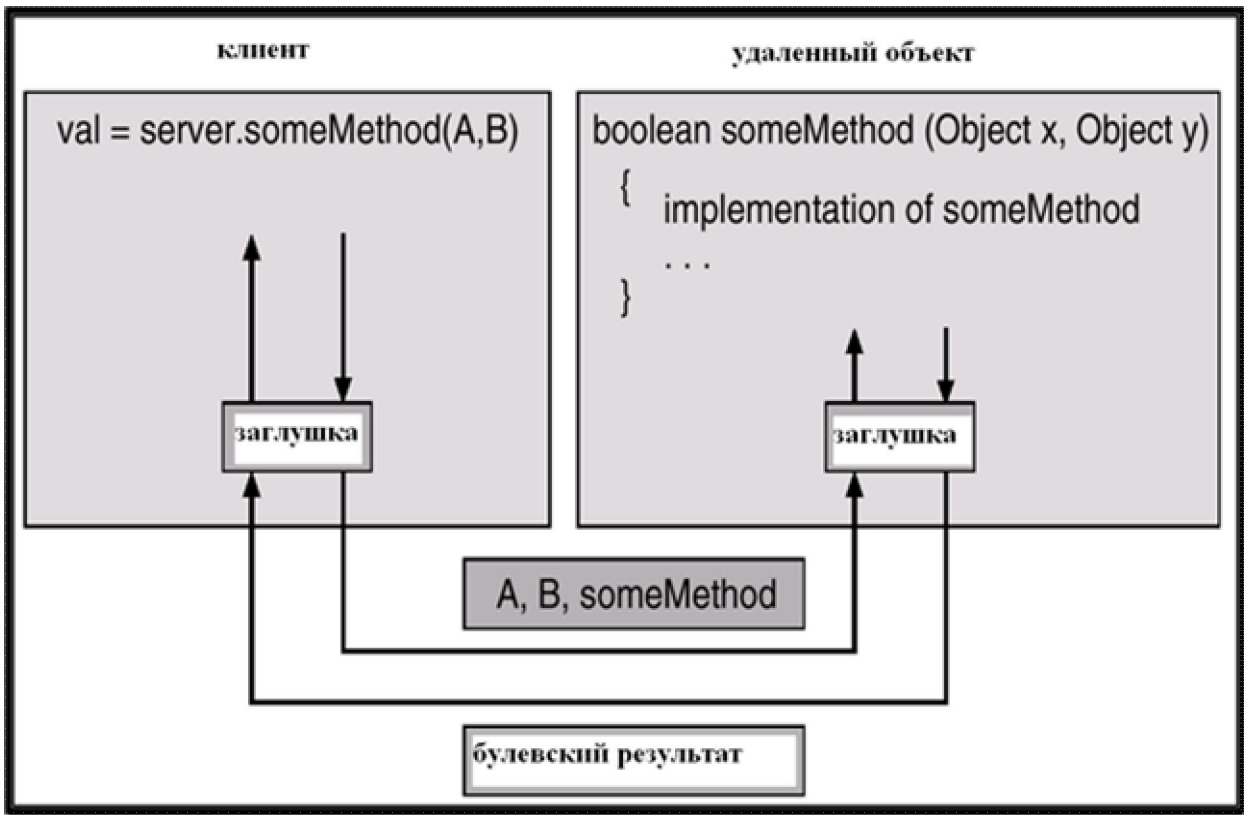

Рис. 9.4. Выстраивание параметров при удаленном вызове.

Ключевые термины

Send – операция отправки сообщения другому процессу.

Receive – операция получения сообщения от другого процесса.

Взаимодействующий (совместный) процесс – процесс, который может влиять на исполнение других процессов или испытывать их влияние.

Выстраивание (marshaling) – механизм преобразования параметров удаленной процедуры (метода) для их передачи по сети в виде последовательного потока.

Косвенная коммуникация (indirect communication) – способ взаимодействия процессов с помощью сообщений, при котором сообщения направляются и получаются через **почтовые ящики**, или **порты**.

Независимый процесс – процесс, никак не связанный с другими процессами, который не может влиять на исполнение других процессов или испытывать их влияние.

Непосредственная коммуникация (direct communication) – способ взаимодействия процессов с помощью сообщений, при котором они именуют друг друга явно – по именам или по адресам (указателям), которые указываются в вызовах коммуникационных примитивов.

Очередь сообщений (message queue) – системная структура (буфер) для хранения сообщений между процессами.

Переполнение буфера (buffer overrun) – ошибочное или преднамеренное превышения размера буфера, которое может привести к обращению в чужую область памяти и используется для внешних атак.

Подчиненный процесс – процесс, зависящий от процесса-родителя; уничтожается при уничтожении родительского процесса; процесс-родитель перед своим завершением должен ожидать завершения всех своих подчиненных процессов.

Почтовый ящик (порт) – системная структура, предназначенные для приема, хранения и передачи сообщений.

Производитель – потребитель (producer – consumer) - парадигма взаимодействия процессов, при которой процесс-производитель **(producer)** генерирует в некотором буфере информацию, которая используется процессом-потребителем (**consumer**).

Рандеву (rendezvous) – механизм коммуникации процессов, при котором оба процесса приостанавливаются до момента окончания передачи сообщения.

Сокет (socket) – метод клиент-серверного сетевого взаимодействия процессов, при котором информация передается через последовательный поток через порт с определенным номером.

Сопроцесс (coprocess, coroutine) – процесс, равноправно взаимодействующий с другими такими же процессами по управлению с помощью операций типа **resume (Q),** возобновляющих приостановленный процесс. Переходит в завершенное состояние операцией **detach**.

Удаленный вызов метода (Remote Method Invocation – RMI) – разработанный фирмой Sun объектно-ориентированный механизм Java-технологии для вызова метода Java на другом компьютере сети, аналогичный **удаленному вызову процедуры.**

Удаленный вызов процедуры (Remote Procedure Call – RPC) – разработанный фирмой Sun механизм вызова процедуры на другом компьютере локальной сети с использованием процедур-**заглушек** на клиенте и на сервере, передающих информацию и **выстраивающих** параметры и результат удаленной процедуры.

Краткие итоги

Процессы могут быть независимыми друг от друга и взаимодействующими. Преимущества взаимодействующих процессов – совместное использование данных, модульность, ускорение вычислений.

Дочерний процесс по отношению к процессу-родителю может быть подчиненным (зависящим от родителя), независимым, либо сопроцессом. Сопроцессы – равноправная группа процессов, взаимодействующая между собой операторами явного переключения управления.

Парадигма производитель – потребитель – классическая схема взаимодействия процессов: производитель генерирует информационные элементы в буфере, а потребитель использует их и удаляет из буфера. Буфер может быть неограниченным или иметь ограниченную длину.

Коммуникация процессов может осуществляться с помощью сообщений. Коммуникация бывает непосредственная (с явным указанием адресов или имен процессов-адресатов) и косвенная (через почтовые ящики).

При анализе коммуникации процессов весьма важны способ установления связи, число связей между двумя процессами, пропускная способность линии связи, длина сообщений, ненаправленный или двунаправленный (дуплексный) характер связи.

При косвенной связи могут возникнуть проблемы с недетерминизмом при получении сообщений, если несколько процессов используют общий почтовый ящик.

При коммуникации передача сообщений может быть синхронной (с блокировкой) или асинхронной (без блокировки).

С коммуникационной линией связывается очередь сообщений, которая может иметь нулевую длину (процессы взаимодействуют через механизм рандеву), ограниченную или теоретически неограниченную длину.

Основные виды клиент-серверной коммуникации процессов – сокеты, удаленные вызовы процедур и методов.

Сокеты предназначены для взаимодействия между клиентом и сервером в сетях TCP/IP через порт с определенным номером, при котором информация передается через последовательные потоки.

Удаленный вызов процедуры позволяет вызвать процедуру на другом компьютере локальной сети с использованием процедур-заглушек. осуществляющих выстраивание и передачу параметров и результатов.

Удаленный вызов метода – объектно-ориентированный механизм Java-технологии, аналогичный удаленному вызову процедуры.

10. Лекция: Потоки (threads) и многопоточное выполнение программ (multi-threading).

В лекции рассматриваются понятие потока (thread) и многопоточное выполнение (multi-threading); модели многопоточности; пользовательские потоки и потоки ядра; потоки в "Эльбрусе", Solaris, Linux, POSIX, Windows 2000, Java.

Введение

Многопоточность (multi-threading) – одна из наиболее актуальных тем в данном курсе. Актуальность данной темы особенно велика, в связи с широким распространением многоядерных процессоров. В лекции рассмотрены следующие вопросы:

- Исторический обзор многопоточности
- Модели многопоточного исполнения
- Проблемы, связанные с потоками
- Потоки в POSIX (Pthreads)
- Потоки в Solaris 2
- Потоки в Windows 2000/XP
- Потоки в Linux
- Потоки в Java и .NET.

Однопоточные и многопоточные процессы

К сожалению, до сих пор мышление многих программистов при разработке программ остается чисто последовательным. Не учитываются широкие возможности параллелизма, в частности, многопоточности. 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.

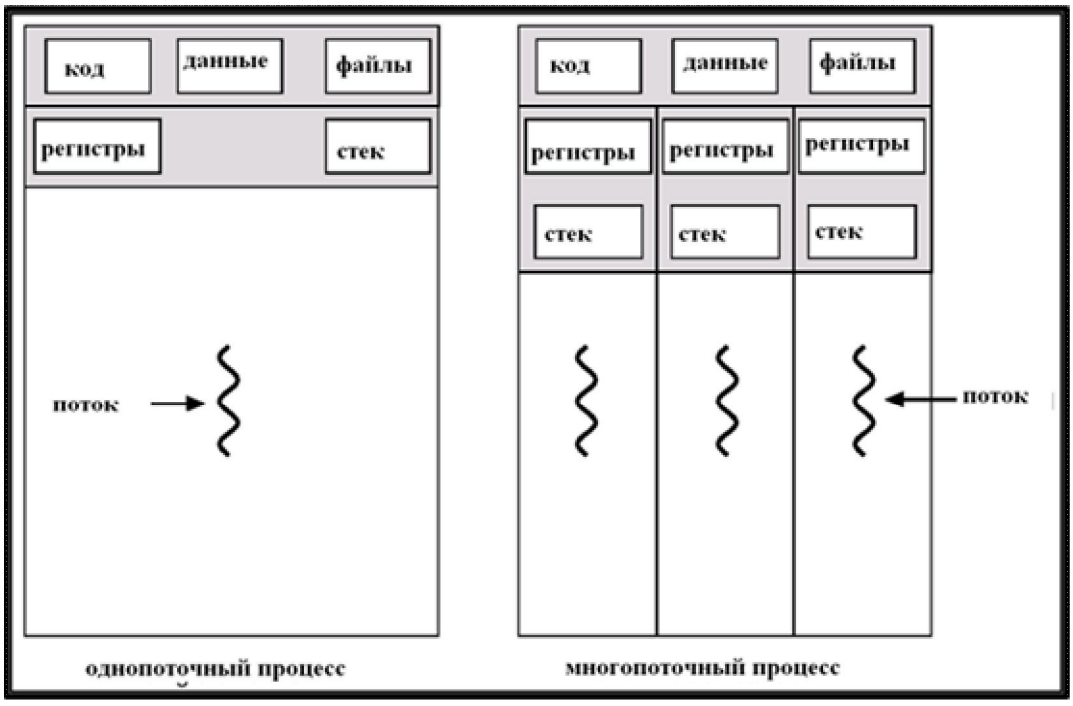

Рис. 10.1. Однопоточный и многопоточный процессы.

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**, работающих в общем пространстве виртуальной памяти. Благодаря многопоточности, не возникает больше неэффективных ситуаций, типичных для классической системы UNIX, в которой каждая команда shell исполнялась как **отдельный процесс**, причем в своем собственном адресном пространстве. В противоположность облегченным процессам, обычные процессы (имеющие собственное адресное пространство) часто называют **тяжеловесными** .
- **Использование общих ресурсов**. Потоки одного процесса используют общую память и файлы.
- **Экономия**. Благодаря многопоточности, достигается значительная экономия памяти. Также достигается 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.

Использование мультипроцессорных архитектур. Это особенно важно в настоящее время, в период широкого использования многоядерных гибридных и многопроцессорных систем. Именно многопоточность программ, основанная на многоядерности процессора, дает возможность, наконец, почувствовать реальные преимущества параллельного выполнения.

История многопоточности

Один из первых шагов на пути к широкому использованию многопоточности, был сделан в 1970-е годы советскими разработчиками компьютерной аппаратуры и программистами. МВК "Эльбрус-1", разработанный в 1979 году, поддерживал в аппаратуре и операционной системе эффективную концепцию процесса, которая была близка к современному понятию облегченного процесса. В частности, процесс в "Эльбрусе" однозначно характеризовался своим стеком. Иначе говоря, все процессы были облегченными и исполнялись в общем пространстве виртуальной памяти – других процессов в "Эльбрусе" просто не было!

Концепция многопоточности начала складываться с 1980-х гг. в системе UNIX и ее диалектах. Наиболее развита многопоточность была в диалекте UNIX фирмы AT&T, на основе которого, была разработана система Solaris. Все это отразилось и в стандарте POSIX, в который вошла и многопоточность, наряду с другими базовыми возможностями UNIX.

Далее, в середине 1990-х гг. была выпущена ОС Windows NT, в которую была также включена многопоточность.

Однако в разных операционных системах интерфейс прикладного программирования для многопоточности существенно отличались. Поэтому многопоточные программы, даже написанные на языках высокого уровня, оказались не переносимыми с одной платформы на другую, что, разумеется, создавало большие неудобства.

По-видимому, именно по причине различий в спецификациях и реализациях многопоточности в различных системах профессор Бьярн Страуструп **не** включил

многопоточность в созданный им язык C++, ставший столь популярным, и его базовый набор библиотек. Программисты на языке C++ были вынуждены по-прежнему использовать многопоточность на уровне системных вызовов и библиотек конкретных операционных систем.

Важный шаг вперед сделали авторы языка Java и Java-технологии, первая версия реализации которых была выпущена в 1995 г. Именно в Java впервые многопоточность была реализована на уровне конструкций языка и базовых библиотек. В частности, в Java введен класс **Thread**,представляющий поток, и операции над ним в виде специальных методов и конструкций языка.

Платформа .NET, появившаяся в 2000 г., предложила свой механизм многопоточности, который фактически является развитием идей Java.

Различие подходов к многопоточности в разных ОС и на разных платформах разработки программ сохраняется и до настоящего времени, что приходится постоянно учитывать разработчикам. Для прикладных программ мы рекомендуем реализовывать многопоточность на платформе Java или .NET, что наиболее удобно и позволяет использовать высокоуровневые понятия и конструкции. Однако в нашем курсе, посвященном операционным системам, мы, естественно, больше внимания уделяем системным вопросам многопоточности и ее реализации в операционных системах.

Пользовательские потоки и потоки ядра

Модели многопоточности. Реализация многопоточности в ОС, как и многих других возможностей, имеет несколько уровней абстракции. Самый высокий из них – пользовательский уровень. С точки зрения пользователя и его программ, управление потоками реализовано через библиотеку **потоков пользовательского уровня (user threads)** Подробнее конкретные операции над пользовательскими потоками будут рассмотрены немного позже. Пока отметим лишь, что существует несколько моделей потоков пользовательского уровня, среди которых:

- **POSIX Pthreads –** потоки, специфицированные стандартом POSIX и используемые в POSIX-приложениях (рассмотрены позже в данной лекции);
- **Mac C-threads** – пользовательские потоки в системе MacOS;
- **Solaris threads** – пользовательские потоки в ОС Solaris (рассмотрены позже в данной лекции).

Низкоуровневые потоки, в которые отображаются пользовательские потоки, называются **потоками ядра (kernel threads)**.Они поддержаны и используются на уровне ядра операционной системы. Как и подходы к пользовательским потокам, подходы к архитектуре и реализации системных потоков и к отображению пользовательских потоков в системные в разных ОС различны. Например, собственные модели потоков ядра со своей спецификой реализованы в следующих ОС:

- Windows 95/98/NT/2000/XP/2003/2008/7;
- Solaris;
- Tru64 UNIX;
- BeOS;
- Linux.

Существуют различные **модели многопоточности** – способы отображения пользовательских потоков в потоки ядра. Теоретически возможны (и на практике реализованы) следующие модели многопоточности:

- Модель **много / один (many-to-one)** – отображение нескольких пользовательских потоков в один и тот же поток ядра. Используется в операционных системах, не поддерживающих множественные системные потоки (например, с целью экономии памяти). Данная модель изображена на рис. 10.2.

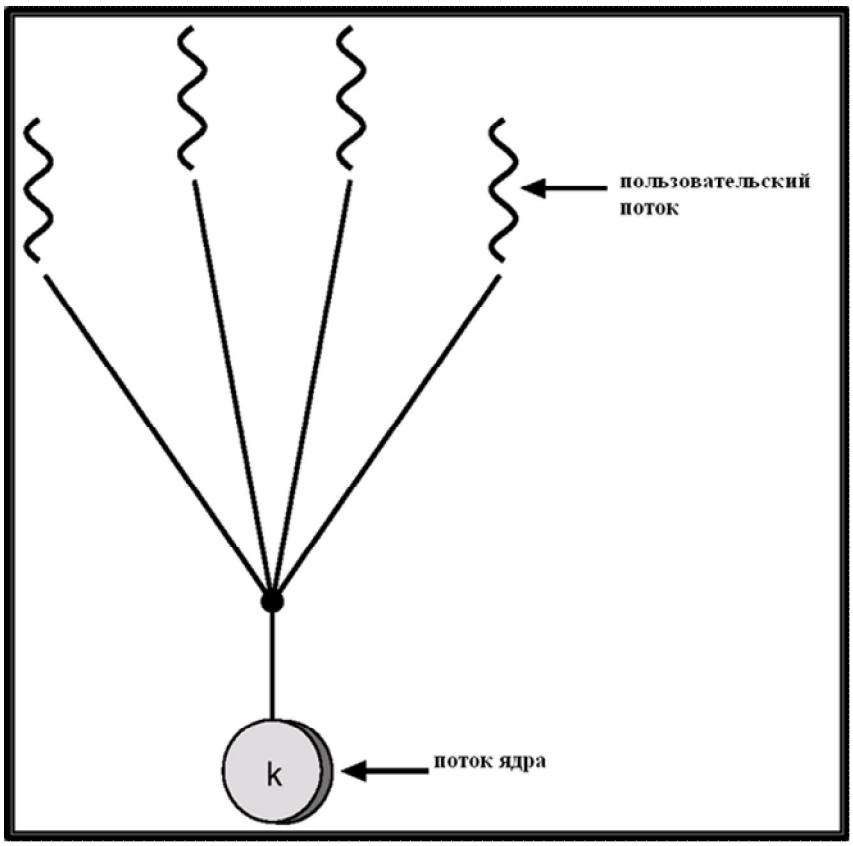

Рис. 10.2. Схема модели многопоточности "много / один".

- Модель **один / один (one-to-one)** – взаимно-однозначное отображение каждого пользовательского потока в определенный поток ядра. Примеры ОС, использующих данную модель, - Windows 95/98/NT/2000/XP/2003/2008/7; OS/2. Данная модель изображена на рис. 10.3.

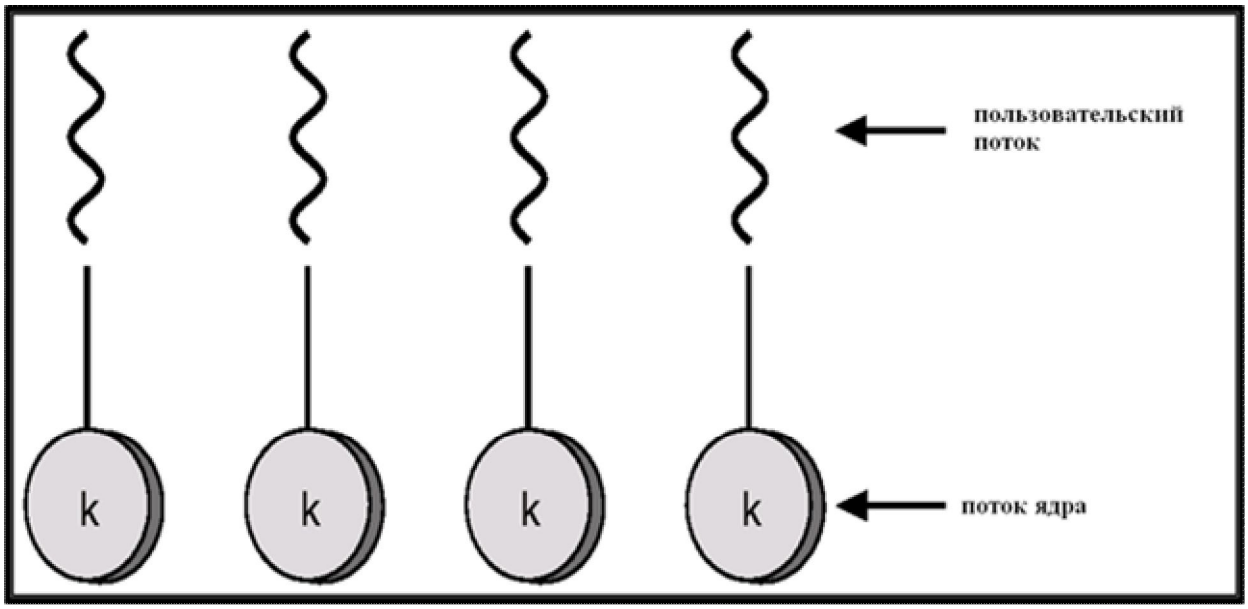

Рис. 10.3. Схема модели многопоточности "один / один".

- Модель **много / много (many-to-many)** – модель, допускающая отображение нескольких пользовательских потоков в несколько системных потоков. Такая модель позволяет ОС создавать большое число системных потоков. Характерным примером ОС,

использующей подобную модель, является ОС Solaris, а также Windows NT / 2000 / XP / 2003 / 2008 / 7 с пакетом **ThreadFiber**. Данная модель изображена на рис. 10.4.

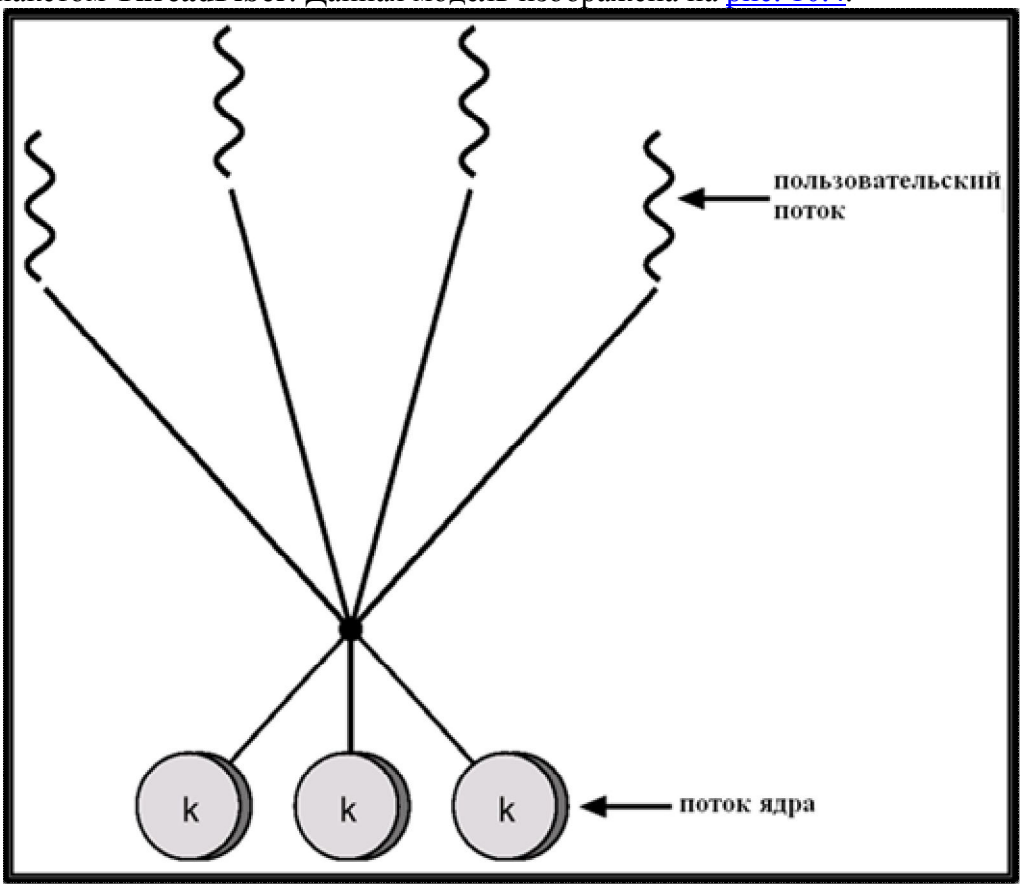

Рис. 10.4. Схема модели многопоточности "много / много".

Проблемы многопоточности

Многопоточность – весьма сложная, еще не полностью изученная и, тем более, не полностью формализованная область, в которой имеется много интересных проблем. Рассмотрим некоторые из них.

Семантика системных вызовов fork() и exec(). В классической ОС UNIX системный вызов **fork** создает новый "тяжеловесный" процесс со своим адресным пространством, что значительно "дороже", чем создание потока. Однако, с целью поддержания совместимости программ снизу вверх, приходится сохранять эту семантику, а многопоточность вводить с помощью новых системных вызовов.

Прекращение потоков. Важной проблемой является проблема прекращения потоков: например, если родительский поток прекращается, то должен ли при этом прекращаться дочерний поток? Если прекращается стандартный процесс, создавший несколько потоков, то должны ли прекращаться все его потоки? Ответы на эти вопросы в разных ОС неоднозначны.

Обработка сигналов. **Сигналы** в UNIX – низкоуровневый механизм обработки ошибочных ситуаций. Примеры сигналов: **SIGSEGV -** нарушение сегментации (обращение по неверному адресу, чаще всего по нулевому); **SIGKILL –** сигнал процессу о выполнении команды **kill** его уничтожения. Пользователь может определить свою процедуру-обработчик сигнала системным вызовом **signal**. Проблема в следующем: как распространяются сигналы в многопоточных программах и каким потоком они должны обрабатываться? В большинстве случаев этот вопрос решается следующим образом: сигнал обрабатывается потоком, в котором он сгенерирован, и влияет на исполнение только этого потока. В более современных ОС (например, Windows 2000 и более поздних версиях Windows), основанных на объектно-ориентированной методологии, концепция сигнала заменена более высокоуровневой концепцией **исключения (exception)**.Исключение распространяется по стеку потока в порядке, обратном порядку вызовов методов, и обраба тывается первым из них, в котором

система находит подходящий обработчик. Аналогичная схема обработки исключений реализована в Java и в .NET.

Группы потоков. В сложных задачах, например, задачах моделирования, при большом числе разнородных потоков, возникает потребность в их структурировании и помощью концепции **группы потоков** – совокупности потоков, имеющей свое собственное имя, над потоками которой определены групповые операции. Наиболее удачно, с нашей точки зрения, группы потоков реализованы в Java (с помощью класса **ThreadGroup**). Следует отметить также эффективную реализацию **пулов потоков (ThreadPool)** в .NET.

Локальные данные потока (thread-local storage - TLS) – данные, принадлежащие только определенному потоку и используемые только этим потоком. Необходимость в таких данных очевидна, так как многопоточность – весьма важный метод распараллеливания решения большой задачи, при котором каждый поток работает над решением порученной ему части. Все современные операционные системы и платформы разработки программ поддерживают концепцию локальных данных потока.

Синхронизация потоков. Поскольку потоки, как и процессы (могут использовать общие ресурсы и реагировать на общие события, необходимы средства их синхронизации. Эти средства подробно рассмотрены позже в данном курсе.

Тупики (deadlocks) и их предотвращение. Как и процессы, потоки могут взаимно блокировать друг друга (т.е. может создаться ситуация **deadlock**), при их неаккуратном программировании.

Потоки и процессы в Solaris

В ОС Solaris, как уже было отмечено, используется модель потоков **много / много**. Кроме того, в системе используется также уже известное нам понятие **облегченный процесс (lightweight process)** промежуточное между концепцией пользовательского потока и системного потока. Таким образом, в ОС Solaris каждый пользовательский поток отображается в свой облегченный процесс, который, в свою очередь, отображается в поток ядра; последний может исполняться на любом процессоре (или ядре процессора) компьютерной системы. Схема организации потоков в Solaris изображена на рис. 10.5.

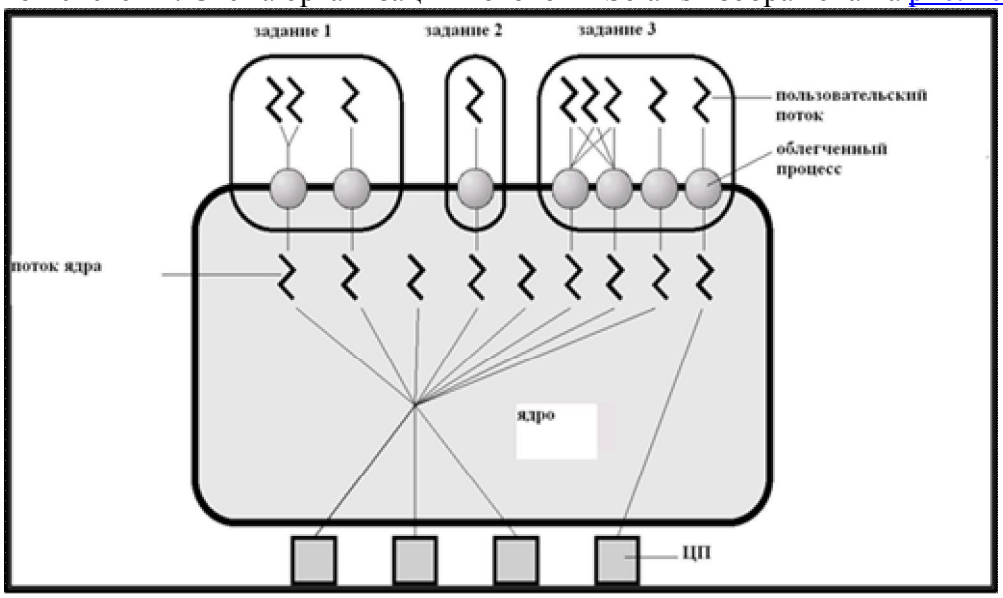

Рис. 10.5. Потоки в Solaris.

На рис. 10.6 изображена схема организации процесса в ОС Solaris.

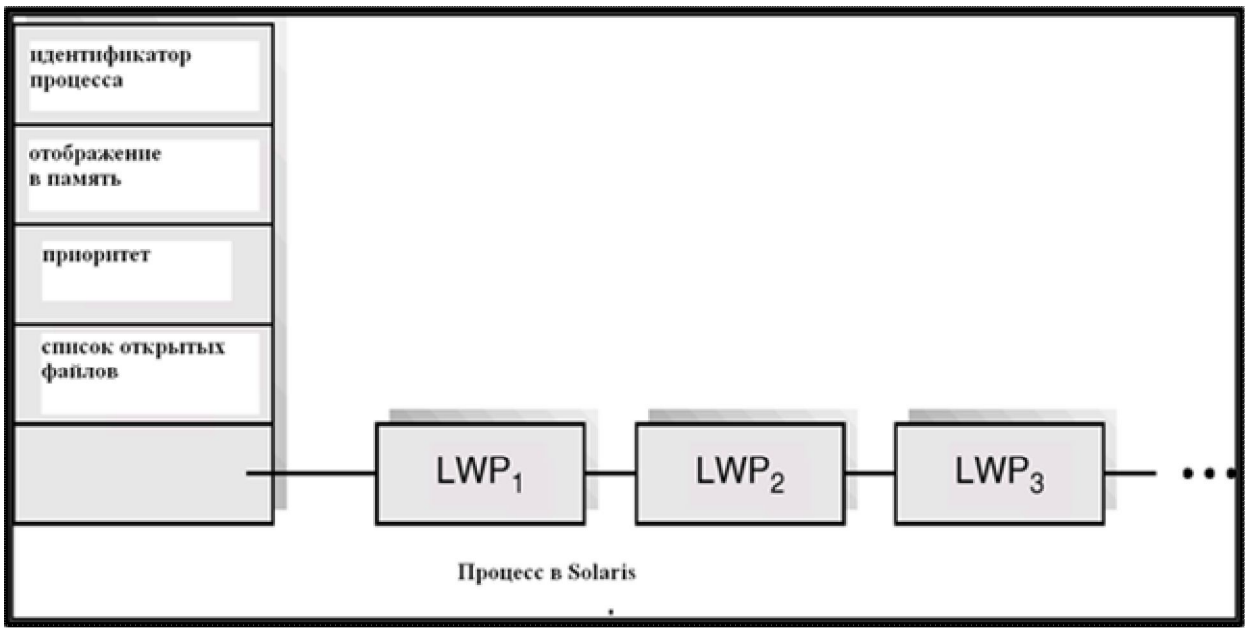

Рис. 10.6. Процессы в Solaris.

На схеме видно, что каждый процесс содержит, кроме стандартной информации блока управления процессом, также список всех своих облегченных процессов для управления ими.

Потоки в Windows 2000

Как уже отмечалось, в системе Windows реализована модель многопоточности "один / один". Каждый поток содержит:

- идентификатор потока (thread id);
- набор регистров
- отдельные стеки для пользовательских и системных процедур;
- область памяти для локальных данных потока (thread-local storage – TLS).

Потоки в Linux

В системе Linux потоки называются **tasks (задачами)**,а не **threads**. Поток создается системным вызовом clone(). Данный системный вызов позволяет дочерней задаче использовать общее адресное пространство с родительской задачей (процессом).

Потоки в Java

Как уже отмечалось, Java – первая платформа для разработки программ, в которой многопоточность поддержана на уровне языка и базовых библиотек. Потоки в Java могут быть созданы следующими способами:

- Как расширения класса **Thread**
- Как классы, реализующие интерфейс **Runnable**, который содержит единственный метод **run** – исполняемое тело потока.

Потоки в Java управляются JVM. Возможно создание групп потоков и иерархии таких групп.

Возможные состояния потоков в Java изображены на рис. 10.7. Подобно потокам в ОС, поток в Java создается и находится в состоянии **новый**, затем – **выполняемый** ; при вызове методов типа **wait, sleep** и др. поток переходит в состояние ожидания; при завершении метода **run** поток завершается.

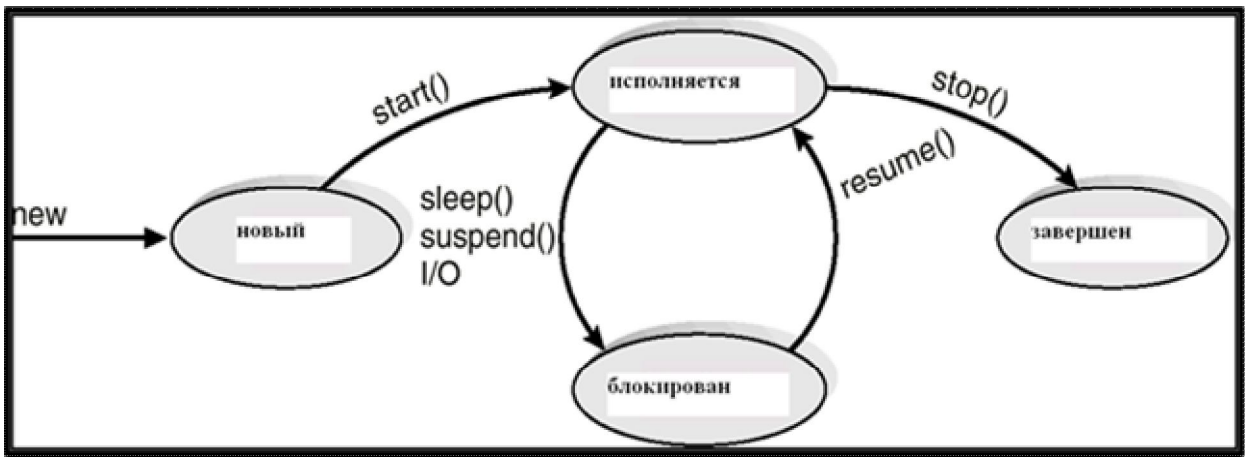

Рис. 10.7. Состояния потоков в Java.

Ключевые термины

Mac C-threads – пользовательские потоки в системе MacOS.

POSIX Pthreads – потоки, специфицированные стандартом POSIX и используемые в POSIX-приложениях.

Solaris threads – пользовательские потоки в ОС Solaris.

Thread – класс, представляющий поток, в языке Java.

Атрибуты потока – совокупность атрибутов POSIX-потока, описываемая типом pthread_attr_t.

Группа потоков (thread group) – совокупность потоков, имеющей свое собственное имя, над потоками которой определены групповые операции.

Дескриптор потока – ссылка на **POSIX-поток**, описываемая типом pthread_t.

Задача (task) – название потока в Linux.

Исключение (exception) – высокоуровневый механизм обработки ошибочных ситуаций в объектно-ориентированных языках и операционных системах.

Локальные данные потока (thread-local storage - TLS) – данные, принадлежащие только определенному потоку и используемые только этим потоком.

Модель многопоточности – способ отображения пользовательских потоков в потоки ядра.

Модель много / много - модель многопоточности,при которой различные **пользовательские потоки** могут быть отображены в различные **потоки ядра**.

Модель много / один - модель многопоточности, при которой несколько **пользовательских потоков** могут быть отображены в один **поток ядра**.

Модель один / один – модель многопоточности, при которой каждый **пользовательский поток** отображается в один определенный **поток ядра**.

Мьютекс (mutex) – аналог семафоров, обеспечивающий взаимное исключение, используемый в операционных системах.

Облегченный процесс (lightweight process) – процесс, работающий в общем пространстве виртуальной памяти с процессом-родителем.

Поток (thread) – запускаемый из какого-либо процесса более эффективный вариант параллельного процесса, выполняемый в том же адресном пространстве, что и процесс-родитель.

Поток пользовательского уровня (user thread) - высокоуровневый поток, операции над которым включены в интерфейс пользователя ОС.

Поток ядра (kernel thread) - низкоуровневый системный поток, поддержанный и использующийся на уровне ядра операционной системы; используется для реализации **потоков пользовательского уровня**.

Пул потоков (ThreadPool) – эффективный механизм структурирования потоков в группы в .NET.

Сигналы (в UNIX) – низкоуровневый механизм обработки ошибочных ситуаций.

"Тяжеловесный" (heavyweight) процесс – название классического процесса, работающего в собственном адресном пространстве, в противоположность **облегченному процессу**.

Условная переменная (conditional variable) - синхронизирующий объект, используемый в операционных системах, с операциями wait и signal.

Краткие итоги

Многопоточность (multi-threading) – современное направление программирования, особенно актуальное в связи с широким распространением параллельных компьютерных архитектур. Поток – особый вид процесса, выполняемый в общем адресном пространстве с процессом-родителем. Поток характеризуется своим стеком, потоком управления и значениями регистров. Облегченный процесс (lightweight process) – механизм, с помощью которого реализуются потоки в ОС.

Впервые понятие процесса, близкое современной концепции потока, было реализовано в системе "Эльбрус" в конце 1970-х гг. Многопоточность появилась в UNIX, затем – в Solaris и Windows NT. В различных ОС архитектуры библиотек поддержки многопоточности различаются. В Java-технологии, а вслед за ней – в .NET, впервые многопоточность была реализована на уровне языка и базовых библиотек.

Архитектура потоков – многоуровневая: потоки пользовательского уровня реализуются с помощью системных потоков (потоков ядра). Существуют различные модели многопоточности (способы отображения пользовательских потоков в системные) – один-один, один-много, много-один.

Многопоточность ставит ряд интересных проблем: семантика системных вызовов fork и exec; прекращение потоков; обработка сигналов; структуризация потоков в группы; поддержка локальных данных потока (TLS); синхронизация потоков; тупики (взаимная блокировка потоков) и их предотвращение.

POSIX threads (Pthreads) - стандартизация API для поддержки многопоточности для операционных систем типа UNIX. Поток характеризуется своим дескриптором и атрибутами. Для синхронизации потоков используются мьютексы и условные переменные.

Потоки в ОС Solaris отличаются тем, что явно присутствует понятие облегченного процесса, наряду с понятиями пользовательского и системного потоков. Каждый традиционный процесс хранит список созданных в нем облегченных процессов. Используется модель многопоточности "много-много".

В Windows 2000 используется модель многопоточности "один-один". Каждый поток содержит свой номер, набор регистров, отдельные стеки для пользовательских и системных процедур, локальную память потока (TLS).

В Linux потоки называются задачами (tasks) и создаются системным вызовом clone.

Потоки в Java поддержаны на уровне языка и базовых библиотек. Представляются объектами класса Thread и его подклассов. Управляются виртуальной машиной Java. Возможно создание групп потоков. Состояния потоков аналогичны используемым в ОС.

Источник: Курс лекций основан на курсе лекций, опубликованном в Негосударственное образовательном частном учреждении «Национальный Открытый Университет «ИНТУИТ», профессора кафедры информатики мат-мех. факультета СПбГУ ***Сафонова В. О.***