Основные функции и компоненты операционной системы

Операционная система – это комплект программного обеспечения, предназначенный для управления компьютером и обеспечивающий поддержку хранения, исполнения и разработки прикладных программ. Любая современная операционная система должна, как минимум, решать следующие задачи:  
1) хранение, загрузка и исполнение прикладных программ;  
2) организация файловой системы на устройствах долговременной памяти;  
3) планирование и динамическое перераспределение ресурсов компьютера;  
4) взаимодействие параллельных программ и синхронизация их работы;  
5) учет и разграничение полномочий пользователей системы;  
6) предоставление пользовательского интерфейса для работы с компьютером;  
7) защита данных от несанкционированного доступа, разрушения, других случайных или намеренных вредоносных действий, как во время выполнения программ, так и при хранении данных в файлах.  
При этом в любой современной операционной системе, независимо от архитектуры и методов реализации, можно выделить несколько относительно независимых базовых подсистем, совместно работающих для решения перечисленных выше задач.  
Состав операционной системы и связь ее базовых подсистем с решаемыми задачами, схематично показаны на следующем рисунке (Рисунок 1).

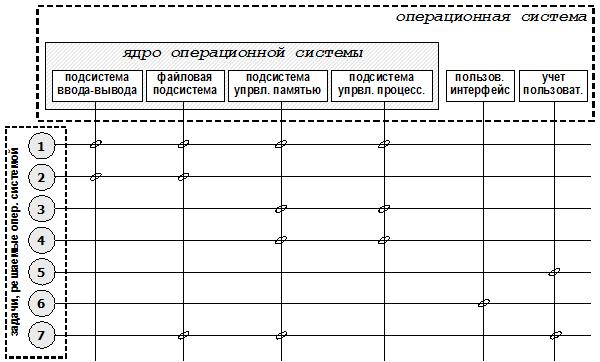


Рисунок 1 Состав и назначение основных подсистем операционной системы

Ряд подсистем операционной системы на рисунке (Рисунок 1) объединены в общий блок, названный ядро операционной системы.  
Понятие ядра является одним из наиболее важных понятий, как и роль ядра в операционной системе. Ядро объединяет в себе базовые компоненты операционной системы, во многом определяющие специфику и основную область применения данной операционной системы. Важнейшим свойством ядра, отличающим его от любых других системных или пользовательских программ, является то, что код ядра исполняется с максимально высоким уровнем привилегий – ему доступны все команды процессора и разрешен прямой доступ к любой аппаратуре компьютера. В конечном итоге, ядро изолирует все остальные программы от аппаратуры компьютера. Все остальные программы получают доступ к аппаратным ресурсам только через посредничество ядра операционной системы.  
Функциональность ядра во многом определяет функциональность всей операционной системы. Чем больше возможности ядра системы, тем более разносторонний сервис операционная система может в конечном итоге предоставить пользователю. Однако потребности различных категорий пользователей в универсальных операционных системах могут существенно различаться. Возможности, необходимые одним пользователям, могут совершенно не интересовать других, и наоборот. Ядро же должно удовлетворить всех, в результате код ядра становиться очень громоздким, оно начинает расходовать неоправданно много ресурсов, прежде всего памяти.  
Так как код ядра в силу своей специфики является аппаратно зависимым, громоздкое ядро универсальной операционной системы оказывается наиболее узким местом при переносе операционной системы на другую аппаратную платформу, например, на вновь разработанный или даже просто модернизированный компьютер.  
Наконец отметим еще, что от устойчивости кода ядра зависит устойчивость всей операционной системы, а отлаживать громоздкое ядро чрезвычайно сложно.  
Стремление найти кардинальное решение описанных проблем привело к появлению новой концепции построения операционной системы – концепции микроядра. При этом многие подсистемы, традиционно относимые к ядру, например, файловая система, выносятся из ядра. Код ядра предельно минимизируется и самостоятельно уже не обеспечивает даже базовой функциональности операционной системы. Зато его легко отладить и относительно легко переписать при переносе операционной системы на новую аппаратную платформу.  
В рамках нашего курса мы будем рассматривать микроядро более детально при изучении архитектуры операционных систем, а сейчас кратко рассмотрим назначение основных подсистем операционной системы.  
  
Подсистема ввода-вывода  
  
Подсистема ввода-вывода принимает запросы на передачу или прием данных от всех выполняющихся программ, как прикладных, так и системных, упорядочивает их и переадресует внешним устройствам с учетом их аппаратных особенностей.  
Можно выделить две основные функции подсистемы ввода-вывода:  
- изоляция всех других компонентов операционной системы и прикладных программ от особенностей аппаратной реализации внешних устройств компьютера;  
- организация бесконфликтного и эффективного использования внешних устройств всеми исполняющимися программами.  
Фактически, задачу изоляции от аппаратных особенностей внешних устройств решают драйверы – специальные подпрограммы в составе подсистемы ввода-вывода, которые непосредственно взаимодействуют с внешними устройствами, при этом для каждого внешнего устройства необходим собственный драйвер. Подсистема ввода-вывода любой современной операционной системы организована так, чтобы набор драйверов можно было легко заменять, обеспечивая работу прикладных программ и самой операционной системы при различных аппаратных конфигурациях компьютера.  
Построение и работа подсистемы ввода-вывода, включая драйверы, в дальнейшем будет рассмотрено более подробно.  
Как показывают многочисленные замеры и эксперименты, программы, кроме чисто вычислительных, большую часть своей жизни проводят в состоянии ожидания завершения операций ввода-вывода. Поэтому от эффективности работы подсистемы ввода-вывода существенно зависит эффективность работы всех остальных программ. Для повышения скорости выполнения запросов ввода-вывода, подсистема ввода-вывода широко использует такие приемы, как переупорядочивание запросов ввода-вывода или кэширование данных. В совокупности, подобные меры могут поднять производительность подсистемы ввода-вывода более, чем в 100 раз. В дальнейшем, при детальном изучении подсистемы ввода-вывода, основные приемы повышения производительности будут рассмотрены довольно подробно.  
  
Файловая подсистема  
  
Файловая подсистема обеспечивает пользователя удобным, аппаратно независимым интерфейсом для упорядоченного хранения пользовательских программ и данных на внешних устройствах долговременной памяти.  
В современных ЭВМ в качестве основного устройства долговременной памяти обычно используется накопитель на жестких магнитных дисках. Физически, данные на диске адресуются подобно ячейкам массива – по номерам блоков данных. Современный диск содержит миллиарды таких блоков, поэтому человеку практически невозможно работать с сырыми данными на диске. Фактически, файловая система позволяет всего лишь обращаться по осмысленному символическому имени к группе дисковых блоков, представляющих логически связанные данные, но это делает работу с данными на диске не только возможной, но и весьма удобной для человека.  
Кроме того, благодаря файловой подсистеме пользователи совершенно единообразно работают с различными типами внешних устройств долговременной памяти. Причем пользователи централизованной файловой системы, представляющей все каталоги от единого корня, не связанного с физическим носителем, часто даже не знают, на каком именно физическом устройстве содержатся те или иные данные.  
Для выполнения своей функции, файловая система должна, как минимум, постоянно вести учет свободных и занятых блоков на диске, отслеживать принадлежность дисковых блоков к конкретным файлам и порядок следования блоков в файле.  
Кроме того, современная файловая система должна быть устойчива к случайным сбоям, таким как внезапное выключение компьютера, или повреждение части данных на диске, другими словами, потеря нескольких блоков данных не должна приводить к разрушению всей файловой системы и, как следствие, потере всех данных.  
Интересно отметить, что во многих современных операционных системах интерфейс файловой подсистемы используется не только для доступа к данным на дисках, но и для всех операций ввода-вывода. Тогда взаимодействие прикладных программ с клавиатурой, монитором, принтером или другим периферийным оборудованием выглядит как запись или чтение некоторых файлов.  
  
Подсистема управления памятью  
  
Подсистема управления памятью в современной операционной системе решает следующие основные задачи:  
- бесконфликтное распределение памяти между исполняющимися программами;  
- защита памяти от несанкционированного доступа.  
В любой современной операционной системе эти задачи полностью решаются средствами виртуальной памяти, механизмы работы которой будут весьма подробно рассмотрены в данном курсе через некоторое время.  
Подсистема управления процессами  
Выполнение любой программы в рамках любой современной операционной системы начинается с создания процесса – специального объекта операционной системы, обеспечивающего выполнение программы, таким образом, подсистема управления процессами в конечном итоге отвечает за выполнение программ.  
Подсистема управления процессами в современной операционной системе решает довольно обширный набор задач, которые удобно представить в виде диаграммы:

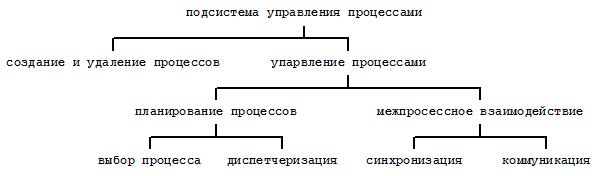


Рисунок 2 Задачи подсистемы управления процессами

Таким образом, подсистема управления процессами решает два класса задач: создание и удаление процессов, и управление существующими процессами.  
Задача создания процесса сводится к размещению в памяти и инициализации структур данных, описывающих процесс. Задача удаления процесса зачастую сводится к освобождению памяти и ресурсов, удерживаемых процессом.  
Более интересными представляются задачи, связанные с управлением процессами. Дадим краткую характеристику этим задачам.  
Управление процессами требует решения двух классов задач: задач планирования процессов и задач межпроцессного взаимодействия.  
  
Планирование процессов  
  
Подсистема планирования процессов решает две независимые задачи:  
1. определяет, какой из готовых к выполнению процессов, когда, и на какой интервал времени следует передать процессору для выполнения, т.е. выполняет собственно планирование процессов;  
2. выполняет переключение процессов, т.е. снимает с исполнения текущий процесс и ставит на исполнение запланированный процесс. Такое переключение процессов называют диспетчеризацией.  
Заметим, что задача планирования является аппаратно независимой. Планирование процессов осуществляется исходя из степени важности процессов для системы или пользователя и временной статистики их исполнения, т.е. без учета каких-либо аппаратных особенностей компьютера.  
Задача диспетчеризации, наоборот, является аппаратно зависимой, т.к. переключение процессов связано с сохранением и загрузкой регистров конкретного процессора.  
Межпроцессное взаимодействие  
В рамках межпроцессного взаимодействия необходимо решать две задачи: задачу синхронизации процессов и задачу межпроцессной коммуникации.  
Синхронизация процессов является одним из важнейших аспектов функционирования многозадачной операционной системы. Действительно, в многозадачной операционной системе часто возникает ситуация, когда некоторый процесс не может выполняться дальше, пока не будут завершены определенные действия другого процесса. Например, один процесс, назовем его процесс-писатель, помещает данные в некоторый буфер, а другой процесс – процесс-читатель, считывает данные из этого буфера. Очевидно, что процесс-писатель не должен помещать в буфер новые данные, пока процесс-читатель не считает ранее записанные данные. Как будет показано в нашем курсе далее, осуществить необходимую синхронизацию на уровне пользовательских процессов, без специальных средств операционной системы, не представляется возможным.  
Что касается межпроцессной коммуникации, то каждый процесс выполняется в рамках собственного виртуального адресного пространства, закрытого для других процессов, поэтому межпроцессные коммуникации могут быть реализованы только при наличии специальной поддержки со стороны операционной системы.  
Конкретные механизмы межпроцессного взаимодействия будут подробно изучаться в нашем курсе в дальнейшем.  
  
Пользовательский интерфейс  
  
Функциональное назначение пользовательского интерфейса состоит в том, чтобы предоставить пользователю удобный, простой в изучении использовании инструмент для управления компьютером, операционной системой и прикладными программами.  
Большинство современных операционных систем поддерживают более или менее развитый графический интерфейс пользователя. Безусловно, это наиболее удобный для пользователя способ взаимодействия с системой, но при чрезмерном усложнении интерфейса, вычислительные затраты на поддержку графического интерфейса могут стать вполне соизмеримыми с затратами на полезные вычисления.  
Кроме того, серьезной проблемой при реализации сложного графического интерфейса является скорость отображения информации на экране компьютера. Если при работе с компьютером будет заметна задержка отображения элементов интерфейса, видна последовательность перерисовки элементов на экране, мерцания, то работать с таким интерфейсом будет неудобно. Чтобы повысить скорость работы интерфейса, его можно включить в состав ядра операционной системы, тогда упростится и ускорится доступ к аппаратным средствам, но при этом мы получаем еще более громоздкое, сложное в отладке и менее надежное ядро.  
Поэтому разработка графического интерфейса неизбежно требует поиска наиболее удачного компромисса между возможностями интерфейса и скоростью его работы.  
В рамках данного курса мы не будем подробно изучать принципы построения и работы графических интерфейсов, а лишь кратко коснемся этой темы.  
  
Подсистема учета пользователей  
  
Подсистема учета пользователей необходима в операционной системе для защиты данных пользователей и сохранения их индивидуальных настроек.  
Первоначально, подсистема учета пользователей ориентировалась исключительно на решение задачи разграничения прав доступа, но в последствии, по мере развития графических интерфейсов пользователя, появилась еще и необходимость сохранять для каждого пользователя его индивидуальные настройки (цветовая схема, состав и расположение пунктов меню и иконок и т.п.).  
В рамках данного курса мы только кратко коснемся организации учета пользователей в операционных системах, а основное внимание будет сосредоточено на изучении алгоритмов управления ресурсами.  
  
Понятие виртуальной машины  
  
Исполнение прикладных программ – это основное назначение любого компьютера. В зависимости от решаемых задач, прикладные программы могут быть совершенно различными, но любая прикладная программа для своего выполнения нуждается в получении от операционной системы определенного сервиса, такого как чтение или запись файлов, вывод полученных результатов на печать или на экран, ввод символов с клавиатуры, синхронизация с ходом выполнения других программ и т.п.  
Прикладные программы обращаются к операционной системе за необходимыми услугами через интерфейс прикладных программ операционной системы. При этом прикладные программы совершенно не интересует, каким образом будут выполняться их запросы, как устроен компьютер и какова структура операционной системы.  
Прикладные программы могут одинаково успешно выполняться при различной аппаратной конфигурации компьютера и под управлением различных операционных систем, главное, чтобы не изменился интерфейс прикладных программ.  
При этом можно утверждать, что прикладные программы исполняются не на реальном компьютере с его аппаратной архитектурой, а на некоторой виртуальной машине, свойства которой определяются возможностями операционной системы.  
Заметим теперь, что операционная система изолирует параллельно выполняемые программы друг от друга, при этом выполнение программы (кроме, конечно, скорости ее работы) не зависит от того, какие еще программы запущены в данный момент на данном компьютере. Следовательно, каждая запущенная программа получает в свое распоряжение свою собственную виртуальную машину.  
Виртуальная машина для прикладных программ строится операционной системой на основе возможностей реальной машины, при этом возможности виртуальной машины могут превосходить возможности реальной машины, за счет того, что операционная система может имитировать для прикладной программы работу аппаратуры, которая на самом деле отсутствует в составе данного компьютера. Устройства, которые на самом деле не представлены в составе компьютера, но имитируются для программ операционной системой, называют виртуальными устройствами.  
На современном уровне развития техники, для того, чтобы разработать компьютерную программу, требуется знать только характеристики и правила работы той виртуальной машины, на которой планируется выполнение программы. Знание характеристик и устройства реального компьютера не требуется. Такой подход обеспечивает для разработчика программ следующие удобства и преимущества:  
- виртуальная машина изолирует программы от аппаратных особенностей ЭВМ, в результате программы становятся переносимыми, т.е. могут без каких-либо переделок запускаться на различных аппаратных платформах;  
- виртуальная машина изолирует запущенные программы от других программ, что с одной стороны обеспечивая требуемый уровень безопасности и надежности, а с другой стороны существенно упрощает разработку программ;  
- уровень и виды услуг, предоставляемых виртуальной машиной, лучше согласуются с потребностями прикладных программ, чем услуги реальной машины. Например, виртуальная машина представляет данные на диске в виде именованных файлов, в то время как реальная машина представляет дисковые данные просто как набор пронумерованных блоков данных, без какой-либо логической связи между ними;  
- виртуальная машина обеспечивает более простой и более мощный интерфейс ввода-вывода, чем может обеспечить реальная машина. В частности, прикладные программы запрашивают услуги ввода вывода в терминах прочитать/записать данные, без конкретизации физических параметров реальных устройств, таких как положение головок дискового накопителя или параметры модуляции модема.  
  
Понятие прерывания  
  
Прерывание – это временное прекращение работы программы, вызванное внешними, по отношению к этой программе причинами. Внешние причины – это значит, что в коде программы нет явной команды на ее приостановку.  
Прерывание всегда неожиданно для программы, и она не может подготовиться к нему и не может сама обеспечить восстановление своей работы после прерывания. Но с другой стороны, прерывание – это все же временное прекращение работы программы, и после завершения прерывания выполнение программы должно быть восстановлено. При этом с точки зрения программы такое восстановление происходит автоматически, более того, программа даже не знает, что она прерывалась.  
Физически, прерывания реализуются совместными действиями процессора и операционной системы. Процессор фиксирует прерывание и автоматически переходит от выполнения текущей программы к выполнению новой программы – обработчика прерывания. При этом в процессоре обычно определено несколько типов прерываний, и с каждым порыванием всегда связан свой собственный обработчик. Операционная система предоставляет обработчики прерывания для каждого типа прерываний и обеспечивает восстановление исполнения прерванной программы.  
Обработка прерываний в операционной системе будут более подробно рассмотрены в рамках нашего курса. Пока только отметим, что механизм прерываний лежит в основе самых фундаментальных механизмов операционной системы, таких как ввод-вывод, многозадачность или интерфейс прикладных программ.  
Пользовательский и привилегированный режимы процессора  
Одной из основных задач операционной системы является защита пользовательских программ и данных, а также самой себя, от вредоносных действий других программ.  
Проблема защиты актуальна даже в наиболее простой однозадачной операционной системе, когда необходимо защищать программы и данные операционной системы от случайных или намеренных повреждений со стороны пользовательских программ. Но еще более остро такая проблема стоит в многозадачных системах, когда необходимо еще защищать различные пользовательские программы друг от друга.  
В современных операционных системах необходимый уровень защиты программного кода и данных достигается благодаря использованию виртуальной памяти, которая полностью изолирует исполняющиеся программы друг от друга. Следовательно, для обеспечения защиты данных, достаточно запретить прикладным программам прямой доступ к аппаратуре компьютера, как минимум, к средствам управления виртуальной памятью, чтобы программы не могли бы действовать в обход средств операционной системы. Но системные программы по-прежнему должны иметь доступ к аппаратуре, и непосредственно управлять этой аппаратурой.  
Очевидно, что разрешить указанное противоречие чисто программными методами, без использования поддержки со стороны аппаратных средств невозможно, и такая поддержка предоставляется непосредственно процессором.  
В современных процессорах предусмотрено как минимум два уровня выполнения программ, которые различаются доступным для программ набором команд. Команды, обеспечивающие прямой доступ к аппаратуре, запрещение и разрешение прерываний, установку обработчиков прерываний или управление регистрами виртуальной памяти, являются привилегированными командами, и доступны программам только в специальном, привилегированном режиме работы процессора. Привилегированный режим работы процессора называют еще режимом ядра или режимом супервизора.  
В режиме ядра исполняются только некоторые программы операционной системы, те, которым действительно необходим прямой доступ к аппаратным средствам машины. А все прикладные программы исполняются в обычном, не привилегированном режиме, который называют пользовательский режим или режим задачи.  
Очевидно, что переход в режим ядра должен осуществляться таким образом, чтобы программы пользовательского режима не могли бы самостоятельно переключиться в режим ядра, иначе весь смысл выделения привилегированного режима пропадает.  
В современных процессорах механизм переключения в режим ядра из режима задачи чаще всего реализован через механизм прерываний. Фактически, при возникновении любого пребывания процессор автоматически переходит в режим ядра и только затем начинает выполнять код программы обработчика прерывания. Заметим теперь, что все обработчики прерываний установлены операционной системой, обычно еще на этапе ее загрузки и не могут быть бесконтрольно изменены никакими прикладными программами, т.к. команда смены обработчика является привилегированной.  
Таким образом, гарантируется, что только программы операционной системы смогут выполняться в привилегированном режиме работы процессора. Если же программа пользовательского режима вдруг попытается выполнить привилегированную команду, например, сменить обработчик прерывания, то процессор зарегистрирует ошибку, т.н. исключительную ситуацию, или просто исключение, и сразу же передаст управление специальной программе в составе операционной системы – обработчику исключений.  
Благодаря разграничению уровня привилегий между режимом ядра и режимом задачи, программная система, которая первой получит процессор в свое распоряжение, может установить полный контроль над всей вычислительной системой, и препятствовать несанкционированным действиям со стороны любых других программ. Заметим, что в реальной ситуации, операционная система получает управление на этапе начальной загрузки компьютера, раньше любой пользовательской программы, следовательно, она получает потенциальную возможность для полного контроля над компьютером.  
  
Понятие программы и процесса  
  
Как мы уже неоднократно говорили, одной из основных задач операционной системы является запуск и максимально эффективное исполнение пользовательских программ. Рассмотрим теперь более строго, что же такое программа, выполнение программы, и какие действия должна предпринять операционная система, чтобы обеспечить запуск и выполнение пользовательской программы.  
Текущее состояние программы и инструкция для следующего шага выполнения программы определяются предыдущим состоянием программы и инструкцией, выполненной на текущем шаге.  
Результатом выполнения программы является вектор состояния, полученный после последнего шага программы.  
Вектор состояния программы на -м шаге выполнения программы определяется не только программой , но также начальным состоянием программы и первой инструкцией программы , то для получения однозначных результатов работы программы, перед запуском программы необходимо установить начальное состояние программы и указать номер первой исполняемой инструкции программы – точку входа в программу.  
Предыстория переходов, приведшая программу в какое-то состояние, не имеет значения для выбора следующего состояния. Следовательно, выполнение программы можно в любой момент приостановить, а затем возобновить, восстановив текущее состояние и номер следующей инструкции.  
Исходя из описанной модели, легко определить действия, которые должна выполнить операционная система для запуска пользовательской программы:  
1) выделить память для размещения инструкций и переменных программы;  
2) загрузить последовательность инструкций программы в память;  
3) инициализировать переменные, определяющие начальное состояние программы.  
После запуска программы, операционная система должна управлять процессом выполнения программы вплоть до ее завершения. Возможно, что по каким-то причинам операционная система должна будет приостановить выполнение текущей программы. Чтобы затем иметь возможность возобновить ее выполнение, операционная система перед остановкой программы должна, как минимум, сохранить номер следующей инструкции программы и значения всех переменных.  
Кроме того, в памяти компьютера может находиться несколько программ одновременно, и операционная система должна уметь различать их.  
Поэтому, подготавливая запуск программы, операционная система также должна:  
4) выделить дополнительную память для хранения информации о текущем состоянии запущенной программы, включая номер текущей инструкции.  
Только завершив рассмотренные подготовительные действия, операционная система может запустить выполнение программы, для чего необходимо:  
5) передать управление на первую инструкцию программы – в точку входа.  
Программу, размещенную в памяти вместе с ее переменными и служебными данными операционной системы, и исполняющуюся под управлением операционной системы, называют процессом. При этом, действия, которые операционная система выполняет в ходе подготовки к запуску программы, называют созданием процесса.  
Еще раз отметим различие между программой и процессом, т.к. эти понятия часто путают. Программа – это просто набор инструкций. Процесс – это внутренний объект операционной системы, связанный с выполнением программы и обеспечивающий поддержку ее выполнения. При этом операционная система может создать несколько процессов на основании одной и той же программы. С другой стороны, для выполнения единственной программы может создаваться более одного процесса.  
Процесс – это один из важнейших объектов операционной системы, действительно:  
- все запросы на выделение и освобождение ресурсов поступают операционной системе от имени процессов, а ресурсы выделяются процессам;  
- права доступа к данным в памяти компьютера и на дисках фактически определяются и проверяются для процессов;  
- планирование работы выполняется операционной системой на уровне процессов, т.е. операционная система решает, какой процесс должен выполняться в данное время, а какой следует приостановить;  
- запуск и завершение любой программы связан с созданием и уничтожением соответствующего процесса;  
- все программы, подпрограммы и функции могут выполняться только в контексте какого-нибудь процесса.  
  
Принципы реализации многозадачности  
  
Многозадачность – это способность операционной системы выполнять более одного процесса одновременно.  
Существует два принципиально отличающихся способа организации многозадачности в операционных системах: корпоративная многозадачность, и вытесняющая многозадачность. Рассмотрим оба подхода по порядку.  
Корпоративная многозадачность  
Очевидно, что если в компьютере один процессор, то реально может выполняться только один процесс, все остальные процессы, даже если они находятся в памяти и готовы к исполнению, будут приостановлены. Многозадачная операционная система создает иллюзию одновременного выполнения всех запущенных процессов, поочередно выполняя небольшие фрагменты каждого. Другими словами, запущенные процессы поочередно используют ресурс процессора в режиме разделения времени. Заметим здесь, что наряду с пользовательскими процессами, операционная система должна запустить свои собственные процессы, предназначенные для планирования выполнения пользовательских процессов и для поддержания пользовательского интерфейса. При этом процессы операционной системы будут конкурировать за процессор и другие ресурсы наряду с пользовательскими процессами.  
С другой стороны, рассматривая выполнение программы (см. выражение (1)), мы отметили, что следующая инструкция для выполнения определяется самой же исполняемой программой. Это означает, что нельзя прервать выполнение процесса, пока сам исполняющийся процесс не передаст управление другому процессу. Тогда для организации многозадачности может быть предложен следующий подход:  
- получив управление, процесс выполняет некоторое количество инструкций своей программы, после чего добровольно передает управление процессу-планировщику операционной системы;  
- планировщик, получив управление, выбирает один из процессов, готовых к исполнению, и передает ему управление.  
Рассмотренный подход организации многозадачности известен как корпоративная многозадачность. Корпоративную многозадачность еще называют согласующей или невытесняющей многозадачностью.  
Так как пользовательский процесс сам определяет момент вызова планировщика и, следовательно, момент передачи управления другому процессу, то говорят, что в операционной системе с корпоративной многозадачностью функции планировщика распределены между операционной системой и пользовательскими процессами.  
Исполнение процессов в режиме корпоративной многозадачности схематически показано на следующем рисунке (Рисунок 3).

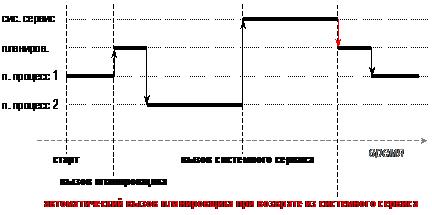


Рисунок 3 Работа в режиме корпоративной многозадачности

Пользовательские процессы в ходе своей работы неизбежно обращаются к операционной системе за услугами ввода-вывода, для синхронизации выполнения с другими процессами или за другим системным сервисом. В таких случаях говорят, что процесс инициировал системный вызов.  
Для повышения эффективности использования центрального процессора, системные вызовы реализованы в операционной системе так, что всякий раз при завершении системного вызова, процесс-планировщик вызывается автоматически (см. Рисунок 3).  
Действительно, если процесс обращается к операционной системе за услугами ввода-вывода, выполнение процесса обычно не может быть продолжено немедленно после обращения к операционной системе. Например, при обращении к файлу необходимо дождаться позиционирования головок дискового накопителя, при обращении к клавиатуре – пока пользователь нажмет клавишу. Крайне нерационально в такой ситуации останавливать центральный процессор и ждать завершения ввода-вывода. Гораздо более целесообразно передать управление другому процессу, готовому к немедленному выполнению.  
  
Недостатки корпоративной многозадачности  
  
Несмотря на простоту реализации и достаточно высокую эффективность работы, корпоративная многозадачность имеет один существенный недостаток: если процесс не захочет добровольно передать управление планировщику, другие процессы, запущенные в системе, в том числе системные процессы, никогда не получат управления и не будут выполняться. Но, если не будут функционировать средства пользовательского интерфейса, то операционная система будет выглядеть зависшей.  
Этот недостаток в некоторой степени устраняется автоматическим вызовом планировщика при завершении системного вызова. Действительно, всем процессам необходимо использовать ввод-вывод для нормальной работы, и даже если процесс не вызывает явно планировщик, когда он обратится к операционной системе за услугами ввода-вывода, планировщик будет вызван автоматически. Но и такое решение не является абсолютно надежным, в частности:  
- вычислительный процесс, выполняющий сложные математические расчеты, может очень долго не обращаться за услугами ввода-вывода и операционная система не сможет вытеснить его с выполнения;  
- если процесс зациклился, то нет никакой возможности прервать его, и система полностью зависает.  
Таким образом, многозадачная операционная система, основанная на корпоративной многозадачности, не способна гарантировать успешное завершение и сохранение результатов работы даже для корректных процессов. Поэтому корпоративная многозадачность редко используется в современных операционных системах.  
  
Вытесняющая многозадачность  
  
Чтобы устранить недостатки, присущие корпоративной многозадачности, необходимо, чтобы операционная система могла бы по своему усмотрению в любое время снять с выполнения любой процесс. Механизм планирования при этом должен быть полностью локализован в рамках операционной системы, и, в отличие от механизма планирования при корпоративной многозадачности, не должен требовать запуска со стороны пользовательских процессов. В противовес корпоративной многозадачности, такой подход называют вытесняющей или истинной многозадачностью.  
Для организации в составе операционной системы вытесняющей многозадачности используется прерывание от системного таймера. При этом управление периодически, через небольшие интервалы времени, передается планировщику, как показано на следующем рисунке (Рисунок 4).

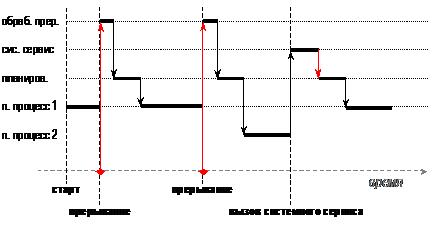


Рисунок 4 Работа в режиме вытесняющей многозадачности

В режиме вытесняющей многозадачности операционная система работает следующим образом:  
1) При возникновении аппаратного прерывания от системного таймера управление получает обработчик прерывания, который сразу же передает управление процессу-планировщику, т.о. планировщик получает управление без явного вызова со стороны исполняющегося процесса, и даже без согласия этого процесса.  
2) Планировщик определяет следующий процесс для выполнения и величину кванта времени, в течение которого планируется выполнение этого процесса. Затем планировщик программирует таймер на этот интервал времени и передает управление выбранному процессу. Таким образом, планировщик гарантирует себе, что снова получит управление не позднее, чем через заданный интервал времени.  
3) Планировщик может получить управление и до истечения установленного кванта времени, если исполняющийся процесс инициирует системный вызов. Как и в случае корпоративной многозадачности, планировщик автоматически вызывается при завершении системного вызова. Мы уже рассмотрели, что это необходимо для повышения эффективности использования процессора.  
Вытесняющая многозадачность является основным методом организации многозадачности в современных операционных системах. Наличие вытесняющей многозадачности позволяет избежать зависания системы при зависании любого из пользовательских процессов и гарантирует приемлемое время реакции в интерактивной системе.