Некоммерческое акционерное общество

АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ

Кафедра систем информационной безопасности

ОПЕРАЦИОННАЯ СИСТЕМА LINUX

Конспект лекций
для студентов специальности
5В100200 – Системы информационной безопасности

Алматы 2017

СОСТАВИТЕЛЬ: Е.А. Зуева. Операционная система Linux. Конспект лекций для студентов специальности 5В100200 – Системы информационной безопасности. - Алматы: АУЭС, 2017. – 47 с.

Данная разработка предназначена для студентов всех форм обучения специальности 5В100200 – «Системы информационной безопасности».

В конспекте лекций рассматриваются основные определения операционных систем и их механизмы действия, организация безопасности, криптографии и шифрования, в полной мере охватываются все вопросы, которые должен знать студент после изучения дисциплины «Операционная система Linux».

Предметом изучения дисциплины является освоение основных принципов работы с ОС Linux различной степени сложности и назначения. Студенту важно понять, как платформы отличаются друг от друга разработанными (и запускаемыми) программными компонентами, как правильно администрировать ресурсы, проводить грамотную политику работы в операционных системах, а также знать принципы построения современных ОС и системного программного обеспечения. При этом студент начинает разбираться в основных принципах организации безопасного интерфейса пользователя с программной системой; в методах анализа, исследования и моделирования вычислительных и информационных процессов, связанных с функционированием объектов профессиональной деятельности и их компонентов; в принципах, методах и способах комплексирования аппаратных и программных средств при создании защищенных вычислительных систем и сетей.

Профессиональная подготовка бакалавра требует умения грамотно работать с различными операционными системами и средами, оболочками. Предметом изучения в рассматриваемой дисциплине являются ОС Linux и средства, расширяющие ее возможности.

Илл. 2, библиогр. - 7 назв.

Рецензент: к.т.н., проф. АУЭС Ибраева Л. К.

Печатается по дополнительному плану издания некоммерческого акционерного общества «Алматинский университет энергетики и связи» на 2017 г.

Содержание

Лекция №1. Операционные системы. Особенности Linux

Цель лекции: ознакомить студентов с понятием операционной системы, с вопросами особенностей *Linux*.

Содержание лекции: основные понятия предмета; основные семейства операционных систем; графический интерфейс пользователя.

Операционная система (ОС) – это программа, которая обеспечивает возможность рационального использования оборудования компьютера удобным для пользователя образом. ОС является фундаментальным компонентом системного программного обеспечения. ОС представляет собой интерфейс между пользователем и компьютером [1]**.**

ОС - это некоторый набор программных средств, реализующих обработку и обработку информации пользователя для передачи ее в машинную логику для выполнения поставленных задач. Как правило, пользователь представляет себе операционную систему лишь как визуальную оболочку. ОС предварительно устанавливаются на любой компьютер, который вы покупаете. Большинство людей используют ту операционную систему, которая уже была предустановлена при покупке компьютера, но при желании они могут обновить или установить другую.

Существуют три наиболее популярных операционных систем для компьютеров: *Microsoft Windows***,** *Apple Mac Os* и *Linux*.

Характерные особенности *Linux* как ОС:

– многозадачность: много программ выполняются одновременно;

– многопользовательский режим: много пользователей одновременно работают на одной и той же машине;

– защита памяти процесса - сбой программы не вызывает зависания ОС;

– экономная загрузка: *Linux* считывает с диска только те части программы, которые действительно используются для выполнения;

– процессы-экземпляры программы могут использовать при выполнении одну и ту же память, что увеличивает быстродействие и экономит память;

– изменение размера виртуальной памяти во время выполнения программ;

– общая память программ и дискового кэша: вся свободная память используется для буферизации обмена с диском;

– динамические загружаемые разделяемые библиотеки;

– анализирование отладчиком не только выполняющуюся, но и завершившуюся аварийно программу;

– сертификация по стандарту *POSIX*.1, совместимость со стандартами *System V* и *BSD* на уровне исходных текстов;

– наличие исходного текста всех программ, включая тексты ядра, драйверов, средств разработки и приложений, они свободно распространяются;

– управление заданиями в стандарте *POSIX*;

– поддержка национальных алфавитов и соглашений, в т.ч. для русского и казахского языков;

– множественные виртуальные консоли: на одном дисплее несколько одновременных независимых сеансов работы, переключаемых с клавиатуры;

– поддержка ряда распространенных файловых систем (*MINIX, Xenix,* файловые системы *System V*);

– наличие собственной передовой файловой системы объемом до 4 терабайт и с именами файлов до 255 знаков;

– специальная файловая система *UMSDOS*, которая позволяет устанавливать *Linux* в файловую систему *DOS;*

– прозрачный доступ к разделам *DOS* (или *OS/2 FAT*): раздел *DOS* выглядит как часть файловой системы *Linux*;

– поддержка *VFAT (WNT, Windows*);

– поддержка всех стандартных форматов *CD ROM*;

– поддержка сети *TCP/IP*, включая *ftp, telnet, NFS* и т.д.

Современные операционные системы используют Графический Интерфейс Пользователя *(англ. GUI - Grafical user interface)*. *GUI* позволяет использовать мышь, клавиатуру и джойстик для управления экранными объектами (иконки, кнопки, значки, меню), представленные пользователю на дисплее, в виде сочетании графики и текста, т.е. все четко и ясно показано на экране компьютера в виде графических изображений, что позволяет легко работать на компьютере с помощью мыши, клавиатуры и т.п. *GUI* каждой ОС имеет свой внешний вид, поэтому если вы переключитесь на другую ОС, то, на первый взгляд и ощупь, незнакомая операционная система может показаться вам непривычным и неудобным. Тем не менее все они разработаны так, чтобы быть максимально простым в использовании [2].

Когда еще не было графического интерфейса *GUI*, компьютеры имели интерфейс командной строки. Это означает, что пользователь должен был ввести каждый раз команду в компьютер, чтобы отобразить в экране только один текст.

Контрольные вопросы

1 Дайте определение операционной системе.
2 Перечислите основные семейства ОС.
3 Какую роль играет командная строка?
4 Что такое *GUI*?

Лекция №2. Краткая эволюция операционных систем

Цель лекции: ознакомить студентов с архитектурой операционных систем.

Содержание лекции: история развития и поколения компьютерной техники; этапы эволюционного развития операционных систем.

Так как операционные системы появились и развивались в процессе конструирования компьютеров, то эти события исторически тесно связаны. Программное и аппаратное обеспечение эволюционировали совместно, оказывая взаимное влияние друг на друга. Появление новых технических возможностей приводило к прорыву в области создания удобных, эффективных и безопасных программ, которые, в свою очередь, были направлены на поиск новых технических решений. Ниже приведена краткая историческая эволюция вычислительных систем:

а) 1 поколение (1945–1955): электронные лампы и коммутационные панели. Операционных систем нет. После неудачных попыток Бэббиджа в конструировании цифровых компьютеров вплоть до Второй мировой войны не было практически никакого прогресса. Примерно в середине 1940-х гг. Говард Айкен (Howard Aiken) в Гарварде, Джон фон Нейман (*John von Neumann*) в Институте углубленного изучения в Принстоне, Дж. Преспер Эккерт (J. *Presper Eckert*), Вильям Мочли (*William Maucl*) в Пенсильванском университете, Конрад Цузе (*Konrad Zuse*) в Германии и многие другие продолжили работу в направлении создания вычислительных машин.

На первых вычислительных машинах использовались механические реле, которые были очень медлительны. Позже реле заменили электронными лампами. Машины получались громоздкими, заполняющими целые комнаты, с десятками тысяч электронных ламп. Каждую машину разрабатывала, строила, программировала, эксплуатировала и поддерживала в рабочем состоянии одна команда. Все программирование выполнялось на абсолютном машинном языке. Управление основными функциями машины осуществлялось при помощи соединения коммутационных панелей проводами. Фактически на компьютерах занимались только прямыми числовыми вычислениями, например, расчетами таблиц синусов, косинусов и логарифмов. К началу 50-х гг. XX в. появилась возможность записывать и считывать программы с перфокарт, но процедура вычислений оставалась прежней;

б) 2 поколение (1955 – начало 1960-х): транзисторы и системы пакетной обработки. Пакетные операционные системы. Впервые сложилось четкое разделение между проектировщиками, сборщиками, операторами, программистами и обслуживающим персоналом. Машины, называемые мэйнфреймами, располагались в специальных комнатах с кондиционированным воздухом, где ими управлял целый штат профессиональных операторов. Чтобы выполнить задание, программист

сначала должен был записать его на бумаге (на языке Фортран или Ассемблер), а затем перенести на перфокарты и передать оператору для последующей обработки. Для повышения эффективности использования машинного времени общепринятым решением стала система пакетной обработки. Суть системы заключалась в том, чтобы собрать полный комплект заданий (перфокарт) в комнате входных данных и затем переписать их на магнитную ленту при использовании небольшого недорогого компьютера. Например, компьютер класса *IBM* 1401 использовался для считывания карт, копирования лент и печати выходных данных, но не подходил для числовых вычислений; для вычислений использовались более дорогостоящие машины такие, как *IBM* 7094.

Структура типичного входного задания начиналась с перфокарты, на которой указывалось максимальное время выполнения задания в минутах, загружаемый учетный номер и имя программиста. Затем поступала карта *&SFORTRAN*, дающая операционной системе указание загрузить компилятор языка Фортран с системной магнитной ленты. Эта карта следовала за программой, которую нужно было компилировать, а после нее шла карта *SLOAD*, указывающая операционной системе загрузить скомпилированную объектную программу. (Скомпилированные программы часто записывались на временных лентах, данные с которых могли стираться сразу после использования). Следом шла карта *SRUN* со сведениями, дающими операционной системе команду выполнять программу. Последняя карта завершения *SEND* отмечала конец задания. Эти примитивные управляющие перфокарты были предшественниками современных языков управления и интерпретаторов команд.

Большие компьютеры второго поколения использовались, главным образом, для научных и технических вычислений таких, как решение дифференциальных уравнений в частных производных, часто встречающихся в физике и инженерных задачах. В основном, на них программировали на языке Фортран и Ассемблер. Типичными операционными системами были: *FMS (Fortran Monitor System)* и *IBSYS* (операционная система, которая была создана корпорацией *IBM* для компьютера *IBM* 7094);

в) 3 поколение (1960-1980): интегральные схемы и многозадачность. Первые многозадачные операционные системы. Переход от отдельных полупроводниковых элементов типа транзисторов к интегральным микросхемам. Вычислительная техника стала более надежной и дешевой. К началу 60-х гг. XX в. большинство изготовителей компьютеров имели две отдельные, полностью несовместимые производственные линии:

– научных крупномасштабных компьютеров с пословной обработкой текста (например, *IBM* 7094, использовавшиеся для числовых вычислений в науке и технике);

– коммерческих компьютеров с посимвольной обработкой (например, *IBM* 1401, используемые банками и страховыми компаниями для сортировки и печатания данных).

Развитие и поддержка двух разных производственных линий для больших и малых ЭВМ для изготовителей были дороги и неудобны. Многим покупателям изначально требовалась небольшая машина, но через некоторое время возникала необходимость в более мощном компьютере. Корпорация *IBM* попыталась решить эти проблемы, выпустив серию программно-совместимых машин IBM/360, варьирующихся от компьютеров размером с *IBM* 1401 до машин, значительно более мощных, чем *IBM* 7094 для поддержки научных и коммерческих вычислений

Так как все машины имели одинаковую структуру и набор команд, то программы, написанные для одного компьютера, могли работать на всех других. Таким образом, одно семейство машин могло удовлетворить нужды всех покупателей. В последующие годы корпорация *IBM* выпустила компьютеры, совместимые с 360-ми, серии: 370, 4300, 3080 и 3090.

Серия 360-х стала первой основной линией компьютеров, на которой использовались мелкомасштабные интегральные схемы, дававшие преимущество в цене и качестве по сравнению с машинами второго поколения, созданными из отдельных транзисторов. В компьютерных центрах до сих пор можно встретить потомков этих машин. В настоящее время они используются для управления огромными базами данных (например, для систем бронирования и продажи билетов на авиалиниях) или как серверы узлов Интернета, которые должны обрабатывать тысячи запросов в секунду.

Для решения поставленной задачи была разработана операционная система *OS*/360, которая состояла из миллионов строк, написанных на Ассемблере разными программистами, и содержала много ошибок. Часть ошибок в последующих версиях исправлялась, но вместо них появлялись новые. Несмотря на свои огромные размеры и недостатки, операционная система *OS*/360 и подобные ей операционные системы третьего поколения, созданные другими производителями компьютеров, удовлетворяли требованиям большинства клиентов.

С повышением производительности процессоров увеличилась сложность задач, решаемых компьютерами. Повышению эффективности использования процессорного времени мешала низкая скорость работы механических устройств «ввода-вывода» (быстрый считыватель перфокарт мог обработать 1200 перфокарт в минуту, принтеры печатали до 600 строк в минуту). Вместо непосредственного чтения пакета заданий с перфокарт в память стали использовать его предварительную запись – сначала на магнитную ленту, а затем и на диск. Когда в процессе выполнения задания требовался ввод данных, они стали читаться с диска. Точно так же выходная информация сначала копировалась в системный буфер и записывалась на ленту или диск, а печаталась только после завершения задания. Вначале действительные операции «ввода-вывода» осуществлялись в режиме *offline* – при использовании других, более простых, отдельно стоящих компьютеров. В дальнейшем они стали выполняться на том же компьютере, который производил вычисления, т. е. в режиме *online*. Такой прием получил название *spooling (Simultaneous Peripheral Operation On Line* – совместная

периферийная операция в интерактивном режиме) или подкачка-откачка данных. Введение техники спулинга в пакетные системы позволило совместить реальные операции «ввода-вывода» одного задания с выполнением другого задания, но потребовало разработки аппарата прерываний для извещения процессора об окончании этих операций.

Магнитные ленты были устройствами последовательного доступа, т. е. информация считывалась с них в том порядке, в каком была записана. Появление магнитного диска, для которого не важен порядок чтения информации – устройства прямого доступа, привело к дальнейшему развитию вычислительных систем. При обработке пакета заданий на магнитной ленте очередность запуска заданий определялась порядком их ввода. При обработке пакета заданий на магнитном диске появилась возможность выбора очередного выполняемого задания. Пакетные системы начали заниматься планированием заданий – задание стало выбираться в зависимости от наличия запрошенных ресурсов, срочности вычислений и др.

Дальнейшее повышение эффективности использования процессора было достигнуто с помощью мультипрограммирования – пока одна программа выполняла операцию «ввода-вывода», процессор не простаивал (как при однопрограммном режиме), а выполнял другую программу; когда операция «ввода-вывода» заканчивалась, процессор возвращался к выполнению первой программы.

Самым важным достижением этого периода явилась многозадачность – управление разделением совместно используемых ресурсов таких, как процессор, оперативная память, файлы и внешние устройства. Другим достижением операционных систем третьего поколения стала способность считывать задание с перфокарт на диск – спулинг. Когда текущее задание заканчивалось, операционная система загружала новое задание с диска в освободившийся раздел памяти и запускала его. Спулинг стали использовать так же для выдачи полученных данных.

Еще одним важным моментом данного времени был большой рост количества мини-компьютеров, начиная с выпуска машин класса *PDP*-1 корпорацией *DEC* в 1961 г., которые обладали очень маленькой оперативной памятью, но стоили дешево и поэтому пользовались большим спросом. Некоторые виды работ они выполняли с такой же скоростью, что и *IBM* 7094. Это дало толчок к появлению новой индустрии мини-компьютеров. Кен Томпсон (*Ken Thompson*), один из специалистов по компьютерам в *Bell Labs*, работавший над проектом *MULTICS*, решил для мини-компьютера *PDP*-7 написать усеченную однопользовательскую версию операционной системы *MULTICS*, которая позже развилась в операционную систему *UNIX*.

Впоследствии появилось очень много версий операционной системы *UNIX*. Чтобы стало возможным писать программы, работающие в любой *UNIX*-системе, Институт инженеров по электротехнике и электронике *IEEE* разработал стандарт системы *UNIX*, называемый *POSIX*, который теперь поддерживает большинство версий *UNIX*. Стандарт *POSIX* определяет минимальный интерфейс системного вызова, который должны поддерживать

совместимые системы *UNIX*. Некоторые другие операционные системы теперь тоже поддерживают интерфейс *POSIX*. В 1987 г. Кен Томпсон создал маленький клон системы *UNIX* для образовательных целей – *MINIX*. Функционально система *MINIX* очень похожа на *UNIX*, включая поддержку стандарта *POSIX*.

Желание иметь свободно распространяемую рабочую версию *MINIX* (в противоположность образовательной) привело финского студента Линуса Торвальда (*Linus Torvalds*) к написанию системы *Linux*. Эта система была разработана на основе операционной сстемы *MINIX* и первоначально обладала ее характерными особенностями, например, поддерживала ту же файловую систему. Позже система *Linux* была значительно расширена, но сохранила большую часть структуры, общей как для системы *MINIX*, так и для системы *UNIX*, на которой и была основана;

г) 4 поколение (1980 г. - настоящее время): персональные компьютеры, классически сетевые и распределенные системы. Зафиксировано появление больших интегральных схем (*LSI, Large Scale Integration*) – кремниевых микросхем, содержащих тысячи транзисторов на одном квадратном сантиметре. В 1974 г., когда компания *Intel* выпустила первый универсальный 8-разрядный центральный процессор *Intel* 8080, для него потребовалась ОС, с помощью которой можно было бы его протестировать - *CP/M (Control Program for Microcomputers* – программа управления для микрокомпьютеров), положившую начало созданию операционных систем для микроЭВМ.

В начале 80-х гг. XX в. корпорация IBM разработала IB*M PC (Personal Computer* - персональный компьютер) и начала искать для него программное обеспечение. Сотрудники *IBM* обратились к Биллу Гейтсу, чтобы получить лицензию на право использования его интерпретатора языка Бейсик (*BASIC*). Далее корпорация *IBM* обратилась к Гейтсу с просьбой обеспечить ее ОС.

После запроса Гейтс выяснил, что в компании-изготовителе компьютеров *Seattle Computer Products* есть подходящая ОС *DOS (Disk Operating System* – дисковая ОС). Он выкупил *DOS* и создал пакет программ *DOS*/*BASIC*, который был куплен компанией *IBM*. Впоследствии для усовершенствования программы корпорация *IBM* пригласила Билла Гейтса и Тима Патерсона, автора *DOS*, ставшего первым служащим компании Гейтса «*Microsoft*». Видоизмененная система была переименована в *MS-DOS* (*MicroSoft Disk Operating System*) и быстро заняла доминирующее положение на рынке *IBM PC*. Самым важным оказалось решение Гейтса продать *MS-DOS* компьютерным компаниям для установки вместе с их оборудованием.

Позже был написан стандарт *MSX*, который определял ОС и характеристики аппаратных средств для школьных ПЭВМ. Согласно стандарту *MSX* машина должна была иметь: оперативную память объемом не менее 16 Кб; постоянную память объемом 32 Кб с встроенным интерпретатором языка Бейсик; цветной графический дисплей с разрешающей способностью 256Í192 точки и 16 цветами; трехканальный звуковой генератор на 8 октав; параллельный порт для подключения принтера; контроллер для управления внешним накопителем, подключаемым снаружи.

ОС *CP/M, MS DOS* и другие для первых микрокомпьютеров полностью основывались на вводе команд с клавиатуры. Энгельбарт изобрел графический интерфейс пользователя (*GUI, Graphical User Interface*), состоящий из окон, значков, различных меню и мыши. Эту идею переняли разработчики из *Xerox PARC* и встроили в сконструированные ими машины.

Однажды Стив Джобс, который изобрел компьютер Apple, посетил *PARC* и увидел *GUI*. Он осознал его потенциальную ценность и приступил к созданию *Apple* с графическим интерфейсом. Это привело к проекту *Lisa*, который был слишком дорог и потерпел коммерческую неудачу. Вторая попытка Джобса, *Apple Macintosh*, имела огромный успех не только из-за невысокой цены, но и потому, что на нем работал дружественный интерфейс, т. е. предназначенный для пользователей, ничего не знающих о компьютерах и не желающих чему-либо обучаться.

В первом проекте дисплей был монохромным, во втором – цветным, но оба имели высокую разрешающую способность и скорость вывода графической информации. Операционные системы для этих машин были спроектированы так, чтобы максимально использовать возможности работы с графикой. В них применяется многооконный интерфейс и манипулятор мышь. Для выбора той или иной операции или рабочего объекта на экран выводится несколько условных графических символов (пиктограмм), среди которых пользователь делает выбор с помощью мыши.

Когда корпорация *Microsoft* решила создать преемника *MS DOS*, она находилась полностью под влиянием успехов компании *Macintosh*. Была разработана система, получившая название *Windows*, которая на протяжении 10 лет, с 1985 по 1995 гг., исполняла роль графической среды поверх операционной системы *MS DOS*. В 1995 гг. вышла в свет автономная версия *Windows* 95, включающая в себя множество особенностей операционной системы *MS DOS*, но только для загрузки и выполнения старых программ. В 1998 гг. была выпущена измененная версия этой системы – *Windows* 98, которая все еще содержала большое количество программ 16-разрядного Ассемблера *Intel*.

Другой ОС *Microsoft* стала *Windows NT* (*New Technology* – новая технология), которая на определенном уровне совместима с *Windows* 95, но ее ядро полностью переписано. Это 32-разрядная система. Дэвид Катлер, главный разработчик *Windows NT*, был также одним из создателей операционной системы *VMS* для компьютеров *VAX*, поэтому некоторые идеи системы *VMS* присутствуют и в *Windows NT*. Корпорация *Microsoft* ожидала, что первая же версия *Windows NT* вытеснит *MS DOS* и все другие версии *Windows*, так как это была система, намного превосходящая предыдущие, но надежда не оправдалась. И только системе *Windows NT* 4.0 удалось получить относительно широкое распространение, особенно в корпоративных сетях. Версия *Windows NT* 5.0 была переименована в *Windows* 2000 в начале 1999 г. Она должна была стать преемником *Windows* 98 и *Windows NT* 4.0. Но этому также не было суждено случиться, поэтому корпорация *Microsoft* выпустила

еще одну версию *Windows* 98, названную *Windows Me* (*Me, Millennium edition* – выпуск тысячелетия).

Главным соперником *Windows* в мире персональных компьютеров становится система *UNIX*, которая является самой сильной ОС для рабочих станций и сетевых серверов. Она стала особенно популярна на машинах с высокопроизводительными *RISC*-процессорами (*RISC, reduced instruction set computer* – компьютер с сокращенным набором команд). На компьютерах с процессорами *Pentium* популярной альтернативой *Windows* для студентов и других разнообразных пользователей становится *Linux*.

С середины 80-х гг. *XX* в. начали развиваться сети персональных компьютеров, управляемые сетевыми и распределенными ОС. В сетевой операционной системе пользователи могут регистрироваться на удаленных машинах и копировать файлы с одной машины на другую. Каждый компьютер работает под управлением локальной операционной системы и имеет своего собственного локального пользователя. Сетевые операционные системы несущественно отличаются от однопроцессорных операционных систем. Дополнительно им требуются: сетевой интерфейсный контроллер; специальное низкоуровневое программное обеспечение, поддерживающее работу контроллера; программы, разрешающие пользователям удаленную регистрацию в системе и доступ к удаленным файлам.

Контрольные вопросы

1 Опишите этапы эволюционного развития ОС.
2 Расскажите об архитектуре построения ОС.
3 Приведите примеры ОС на разные группы.

Лекция №3. Концепции разработки систем

Цель лекции: ознакомление студентов с концепциями, составом и компонентами операционных систем.

Содержание лекции: состав и компоненты, принципы разработки операционных систем; типы ядер систем.

Компоненты ОС:

– управление файловой системой: процесс работы компьютера в определенном смысле сводится к обмену файлами между устройствами, в ОС имеются программные модули, управляющие файловой системой;

– командный процессор запрашивает у пользователя команды и выполняет их;

– драйверы устройств: к магистрали компьютера подключаются различные устройства (дисководы, монитор, клавиатура, мышь, принтер и др.); и каждое устройство выполняет определенную функцию (ввод

информации, хранение информации, вывод информации); при этом техническая реализация устройств существенно различается - при этом драйверы обеспечивают управление работой устройств и согласование информационного обмена с другими устройствами, а также позволяют производить настройку некоторых параметров устройств. Каждому устройству соответствует свой драйвер;

– графический интерфейс: для упрощения работы пользователя в состав современных ОС входят программные модули, создающие графический пользовательский интерфейс, благодаря которым пользователь может вводить команды с помощью мыши, тогда как в режиме командной строки необходимо вводить команды с помощью клавиатуры;

– сервисные программы или утилиты - позволяют обслуживать диски (проверять, сжимать, дефрагментировать и так далее), выполнять операции с файлами (архивировать и так далее), работать в компьютерных сетях и т.д.;

– справочная система позволяет оперативно получить необходимую информацию как о функционировании операционной системы в целом, так и о работе ее отдельных модулей.

Принципы разработки архитектуры ОС:

– принцип модульности отражает технологические и эксплуатационные свойства системы. Наибольший эффект от его использования достижим в случае, когда принцип распространен одновременно на операционную систему, прикладные программы и аппаратуру;

– принцип функциональной избирательности: в ОС выделяется некоторая часть важных модулей, которые должны постоянно находиться в оперативной памяти для более эффективной организации вычислительного процесса, эту часть в ОС называют ядром. Помимо программных модулей, входящих в состав ядра и постоянно располагающихся в оперативной памяти, может быть много других системных программных модулей – транзитных, которые загружаются в оперативную память только при необходимости и в случае отсутствия свободного пространства могут быть замещены другими транзитными модулями;

– принцип генерируемости ОС - такой способ исходного представления центральной системной управляющей программы ОС (ее ядра и основных компонентов, которые должны постоянно находится в оперативной памяти), который позволял бы настраивать эту системную супервизорную часть, исходя из конкретной конфигурации конкретного вычислительного комплекса и круга решаемых задач и процесс генерации осуществляется с помощью специальной программы-генератора и соответствующего входного языка для этой программы, позволяющего описывать программные возможности системы и конфигурацию машины, в результате генерации получается полная версия ОС и сгенерированная версия ОС представляет собой совокупность системных наборов модулей и данных. Принцип модульности положительно

проявляется при генерации ОС - он существенно упрощает настройку ОС на требуемую конфигурацию вычислительной системы;

– принцип функциональной избыточности учитывает возможность проведения одной и той же работы различными средствами. В состав ОС может входить несколько типов мониторов (модулей супервизора, управляющих тем или другим видом ресурса), различные средства организации коммуникаций между вычислительными процессами. Наличие нескольких типов мониторов, нескольких систем управления файлами позволяет пользователям быстро и наиболее адекватно адаптировать ОС к определенной конфигурации вычислительной системы, обеспечивать максимально эффективную загрузку технических средств при решении конкретного класса задач, получать максимальную производительность при решении заданного класса задач;

– принцип виртуализации позволяет представить структуру системы в виде определенного набора планировщиков процессов и распределителей ресурсов (мониторов) и использовать единую централизованную схему распределения ресурсов;

– принцип совместимости - способность ОС выполнять программы, написанные для других ОС или для более ранних версий данной ОС, а также для другой аппаратной платформы. Необходимо разделять вопросы двоичной совместимости и совместимости на уровне исходных текстов приложений: двоичная совместимость достигается в том случае, когда можно взять исполняемую программу и запустить ее на выполнение на другой ОС, т.е. необходимы совместимость на уровне команд процессора, и совместимость на уровне системных вызовов, и даже на уровне библиотечных вызовов, если они являются динамически связываемыми. Совместимость на уровне исходных текстов требует наличия соответствующего транслятора в составе системного программного обеспечения, а также совместимости на уровне библиотек и системных вызовов, т.е. необходима перекомпиляция имеющихся исходных текстов в новый выполняемый модуль;

– принцип мобильности (переносимости) - ОС относительно легко должна переноситься с процессора одного типа на процессор другого типа и с аппаратной платформы (которая включает наряду с типом процессора и способ организации всей аппаратуры компьютера, иначе говоря, архитектуру вычислительной системы) одного типа на аппаратную платформу другого типа. Заметим, что принцип переносимости очень близок принципу совместимости, хотя это и не одно и то же;

– принцип обеспечения безопасности вычислений - защита ресурсов одного пользователя от других и установление квот по ресурсам для предотвращения захвата одним пользователем всех системных ресурсов (таких, как память).

Классификация ядер ОС:

– монолитное ядро предоставляет богатый набор абстракций оборудования. Все части монолитного ядра работают в одном адресном

пространстве. Это такая схема операционной системы, при которой все компоненты её ядра являются составными частями одной программы, используют общие структуры данных и взаимодействуют друг с другом путём непосредственного вызова процедур. Монолитное ядро - старейший способ организации операционных систем. Примером систем с монолитным ядром является большинство *UNIX*-систем. Достоинства: скорость работы, упрощённая разработка модулей. Недостатки: поскольку всё ядро работает в одном адресном пространстве, сбой в одном из компонентов может нарушить работоспособность всей системы. Примеры: Традиционные ядра *UNIX* (такие как *BSD*), ядро *MS-DOS*, ядро *KolibriOS*;

– модульное ядро - современная, усовершенствованная модификация архитектуры монолитных ядер ОС. В отличие от «классических» монолитных ядер, модульные ядра, как правило, не требуют полной перекомпиляции ядра при изменении состава аппаратного обеспечения компьютера. Вместо этого модульные ядра предоставляют тот или иной механизм подгрузки модулей ядра, поддерживающих то или иное аппаратное обеспечение (например, драйверов). При этом подгрузка модулей может быть как динамической (выполняемой «на лету», без перезагрузки ОС, в работающей системе), так и статической (выполняемой при перезагрузке ОС после переконфигурирования системы на загрузку тех или иных модулей). Пример - современный *Linux*;

– микроядро предоставляет только элементарные функции управления процессами и минимальный набор абстракций для работы с оборудованием. Бóльшая часть работы осуществляется с помощью специальных пользовательских процессов, называемых сервисами. Решающим критерием «микроядерности» является размещение всех или почти всех драйверов и модулей в сервисных процессах, иногда с явной невозможностью загрузки любых модулей расширения в собственно микроядро, а также разработки таких расширений. Достоинства: устойчивость к сбоям оборудования, ошибкам в компонентах системы. Существенно упрощается процесс отладки компонентов ядра, так как новая версия драйвера может загружаться без перезапуска всей операционной системы. Микроядерная архитектура повышает надежность системы, поскольку ошибка на уровне непривилегированной программы менее опасна, чем отказ на уровне режима ядра. Недостатки: передача данных между процессами требует накладных расходов. Примеры: *Symbian OS; Windows CE; OpenVMS; Mach*, используемый в *GNU/Hurd* и *Mac OS X*; *QNX; AIX; Minix; ChorusOS; AmigaOS; MorphOS*;

– экзоядро - ядро операционной системы, предоставляющее лишь функции для взаимодействия между процессами, безопасного выделения и освобождения ресурсов. Предполагается, что *API* для прикладных программ будут предоставляться внешними по отношению к ядру библиотеками (откуда и название архитектуры). Возможность доступа к устройствам на уровне контроллеров позволит эффективней решать некоторые задачи, которые плохо вписываются в рамки универсальной ОС, например, реализация СУБД будет иметь доступ к диску на уровне секторов диска, а не файлов и кластеров, что положительно скажется на быстродействии;

– наноядро - архитектура ядра ОС, в рамках которой крайне упрощённое и минималистичное ядро выполняет лишь одну задачу - обработку аппаратных прерываний, генерируемых устройствами компьютера. После обработки прерываний от аппаратуры наноядро, в свою очередь, посылает информацию о результатах обработки (например, полученные с клавиатуры символы) вышележащему программному обеспечению при помощи того же механизма прерываний. Примером является KeyKOS - самая первая ОС на наноядре. Первая версия вышла ещё в 1983-м году;

– гибридное ядро - это модифицированные микроядра, позволяющие для ускорения работы запускать «несущественные» части в пространстве ядра. Элементы микроядерной архитектуры и элементы монолитного ядра переплетены в ядре *Windows NT*. Хотя *Windows NT* часто называют микроядерной ОС, это не совсем так. Микроядро *NT* слишком велико (более 1 Мбайт, кроме того, в ядре системы находится, например, ещё и модуль графического интерфейса), чтобы носить приставку «микро». Компоненты ядра *Windows NT* располагаются в вытесняемой памяти и взаимодействуют друг с другом путём передачи сообщений, как и положено в микроядерных операционных системах. В то же время все компоненты ядра работают в одном адресном пространстве и активно используют общие структуры данных, что свойственно операционным системам с монолитным ядром. Причина проста: чисто микроядерный дизайн коммерчески менее выгоден, поскольку менее эффективен (за счет накладных расходов на передачу сообщений там, где можно было обойтись вызовами функций). Таким образом, *Windows NT* можно с полным правом назвать гибридной операционной системой.

Контрольные вопросы

1 Дайте классификацию ОС по типу ядер.
2 На каком типе ядре существует современный *Linux*?
3 Приведите примеры на каждый тип ядра.

Лекция №4. Архитектура и классификация операционных систем

Цель лекции: ознакомить студентов с архитектурой операционных систем, провести классификацию ОС по определенным признакам.

Содержание лекции: архитектура операционных систем; классификация систем по признакам.

Под архитектурой ОС понимают структурную и функциональную организацию ОС на основе некоторой совокупности программных модулей. В состав ОС входят исполняемые и объектные модули стандартных для данной ОС форматов, программные модули специального формата

(например, загрузчик ОС, драйверы «ввода-вывода»), конфигурационные файлы, файлы документации, модули справочной системы и т.д.

На архитектуру ранних операционных систем обращалось мало внимания: во-первых, ни у кого не было опыта в разработке больших программных систем, а во-вторых, проблема взаимозависимости и взаимодействия модулей недооценивалась. В подобных монолитных ОС почти все процедуры могли вызывать одна другую. Такое отсутствие структуры было несовместимо с расширением операционных систем. Первая версия ОС *OS*/360 была создана коллективом из 5000 человек за 5 лет и содержала более 1 млн строк кода. Разработанная несколько позже операционная система *Mastics* содержала к 1975 году уже 20 млн строк [1]. Стало ясно, что разработка таких систем должна вестись на основе модульного программирования.

Большинство современных ОС представляют собой хорошо структурированные модульные системы, способные к развитию, расширению и переносу на новые платформы. Какой-либо единой унифицированной архитектуры ОС не существует, но известны универсальные подходы к структурированию ОС. Принципиально важными универсальными подходами к разработке архитектуры ОС являются:

– модульная организация;

– функциональная избыточность;

– функциональная избирательность;

– параметрическая универсальность;

– концепция многоуровневой иерархической вычислительной системы, по которой ОС представляется многослойной структурой;

– разделение модулей на две группы по функциям: ядро – модули, выполняющие основные функции ОС, и модули, выполняющие вспомогательные функции ОС;

– разделение модулей ОС на две группы по размещению в памяти вычислительной системы: резидентные, постоянно находящиеся в оперативной памяти, и транзитные, загружаемые в оперативную память только на время выполнения своих функций;

– реализация двух режимов работы вычислительной системы: привилегированного режима (режима ядра – *Kernel mode*), или режима супервизора (*supervisor mode*), и пользовательского режима (*user mode*) или режима задачи (*task mode*);

– ограничение функций ядра (и количества модулей ядра) до минимального количества необходимых самых важных функций.

Первые ОС разрабатывались как монолитные системы без четко выраженной структуры. Она предполагала следующую структуру [3]:

– главная программа, которая вызывает требуемые сервисные процедуры;

– набор сервисных процедур, реализующих системные вызовы;

– набор утилит, обслуживающих сервисные процедуры.

В этой модели для каждого системного вызова имеется одна сервисная процедура. Утилиты выполняют функции, которые нужны нескольким сервисным процедурам.

Классической считается архитектура ОС, основанная на концепции иерархической многоуровневой машины, привилегированном ядре и пользовательском режиме работы транзитных модулей. Модули ядра выполняют базовые функции ОС: управление процессами, памятью, устройствами ввода-вывода и т.п. Ядро составляет сердцевину ОС, без которой она является полностью неработоспособной и не может выполнить ни одну из своих функций. В ядре решаются внутрисистемные задачи организации вычислительного процесса, недоступные для приложения.

Особый класс функций ядра служит для поддержки приложений, создавая для них так называемую прикладную программную среду. Приложения могут обращаться к ядру с запросами – системными вызовами – для выполнения тех или иных действий, например, открытие и чтение файла, получение системного времени, вывода информации на дисплей и т.д. Функции ядра, которые могут вызываться приложениями, образуют интерфейс прикладного программирования - *API (Application Programming Interface*).

Для обеспечения высокой скорости работы ОС модули ядра (по крайней мере, большая их часть) являются резидентными и работают в привилегированном режиме (*Kernel mode*). Этот режим, во-первых, должен обезопасить работу самой ОС от вмешательства приложений, и, во-вторых, должен обеспечить возможность работы модулей ядра с полным набором машинных инструкций, позволяющих собственно ядру выполнять управление ресурсами компьютера, в частности, переключение процессора с задачи на задачу, управлением устройствами ввода-вывода, распределением и защитой памяти и др. Остальные модули ОС выполняют не столь важные функции, как ядро, и являются транзитными. Например, это могут быть программы архивирования данных, дефрагментации диска, сжатия дисков, очистки дисков и т.п.

В концепции многоуровневой (многослойной) иерархической машины структура ОС также представляется рядом слоев. При такой организации каждый слой обслуживает вышележащий слой, выполняя для него некоторый набор функций, которые образуют межслойный интерфейс. На основе этих функций следующий верхний по иерархии слой строит свои функции – более сложные и более мощные и т.д. Такая организация системы существенно упрощает ее разработку, т.к. позволяет сначала сверху вниз определить функции слоев и межслойные интерфейсы, а при детальной реализации – наращивать мощность функции слоев. Кроме того, модули каждого слоя можно изменять без необходимости изменений в других слоях (но не меняя межслойных интерфейсов) [4].

Суть этой архитектуры состоит в следующем. В привилегированном режиме остается работать только очень небольшая часть ОС, называемая микроядром. Микроядро защищено от остальных частей ОС и приложений. В

его состав входят машинно-зависимые модули, а также модули, выполняющие базовые механизмы обычного ядра. Все остальные более высокоуровневые функции ядра оформляются как модули, работающие в пользовательском режиме. Так, менеджеры ресурсов, являющиеся неотъемлемой частью обычного ядра, становятся периферийными модулями, работающими в пользовательском режиме. Схематично механизм обращений к функциям ОС, оформленным в виде серверов, выглядит, как показано на рисунке 1.

Рисунок 1 – Архитектура микроядра

Контрольные вопросы

1 Что такое архитектура ОС?
2 Дайте этапное развитие архитектуры ОС.
3 Какая структура у микроядерной архитектуры?

Лекция №5. Компоненты, назначение и состав, функции Linux

Цель лекции: ознакомить студентов со структурой операционной системы, разобрать основные системные понятия операционных систем.

Содержание лекции: структура операционной системы *Linux*; системные составляющие системы; классификация ОС и функции.

Структура операционной системы *Linux* являет собой многослойную систему, нижний слой которой составляет ядро. Это основание, обязующееся держать все под контролем, обрабатывать ошибки и команды. Если не вдаваться в подробности структуры аппаратной составляющей ПК, то лежит основание на машинной логике. Верхний слой ОС – это то, что видит пользователь: изображения, скринсейвер, курсор и т.д.

В состав этого программного сбора входит:

– само ядро, состоящее из вспомогательных модулей, которые вместе и выполняют все функции ядра);

– загрузчик;

– драйверы устройств;

– командный процессор, благодаря чему ОС быстрее и функциональнее;

– интерфейс.

В общей вычислительной системе компьютера расположена «ось» между встроенным программным обеспечением компьютера (*BIOS* и все то, что заставляет работать самые простые команды: сложение, вычитание и сдвиг регистра) и программными приложениями пользователя (тут уже область пользователя: приложения, файлы и др.).

ОС заведует вводом и выводом данных и раздает эти и задачи тем или иным устройствам, загружает программы в оперативную память и выполняет их, да и вообще управляет оперативной памятью, раздавая ее направо и налево разным запущенным вами приложениям в зависимости от того, как будет рациональнее использовать ее. Операционная система также управляет доступом ко всем источникам данных (съемным и оптическим дискам, флеш-носителям и т.д.), защищает данные и саму себя (от взломщиков, вредоносных программ и пользовательских ошибок). ОС отвечает и за многозадачность ПК, обеспечивает возможность работы на компьютере множества пользователей. Если вы думаете, что без операционной системы мы видели бы только черный экран и белые буквы с цифрами, ошибаетесь – мы не видели бы и этого, потому как для отображения букв уже нужна ОС.

Архитектура большинства компьютеров на уровне машинных команд очень неудобна для использования прикладными программами. Например, работа с диском предполагает знание внутреннего устройства его электронного компонента – контроллера для ввода команд вращения диска, поиска и форматирования дорожек, чтения и записи секторов и т. д. Ясно, что средний программист не в состоянии учитывать все особенности работы оборудования (то есть заниматься разработкой драйверов устройств), а должен иметь простую высокоуровневую абстракцию, скажем представляя информационное пространство диска как набор файлов. Файл можно открывать для чтения или записи, использовать для получения или сброса информации, а потом закрывать. Это концептуально проще, чем заботиться о деталях перемещения головок дисков или организации работы мотора. Аналогичным образом, с помощью простых и ясных абстракций, скрываются от программиста все ненужные подробности организации прерываний, работы таймера, управления памятью и т.д. Более того, на современных вычислительных комплексах можно создать иллюзию неограниченного размера операционной памяти и числа процессоров. Всем этим занимается операционная система. Таким образом, операционная система представляется пользователю виртуальной машиной, с которой проще иметь дело, чем непосредственно с оборудованием компьютера.

Операционная система предназначена для управления всеми частями весьма сложной архитектуры компьютера. Представим, к примеру, что

произойдет, если несколько программ, работающих на одном компьютере, будут пытаться одновременно осуществлять вывод на принтер. Мы получили бы мешанину строчек и страниц, выведенных различными программами. Операционная система предотвращает такого рода хаос за счет буферизации информации, предназначенной для печати, на диске и организации очереди на печать. Для многопользовательских компьютеров необходимость управления ресурсами и их защиты еще более очевидна. Следовательно, операционная система, как менеджер ресурсов, осуществляет упорядоченное и контролируемое распределение процессоров, памяти и других ресурсов между различными программами [2].

Если вычислительная система допускает совместную работу нескольких пользователей, то возникает проблема организации их безопасной деятельности. Необходимо обеспечить сохранность информации на диске, чтобы никто не мог удалить или повредить чужие файлы. Нельзя разрешить программам одних пользователей произвольно вмешиваться в работу программ других пользователей. Нужно пресекать попытки несанкционированного использования вычислительной системы. Всю эту деятельность осуществляет операционная система как организатор безопасной работы пользователей и их программ.

Если перейти от частного (*Linux*) к общему, то, взяв многообразие существующих ОС, можно классифицировать по следующим принципам:

1) По количеству одновременно существующих программных процессов ОС делятся на однопрограммные и мультипрограммные. В мультипрограммных Ос, в отличие от однопрограммных, вычислительный процесс организуется таким образом, что в памяти компьютера находятся одновременно несколько программ, попеременно выполняющихся на одном процессоре.

2) По числу пользователей, осуществляющих доступ к вычислительной системе, различают однопользовательские и многопользовательские ОС. Многопользовательские системы предоставляют возможность одновременного доступа к вычислительной системе нескольким пользователям. При этом каждый из них работает за своим терминалом, однако все вычисления производятся на одном компьютере.

3) По назначению ОС делятся на универсальные и специализированные. Специализированные ОС работают с фиксированным набором программ.

4) По способу загрузки можно выделить загружаемые ОС и системы, постоянно находящиеся в памяти вычислительной системы. Последние, как правило, используются для управления работой специализированных устройств.

5) По особенности области использования ОС подразделяются на системы пакетной обработки, системы разделения времени и системы реального времени.

Системы пакетной обработки предназначаются в основном для решения задач вычислительного характера, не требующих быстрого получения результата.

Системы разделения времени организуют вычислительный процесс таким образом, что каждой задаче выделяется квант процессорного времени, вследствие чего ни одна задача не занимает процессор надолго, и это дает возможность пользователю вести диалог со своей программой.

Системы реального времени используются для управления различными техническими объектами или технологическими процессами. Такие системы характеризуются предельно допустимым временем реакции на внешнее событие, в течение которого должна быть выполнена программа, управляющая объектом. Система должна обрабатывать поступающие данные быстрее, чем они могут поступать, причем от нескольких источников одновременно.

В любой операционной системе поддерживается механизм, который позволяет пользовательским программам обращаться к услугам ядра ОС. В операционных системах наиболее известной советской вычислительной машины БЭСМ-6 соответствующие средства «общения» с ядром назывались экстракодами, в операционных системах *IBM* назывались системными макрокомандами и т.д. В ОС *Linux* такие средства - системные вызовы.

Системные вызовы (*system calls)* – это интерфейс между операционной системой и пользовательской программой. Они создают, удаляют и используют различные объекты, главные из которых – процессы и файлы. Пользовательская программа запрашивает сервис у операционной системы, осуществляя системный вызов. Имеются библиотеки процедур, которые загружают машинные регистры определенными параметрами и осуществляют прерывание процессора, после чего управление передается обработчику данного вызова, входящему в ядро операционной системы. Цель таких библиотек – сделать системный вызов похожим на обычный вызов подпрограммы.

Основное отличие состоит в том, что при системном вызове задача переходит в привилегированный режим или режим ядра (*kernel mode*). Поэтому системные вызовы иногда еще называют программными прерываниями, в отличие от аппаратных прерываний, которые чаще называют просто прерываниями.

В этом режиме работает код ядра операционной системы, причем исполняется он в адресном пространстве и в контексте вызвавшей его задачи. Таким образом, ядро операционной системы имеет полный доступ к памяти пользовательской программы, и при системном вызове достаточно передать адреса одной или нескольких областей памяти с параметрами вызова и адреса одной или нескольких областей памяти для результатов вызова.

В большинстве операционных систем системный вызов осуществляется командой программного прерывания (*INT*). Таким образом, программное прерывание – это синхронное событие.

Прерывание (*hardware interrupt*) – это событие, генерируемое внешним (по отношению к процессору) устройством. Посредством аппаратных прерываний аппаратура либо информирует центральный процессор о том, что произошло какое-либо событие, требующее немедленной реакции (например,

пользователь нажал клавишу), либо сообщает о завершении асинхронной операции «ввода-вывода» (например, закончено чтение данных с диска в основную память). Важный тип аппаратных прерываний – прерывания таймера, которые генерируются периодически через фиксированный промежуток времени. Прерывания таймера используются операционной системой при планировании процессов. Каждый тип аппаратных прерываний имеет собственный номер, однозначно определяющий источник прерывания. Аппаратное прерывание – это асинхронное событие, то есть оно возникает вне зависимости от того, какой код исполняется процессором в данный момент. Обработка аппаратного прерывания не должна учитывать, какой процесс является текущим.

Исключительная ситуация (*exception*) – событие, возникающее в результате попытки выполнения программой команды, которая по каким-то причинам не может быть выполнена до конца. Примерами таких команд могут быть попытки доступа к ресурсу при отсутствии достаточных привилегий или обращения к отсутствующей странице памяти. Исключительные ситуации, как и системные вызовы, являются синхронными событиями, возникающими в контексте текущей задачи. Исключительные ситуации можно разделить на исправимые и неисправимые. К исправимым относятся такие исключительные ситуации, как отсутствие нужной информации в оперативной памяти. После устранения причины исправимой исключительной ситуации программа может выполняться дальше. Возникновение в процессе работы операционной системы исправимых исключительных ситуаций считается нормальным явлением. Неисправимые исключительные ситуации чаще всего возникают в результате ошибок в программах (например, деление на ноль). Обычно в таких случаях операционная система реагирует завершением программы, вызвавшей исключительную ситуацию.

Файлы предназначены для хранения информации на внешних носителях, то есть принято, что информация, записанная, например, на диске, должна находиться внутри файла. Обычно под файлом понимают именованную часть пространства на носителе информации.

Главная задача файловой системы (*file system*) – скрыть особенности «ввода-вывода» и дать программисту простую абстрактную модель файлов, независимых от устройств. Для чтения, создания, удаления, записи, открытия и закрытия файлов также имеется обширная категория системных вызовов (создание, удаление, открытие, закрытие, чтение и т.д.). Пользователям хорошо знакомы такие связанные с организацией файловой системы понятия, как каталог, текущий каталог, корневой каталог, путь. Для манипулирования этими объектами в операционной системе имеются системные вызовы.

Процессы, нити. Концепция процесса в ОС - одна из наиболее фундаментальных. Процессы подробно рассмотрены в следующих лекциях.

Функции ОС.

Все многообразие программ, используемых на современном компьютере, называется программным обеспечением - ПО (*software*).

Программы, составляющие ПО, можно разделить на три группы: системное ПО, системы программирования, прикладное ПО. Ядром системного ПО является операционная система (ОС).

ОС - это неотъемлемая часть ПО, управляющая техническими средствами компьютера (*hardware*). Операционная система - это программа, координирующая действия вычислительной машины; под ее управлением осуществляется выполнение программ.

К функциям ОС относят:

- осуществление диалога с пользователем;
- ввод-вывод и управление данными;
- планирование и организация процесса обработки программ;
- распределение ресурсов (оперативной памяти, процессора, внешних устройств);
- запуск программ на выполнение;
- всевозможные вспомогательные операции обслуживания;
- передача информации между различными внутренними устройствами;
- программная поддержка работы периферийных устройств (дисплея, клавиатуры, принтера и др.).

Контрольные вопросы

1 Дайте классификацию компонентов ОС.
2 Каково назначение ОС?
3 Что такое системные функции в *Linux*?
4 Перечислите функции ОС.

Лекция №6. Требования и принципы разработки архитектуры систем

Цель лекции: ознакомить студентов с требованиями, используемыми при разработке операционных систем.

Содержание лекции: требования к разработке операционных систем; важность критериев разделения операционных систем.

К современным системам предъявляются следующие требования:

- совместимость - ОС должна включать средства для выполнения приложений, подготовленных для других ОС;
- переносимость - обеспечение возможности переноса ОС с одной аппаратной платформы на другую;
- надежность и отказоустойчивость предполагает защиту ОС от внутренних и внешних ошибок, сбоев и отказов;
- безопасность - ОС должна содержать средства защиты ресурсов одних пользователей от других;

– расширяемость - ОС должна обеспечивать удобство внесения последующих изменений и дополнений;

– производительность - система должна обладать достаточным быстродействием.

Многозадачность и распределение полномочий требуют определённой иерархии привилегий компонентов в самой ОС:

– ядро, содержащее планировщик; драйверы устройств, управляющие оборудованием; сетевая подсистема, файловая система;

– системные библиотеки;

– оболочка с утилитами.

Принципы разработки архитектуры операционных систем:

– состав модулей (компонент) ОС;

– структура связей между отдельными модулями ОС;

– принципы взаимодействия модулей ОС;

– принципы функционирования ОС в плане выполнения отдельных функций и в целом.

Контрольные вопросы

1 Перечислите требования к разработке операционных систем.
2 Поясните суть каждого требования.
3 Раскройте принципы разработки архитектуры операционных систем.

Лекция №7. Режимы работы Linux. Виртуальные машины

Цель лекции: ознакомить студентов с режимами работы ОС Linux и виртуальными машинами.

Содержание лекции: режимы работы ОС *Linux*; виртуальные машины.

По уровню привилегий работы *Linux* различают:

- режим пользователя (прикладной): минимальный уровень привилегий, разрешены только операции с данными и переходы в пределах адресного пространства пользователя. Все остальные операции либо игнорируются, либо с помощью механизма обработки исключений вызывают переключение в привилегированный режим и передачу управления ядру операционной системы для выполнения специальных функций (например, отображения данных на дисплее) или аварийного завершения потока управления;

- привилегированный режим (режим ядра): наравне с операциями режима пользователя, разрешены дополнительные «операции – запрет» или разрешение прерываний, доступ к портам «ввода-вывода», специальным регистрам процессора (например, для настройки блока управления памятью).

Переключение с понижением уровня привилегий (из режима ядра в режим пользователя) возможно с продолжением работы в любой конфигурации и с любого адреса. Обратное переключение (с повышением уровня привилегий) возможно лишь в определенные конфигурации и адреса в коде, заранее установленные в привилегированном режиме, за счет чего обеспечивается полный контроль ядра за повышением привилегий исполняемого кода.

Режим супервизора, режим ядра (англ. *kernel mode*) - привилегированный режим работы процессора, используемый для выполнения ядра ОС. Доступны привилегированные операции «ввода-вывода» к периферийным устройствам, изменение параметров защиты памяти, настроек виртуальной памяти, системных параметров и прочих параметров конфигурации.

Виртуальная машина эмулирует работу реального компьютера. На виртуальную машину можно устанавливать *Linux*, есть свой *BIOS*, оперативная память, жёсткий диск (выделенное место на жёстком диске реального ПК), могут эмулироваться периферийные устройства. На одном компьютере может функционировать несколько виртуальных машин.

Виртуальные машины могут использоваться:

- для защиты информации и ограничения возможностей процессов;
- для исследования производительности ПО или новой архитектуры;
- для эмуляции различных архитектур (например, эмулятор приставки).

Контрольные вопросы

1 Дайте классификацию режимам загрузки *Linux*.
2 Что такое виртуальная машина?

Лекция №8. Множественные прикладные среды. Совместимость

Цель лекции: ознакомить студентов с прикладными средами, совместимостью.

Содержание лекции: раскрытие архитектурных и функциональных особенностей ОС; совместимость, типы совместимости.

Конкретные архитектурные и функциональные особенности любой ОС непосредственно должны касаться лишь системных программистов и могут совершенно не быть известны рядовому пользователю. В то время как некоторые идеи (например, объектно-ориентированный подход) находятся в ведении только разработчиков и лишь косвенно влияют на конечного пользователя, концепция множественных прикладных сред приносит пользователю долгожданную возможность выполнять на своей ОС программы, написанные для других ОС и процессоров.

Свойство ОС, характеризующее возможность выполнения в ОС приложений, написанных для других ОС называется совместимостью.

Существует 2 вида совместимости: совместимость на двоичном уровне и совместимость на уровне исходных текстов. Приложения хранятся в виде исполняемых файлов, содержащих двоичные образы кодов. Двоичная совместимость достигается, если можно взять исполняемую программу, работающую в среде одной ОС, и запустить ее на выполнение в другой ОС.

Совместимость на уровне исходных текстов требует наличия соответствующих компиляторов в составе программного обеспечения компьютера, на котором предполагается использовать данное приложение, а также совместимости на уровне библиотек и системных вызовов. При этом необходима перекомпиляция исходных текстов программ в новые исполняемые модули.

Таким образом, совместимость на уровне исходных текстов наиболее важное значение имеет для разработчиков приложений, в распоряжении которых находятся эти исходные тексты. Для конечных же пользователей практическое значение имеет только двоичная совместимость, так как только в этом случае они могут без специальных навыков и умений использовать программный продукт, поставляемый в виде двоичного исполняемого кода, в различных операционных средах и на разных компьютерах. Множественные прикладные среды как раз и обеспечивают совместимость данной ОС с приложениями, написанными для других ОС и процессоров, на двоичном уровне, а не на уровне исходных текстов.

Каким типом совместимости - двоичной или совместимостью исходных текстов обладает ОС, зависит от многих факторов. Самый значительный из них - архитектура процессора, на котором работает ОС. Чтобы достичь двоичной совместимости, достаточно соблюсти несколько следующих условий:

- вызовы функций *API*, которые содержит приложение, должны поддерживаться дайной ОС;
- внутренняя структура исполняемого файла приложения должна соответствовать структуре исполняемых файлов данной ОС.

Несравнимо сложнее достигнуть двоичной совместимости операционным системам, предназначенным для выполнения на процессорах, имеющих различающиеся архитектуры. Кроме соблюдения приведенных выше условий, необходимо также организовать эмуляцию двоичного кода. Для того чтобы компьютер смог интерпретировать машинные инструкции, которые ему изначально непонятны, на нем должно быть установлено специальное программное обеспечение - эмулятор.

Назначение эмулятора состоит в том, чтобы последовательно выбирать каждую двоичную инструкцию процессора, например, *Intel*, программным способом дешифровать ее, чтобы определить, какие действия она задает, а затем выполнять эквивалентную подпрограмму, написанную в инструкциях процессора, например, *Motorola*.

Тем не менее, существует другой, гораздо более эффективный выход из описанной ситуации - использование так называемых прикладных программных сред. Одной из составляющих, формирующих программную среду, является набор функций интерфейса прикладного программирования *API*, которым ОС обеспечивает свои приложения. Для сокращения времени выполнения чужих программ прикладные среды имитируют обращения к библиотечным функциям. Эффективность данного подхода определяется тем, что большинство современных программ работают под управлением графических интерфейсов пользователя (*GUI*) типа *Windows, UNIX* при этом приложения, как правило, наибольшую часть времени тратят на выполнение, некоторых хорошо предсказуемых действий. Они непрерывно осуществляют вызовы библиотек *GUI* для манипулирования окнами и для других, связанных с *GUI*, действий. Именно эта особенность приложений позволяет прикладным средам компенсировать большие затраты времени, потраченные на покомандное эмулирование программы. Тщательно спроектированная программная среда имеет в своем составе библиотеки, имитирующие внутренние библиотеки *GUI*, но написанные на «родном» коде данной ОС. Таким образом, достигается существенное ускорение выполнения программ с *API* другой операционной системы. Для того чтобы отличить такой подход от более медленного процесса эмулирования кода по одной команде за раз, его называют трансляцией.

Для того чтобы программа, написанная для одной ОС, могла быть выполнена в рамках другой ОС, недостаточно лишь обеспечить совместимость *API*. Вполне может случиться так, что концепции, положенные в основу разных ОС, войдут в противоречие друг с другом. Например, в одной ОС приложению может быть разрешено непосредственно управлять устройствами «ввода-вывода», а в другой действия являются прерогативой ОС. Совершенно естественно, что каждая ОС имеет свои собственные механизмы защиты ресурсов: свои алгоритмы обработки ошибок и исключительных ситуаций; особую структуру процесса и схему управления памятью; свою семантику доступа к файлам и графический пользовательский интерфейс. Все эти отличия определяются спецификой аппаратной платформы, на которой работает ОС, особенностями ее реализации или заложены разработчиками системы как присущие данной системе свойства. Для обеспечения совместимости необходимо организовать бесконфликтное сосуществование в рамках одной ОС нескольких способов управления ресурсами компьютера.

Существует способ построения множественных прикладных сред, использующий концепцию микроядерного подхода. Все функции ОС реализуются микроядром и серверами пользовательского режима. Каждая прикладная среда оформляется и виде отдельного сервера пользовательского режима и не включает базовых механизмов. Приложения, используя *API*, обращаются с системными вызовами к соответствующей прикладной среде через микроядро. Прикладная среда обрабатывает запрос, выполняет его и отсылает приложению результат. В ходе выполнения запроса

прикладной среде приходится, в свою очередь, обращаться к базовым механизмам ОС, реализуемым микроядром и другими серверами ОС.

Такому подходу к конструированию множественных прикладных средств присущи достоинства и недостатки микроядерной архитектуры:

- очень просто можно добавлять и исключать прикладные среды, что является следствием хорошей расширяемости микроядерных ОС;
- надежность и стабильность выражаются в том, что при отказе одной из прикладных сред все остальные сохраняют работоспособность;
- низкая производительность микроядерных ОС сильно сказывается на скорости работы прикладных сред, а значит, и на скорости выполнения приложений.

Чтобы программа, написанная для одной ОС, могла быть выполнена в рамках другой ОС, недостаточно лишь совместимости *API*. Кроме этого, ей необходимо «родное» окружение: структура процесса, средства управления памятью, средства обработки ошибок и исключительных ситуаций, механизмы защиты ресурсов и семантика файлового доступа. Поддержка нескольких прикладных программных сред является очень сложной задачей, тесно связанной со структурой ОС.

Контрольные вопросы

1 Раскройте понятие совместимости.
2 Назовите способы реализации совместимости.
3 Для чего существуют эмуляторы?

Лекция №9. Организация и управление процессами Linux

Цель лекции: ознакомить студентов с процессами, способами управления ими.

Содержание лекции: понятие ресурса и процесса; приоритеты; мультипроцессорная обработка.

Процесс является экземпляром компьютерной программы, которая содержит программный код и код его текущей деятельности. В зависимости от используемой операционной системы процесс может быть составлен из нескольких потоков выполнения, которые выполняют инструкции одновременно.

Понятие процесса характеризует некоторую совокупность набора исполняющихся команд, ассоциированных с ним ресурсов (выделенная для исполнения память или адресное пространство, стеки, используемые файлы и устройства ввода-вывода и т.д.) и текущего момента его выполнения (значения регистров, программного счетчика, состояние стека и значения переменных), находящуюся под управлением операционной системы. Не

существует взаимно-однозначного соответствия между процессами и программами, обрабатываемыми вычислительными системами. В некоторых операционных системах для работы определенных программ можно организовывать более одного процесса или один и тот же процесс может исполнять последовательно несколько различных программ. Более того, даже в случае обработки только одной программы в рамках одного процесса нельзя считать, что процесс представляет собой просто динамическое описание кода исполняемого файла, данных и выделенных для них ресурсов. Процесс находится под управлением операционной системы, поэтому в нем может выполняться часть кода ее ядра (не находящегося в исполняемом файле!), как в случаях, специально запланированных авторами программы (например, при использовании системных вызовов), так и в непредусмотренных ситуациях (например, при обработке внешних прерываний).

Многозадачность (*multitasking*) - свойство операционной системы или среды программирования, обеспечивать возможность параллельной (или псевдопараллельной) обработки нескольких процессов. Истинная многозадачность операционной системы возможна только в распределенных вычислительных системах.

Примитивные многозадачные среды обеспечивают чистое «разделение ресурсов», когда за каждой задачей закрепляется определённый участок памяти, и задача активизируется в строго определённые интервалы времени.

Более развитые многозадачные системы проводят распределение ресурсов динамически, когда задача стартует в памяти или покидает память в зависимости от её приоритета и от стратегии системы. Такая многозадачная среда обладает следующими особенностями:

а) каждая задача имеет свой приоритет, в соответствии с которым получает время и память:

– система организует очереди задач так, чтобы все задачи получили ресурсы, в зависимости от приоритетов и стратегии системы;

– система организует обработку прерываний, по которым задачи могут активироваться, деактивироваться и удаляться;

– по окончании положенного кванта времени задача может временно выбрасываться из памяти, отдавая ресурсы другим задачам, а потом через определённое системой время, восстанавливаться в памяти (свопинг);

– система обеспечивает защиту памяти от несанкционированного вмешательства других задач;

– система распознаёт сбои и зависания отдельных задач и прекращает их;

– система решает конфликты доступа к ресурсам и устройствам, не допуская тупиковых ситуаций общего зависания от ожидания заблокированных ресурсов;

– система гарантирует каждой задаче, что рано или поздно она будет активирована;

– система обрабатывает запросы реального времени;
– система обеспечивает коммуникацию между процессами;

б) невытесняющая многозадачность - тип многозадачности, при котором операционная система одновременно загружает в память два или более приложений, но процессорное время предоставляется только основному приложению. Для выполнения фонового приложения оно должно быть активизировано;

в) совместная или кооперативная многозадачность - тип многозадачности, при котором фоновые задачи выполняются только во время простоя основного процесса и только в том случае, если на это получено разрешение основного процесса.

Мультипроцессорная обработка - это такой способ организации вычислительного процесса в системах с несколькими процессорами, при котором несколько задач (процессов, потоков) могут одновременно выполняться на разных процессорах системы.

Контрольные вопросы

1 Каково понятие процесса?
2 Что характеризует процесс?
3 Что такое приоритет?

Лекция №10. Представление процесса в Linux

Цель лекции: донести до студента суть процессов в ОС.

Содержание лекции: процессы в системе, задания; состояния процессов.

Рассмотрим следующий пример. Если запустить программу извлечения квадратного корня, то, извлекая значения корня с разными исходными данными, компьютерной система, заниматься двумя различными вычислительными процессами, так как разные исходные данные приводят к разному набору вычислений. С точки зрения пользователя, запущена одна и та же программа, но сформированы два различных задания. Теперь другой пример. Пусть оба пользователя извлекают корень квадратный из 1, то есть они сформировали идентичные задания, но загрузили их в вычислительную систему со сдвигом по времени. В то время как одно из выполняемых заданий приступило к печати полученного значения и ждет окончания операции «ввода-вывода», второе только начинает исполняться. Можно ли говорить об идентичности заданий внутри вычислительной системы в данный момент? Нет, так как состояние процесса их выполнения различно. Следовательно, «программа» и «задание» в пользовательском смысле не может применяться для описания происходящего в вычислительной системе. Это происходит

потому, что термины «программа» и «задание» предназначены для описания статических, неактивных объектов. Программа же в процессе исполнения является динамическим, активным объектом. В процессе работы компьютер обрабатывает различные команды и преобразует значения переменных. Для выполнения программы ОС должна выделить определенное количество оперативной памяти, закрепить за ней определенные устройства «ввода-вывода» или файлы (откуда должны поступать входные данные и куда нужно доставить полученные результаты), то есть зарезервировать определенные ресурсы из общего числа ресурсов всей вычислительной системы. Их количество и конфигурация с течением времени могут изменяться. Для описания таких активных объектов внутри компьютерной системы вместо терминов «программа» и «задание» будет использоваться термин «процесс».

Процесс характеризует некоторую совокупность набора исполняющихся команд, ассоциированных с ним ресурсов (выделенная для исполнения память или адресное пространство, стеки, используемые файлы и устройства ввода-вывода и т. д.) и текущего момента его выполнения (значения регистров, программного счетчика, состояние стека и значения переменных), находящуюся под управлением операционной системы.

Процесс не может перейти из одного состояния в другое самостоятельно. Изменением состояния процессов занимается операционная система, совершая операции над ними. Количество таких операций в модели пока совпадает с количеством стрелок на диаграмме состояний (рисунок 2).

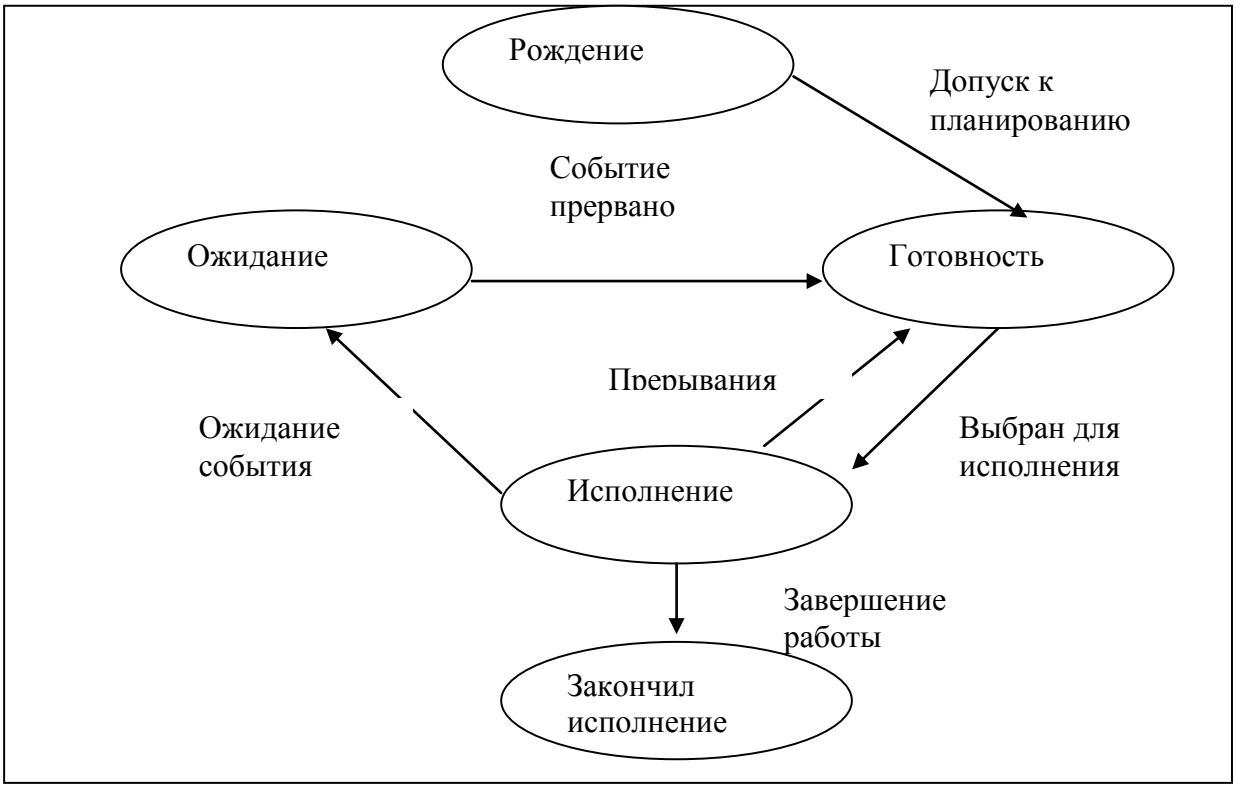

Рисунок 2 – Диаграмма состояний процессов

Удобно объединить их в три пары:

- создание процесса - завершение процесса;

– приостановка процесса (перевод из состояния исполнение в состояние готовность) - запуск процесса (перевод из состояния готовность в состояние исполнение);

– блокирование процесса (перевод из состояния исполнение в состояние ожидание) - разблокирование процесса (перевод из состояния ожидание в состояние готовность).

Еще есть операция, не имеющая пары: изменение приоритета процесса.

Операции создания и завершения процесса являются одноразовыми, так как применяются к процессу не более одного раза (некоторые системные процессы при работе вычислительной системы не завершаются никогда). Все остальные операции, связанные с изменением состояния процессов, будь то запуск или блокировка, как правило, являются многоразовыми.

Контрольные вопросы

1 Что характеризует процесс?
2 Перечислите состояния процессов в *Linux*.
3 Какое состояние процесса в ОС не имеет пары?

Лекция №11. Управление процессами

Цель лекции: ознакомить студента с механизмами управления процессами в *Linux*.

Содержание лекции: состояния процессов; управление, одноразовые, многоразовые операции.

Все, что выполняется в вычислительных системах (не только программы пользователей, но и определенные части ОС), организовано как набор процессов. На однопроцессорной компьютерной системе в каждый момент времени может исполняться только один процесс. Для мультипрограммных вычислительных систем псевдопараллельная обработка нескольких процессов достигается с помощью переключения процессора с одного процесса на другой. Пока один процесс выполняется, остальные ждут своей очереди.

В конкретных операционных системах состояния процесса могут быть еще более детализированы, могут появиться некоторые варианты переходов из одного состояния в другое. Так, например, модель состояний процессов для *Windows NT* содержит 7 состояний, а для *Unix* - 9. Для *Linux* - 7:

– создание процесса (переход из состояния рождения в состояние готовности);

– уничтожение процесса (переход из состояния выполнения в состояние смерти);

– восстановление процесса (переход из состояния готовности в состояние выполнения);

– изменение приоритета процесса (переход из выполнения в готовность);

– блокирование процесса (переход в состояние ожидания из состояния выполнения);

– пробуждение процесса (переход из состояния ожидания в состояние готовности);

– запуск процесса /или его выбор (переход из состояния готовности в состояние выполнения).

Для создания процесса системе нужно:

– присвоить процессу имя;

– добавить информацию о процессе в список процессов;

– определить приоритет процесса;

– сформировать блок управления процессом;

– предоставить процессу нужные ему ресурсы.

Process Control Block и контекст процесса.

В ОС каждый процесс представляется в ней структурой данных. Эта структура содержит информацию, специфическую для данного процесса:

– состояние, в котором находится процесс;

– программный счетчик процесса или, другими словами, адрес команды, которая должна быть выполнена для него следующей;

– содержимое регистров процессора;

– данные, необходимые для планирования использования процессора и управления памятью (приоритет процесса, размер и расположение адресного пространства и т. д.);

– учетные данные (идентификационный номер процесса, какой пользователь инициировал его работу, общее время использования процессора данным процессом и т. д.);

– сведения об устройствах «ввода-вывода», связанных с процессом (например, какие устройства закреплены за процессом, таблицу открытых файлов).

Состав и строение ОС зависят от конкретной ОС. Во многих ОС информация, характеризующая процесс, хранится не в одной, а в нескольких связанных структурах данных. Эти структуры могут иметь различные наименования, содержать дополнительную информацию или лишь часть описанной информации, для любого процесса, находящегося в вычислительной системе: вся информация, необходимая для совершения операций над ним, доступна ОС и хранится в одной структуре данных. Она называется *PCB (Process Control Block)* или блоком управления процессом. Блок управления процессом является моделью процесса для ОС. Любая операция, производимая операционной системой над процессом, вызывает определенные изменения в *PCB*. В рамках принятой модели состояний процессов содержимое *PCB* между операциями остается постоянным.

Информацию, для хранения которой предназначен блок управления процессом, удобно для дальнейшего изложения разделить на две части.

Содержимое всех регистров процессора (включая значение программного счетчика) будем называть регистровым контекстом процесса, а все остальное – системным контекстом процесса. Знания регистрового и системного контекстов процесса достаточно для того, чтобы управлять его работой в операционной системе, совершая над ним операции. Однако этого недостаточно для того, чтобы полностью охарактеризовать процесс. ОС не интересует, какими именно вычислениями занимается процесс, т. е. какой код и какие данные находятся в его адресном пространстве. С точки зрения пользователя, наоборот, наибольший интерес представляет содержимое адресного пространства процесса, возможно, наряду с регистровым контекстом определяющее последовательность преобразования данных и полученные результаты. Код и данные, находящиеся в адресном пространстве процесса, будем называть пользовательским контекстом. Совокупность регистрового, системного и пользовательского контекстов процесса для краткости принято называть просто контекстом процесса. В любой момент времени процесс полностью характеризуется своим контекстом.

Одноразовые операции. Сложные ОС создают процессы динамически, по мере необходимости. Инициатором рождения нового процесса после старта операционной системы может выступить либо процесс пользователя, совершивший специальный системный вызов, либо сама операционная система, то есть, в конечном итоге, тоже некоторый процесс. Процесс, инициировавший создание нового процесса, принято называть процессом-родителем (*parent process*), а вновь созданный процесс – процессом-ребенком (*child process*). Процессы-дети могут, в свою очередь, порождать новых детей и т.д., образуя, в общем случае, внутри системы набор генеалогических деревьев процессов – генеалогический лес. Следует отметить, что все пользовательские процессы вместе с некоторыми процессами операционной системы принадлежат одному и тому же дереву леса. В ряде вычислительных систем лес вообще вырождается в одно такое дерево. При рождении процесса система заводит новый *PCB* с состоянием процесса рождения и начинает его заполнять. Новый процесс получает собственный уникальный идентификационный номер. Поскольку для хранения идентификационного номера процесса в операционной системе отводится ограниченное количество битов, для соблюдения уникальности номеров количество одновременно присутствующих в ней процессов должно быть ограничено. После завершения какого-либо процесса его освободившийся идентификационный номер может быть повторно использован для другого процесса.

Обычно для выполнения своих функций процесс-ребенок требует определенных ресурсов: памяти, файлов, устройств «ввода-вывода» и т.д. Существует два подхода к их выделению. Новый процесс может получить в свое распоряжение некоторую часть родительских ресурсов, возможно, разделяя с процессом-родителем и другими процессами-детьми права на них, или может получить свои ресурсы непосредственно от ОС. Информация о выделенных ресурсах заносится в *PCB*. После наделения процесса-ребенка ресурсами необходимо занести в его адресное пространство программный

код, значения данных, установить программный счетчик. Здесь также возможны два решения. В первом случае процесс-ребенок становится дубликатом процесса-родителя по регистровому и пользовательскому контекстам, при этом должен существовать способ определения, кто для кого из процессов-двойников является родителем. Во втором случае процесс-ребенок загружается новой программой из какого-либо файла. *Unix* разрешает порождение процесса только первым способом; для запуска новой программы необходимо сначала создать копию процесса-родителя, а затем процесс-ребенок должен заменить свой пользовательский контекст с помощью специального системного вызова. ОС *VAX*/*VMS* допускает только второе решение. В *Windows NT* возможны оба варианта (в различных *API*).

Порождение нового процесса как дубликата процесса-родителя приводит к возможности существования программ (т.е. исполняемых файлов), для работы которых организуется более одного процесса. Возможность замены пользовательского контекста процесса по ходу его работы (т. е. загрузки для исполнения новой программы) приводит к тому, что в рамках одного и того же процесса может последовательно выполняться несколько различных программ. Процесс-родитель может продолжать свое выполнение одновременно с выполнением процесса-ребенка, а может ожидать завершения работы некоторых или всех своих «детей».

Многоразовые операции не приводят к изменению количества процессов в ОС и не обязаны быть связанными с выделением/ освобождением ресурсов.

Запуск процесса. Из числа процессов, находящихся в состоянии «готовность», ОС выбирает один процесс для последующего исполнения. Для этого процесса ОС обеспечивает наличие в оперативной памяти информации для его выполнения. Далее состояние процесса изменяется: восстанавливаются значения регистров для процесса и управление передается команде, на которую указывает счетчик команд процесса. Данные для восстановления контекста извлекаются из *PCB* процесса.

Приостановка процесса. Процессор автоматически сохраняет счетчик команд и один или несколько регистров в стеке исполняемого процесса, а затем передает управление по специальному адресу обработки данного прерывания.

Блокирование процесса. Процесс блокируется, когда он не может продолжать работу, не дождавшись возникновения какого-либо события в системе. Для этого он обращается к ОС с помощью определенного системного вызова. ОС обрабатывает системный вызов (инициализирует операцию «ввода-вывода», добавляет процесс в очередь процессов, дожидающихся освобождения устройства или возникновения события, и т. д.) и, при необходимости, сохранив нужную часть контекста процесса в его *PCB*, переводит процесс из состояния исполнение в состояние ожидание.

Разблокирование процесса. После возникновения в системе какого-либо события нужно точно определить, какое именно событие произошло. Затем ОС проверяет, находился ли некоторый процесс в состоянии «ожидание» для

данного события, и если находился, переводит его в состояние «готовность», выполняя необходимые действия, связанные с наступлением события.

Переключение контекста.

Деятельность мультипрограммной ОС состоит из цепочек операций, выполняемых над различными процессами, и сопровождается переключением процессора с одного процесса на другой. При исполнении процессором некоторого процесса возникает прерывание от устройства «ввода-вывода», сигнализирующее об окончании операций на устройстве. Над выполняющимся процессом производится операция приостановки. Далее ОС разблокирует процесс, инициировавший запрос на «ввод-вывод», и осуществляет запуск приостановленного или нового процесса, выбранного при выполнении планирования. В результате обработки информации об окончании операции «ввода-вывода» возможна смена процесса, находящегося в состоянии «исполнение».

Контрольные вопросы

1 Что такое процесс в пределах операционной составляющей?
2 Какие вы знаете режимы работы процессов?
3 Что в себе содержит контекст процесса?

Лекция №12. Планирование процессов. Уровни и типы планирования

Цель лекции: ознакомить студентов с уровнями, критериями планирования процессами в ОС.

Содержание лекции: уровни планирования процессов в ОС; типы планирования.

Процедура выбора очередного задания для загрузки в машину, т. е. для порождения соответствующего процесса, называется планированием заданий.

Планирование заданий используется в качестве долгосрочного планирования процессов. Оно отвечает за порождение новых процессов в системе, определяя ее степень мультипрограммирования, т.е. количество процессов, одновременно находящихся в ней. Если степень мультипрограммирования системы поддерживается постоянной, т.е. среднее количество процессов в компьютере не меняется, то новые процессы могут появляться только после завершения ранее загруженных. Поэтому долгосрочное планирование осуществляется достаточно редко: между появлением новых процессов могут проходить минуты и даже десятки минут. Решение о выборе для запуска того или иного процесса оказывает влияние на функционирование вычислительной системы на протяжении достаточно

длительного времени. Отсюда и название этого уровня планирования – долгосрочное.

Планирование использования процессора применяется в качестве краткосрочного планирования процессов. Оно проводится, к примеру, при обращении исполняющегося процесса к устройствам «ввода-вывода» или просто по завершении определенного интервала времени. Поэтому краткосрочное планирование осуществляется, как правило, не реже одного раза в 100 миллисекунд. Выбор нового процесса для исполнения оказывает влияние на функционирование системы до наступления очередного аналогичного события, т.е. в течение короткого промежутка времени, чем и обусловлено название этого уровня планирования – краткосрочное.

В некоторых вычислительных системах бывает выгодно для повышения производительности временно удалить какой-либо частично выполнившийся процесс из оперативной памяти на диск, а позже вернуть его обратно для дальнейшего выполнения. Такая процедура получила название *swapping*, (перекачка), хотя в специальной литературе оно употребляется без перевода – свопинг. Когда и какой из процессов нужно перекачать на диск и вернуть обратно, решается дополнительным промежуточным уровнем планирования процессов – среднесрочным.

Критерии планирования.

Для каждого уровня планирования процессов можно предложить много различных алгоритмов. Выбор конкретного алгоритма определяется классом задач, решаемых вычислительной системой, и целями, которых мы хотим достичь, используя планирование. К числу таких целей можно отнести следующие:

– справедливость – гарантировать каждому заданию или процессу определенную часть времени использования процессора в компьютерной системе, стараясь не допустить возникновения ситуации, когда процесс одного пользователя постоянно занимает процессор, в то время как процесс другого пользователя фактически не начинал выполняться;

– эффективность – постараться занять процессор на все 100% рабочего времени, не позволяя ему простаивать в ожидании процессов, готовых к исполнению. В реальных системах загрузка процессора - от 40 до 90%;

– сокращение полного времени выполнения (*turnaround time*) – обеспечить минимальное время между стартом процесса или постановкой задания в очередь для загрузки и его завершением;

– сокращение времени ожидания (*waiting time*) – сократить время, которое проводят процессы в состоянии «готовность» и задания в очереди для загрузки;

– сокращение времени отклика (*response time*) – минимизировать время, которое требуется процессу в интерактивных системах для ответа на запрос пользователя.

Независимо от поставленных целей планирования желательно также, чтобы алгоритмы обладали следующими свойствами:

– были предсказуемыми. Одно и то же задание должно выполняться приблизительно за одно и то же время. Применение алгоритма планирования не должно приводить, к примеру, к извлечению корня квадратного из 4 за сотые доли секунды при одном запуске и за несколько суток – при втором запуске;

– были связаны с минимальными накладными расходами. Если на каждые 100 миллисекунд, выделенные процессу для использования процессора, будет приходиться 200 миллисекунд на определение того, какой именно процесс получит процессор в свое распоряжение, и на переключение контекста, то такой алгоритм, очевидно, применять не стоит;

– равномерно загружали ресурсы вычислительной системы, отдавая предпочтение тем процессам, которые будут занимать малоиспользуемые ресурсы;

– обладали масштабируемостью, т.е. не сразу теряли работоспособность при увеличении нагрузки. Например, рост количества процессов в системе в два раза не должен приводить к увеличению полного времени выполнения процессов на порядок.

Контрольные вопросы

1 Каковы критерии планирования?
2 Что такое планирование использования процессора?
3 Расскажите про типы планирования процессов.

Лекция №13. Логическая организация файловой системы

Цель лекции: ознакомить студента с устройством логической организацией файловых систем.

Содержание лекции: файловая система; функции файловой системы.

Файловая система - это часть ОС, назначение которой состоит в том, чтобы организовать эффективную работу с данными, хранящимися во внешней памяти, и обеспечить пользователю удобный интерфейс при работе с такими данными. Для того чтобы избавить пользователя компьютера от сложностей взаимодействия с аппаратурой, была придумана ясная абстрактная модель файловой системы. Операции записи или чтения файла концептуально проще, чем низкоуровневые операции работы с устройствами.

Файловая система позволяет при помощи системы справочников (каталогов, директорий) связать уникальное имя файла с блоками вторичной памяти, содержащими данные файла. Иерархическая структура каталогов, используемая для управления файлами, может служить другим примером индексной структуры. В этом случае каталоги или папки играют роль индексов, каждый из которых содержит ссылки на свои подкаталоги. С этой

точки зрения вся файловая система компьютера представляет собой большой индексированный файл. Помимо собственно файлов и структур данных, используемых для управления файлами (каталоги, дескрипторы файлов, различные таблицы распределения внешней памяти), понятие «файловая система» включает программные средства, реализующие различные операции над файлами.

Основные функции файловой системы:

а) идентификация файлов - связывание имени файла с выделенным ему пространством внешней памяти;

б) распределение внешней памяти между файлами - для работы с конкретным файлом не требуется иметь информацию о местоположении этого файла на внешнем носителе информации;

в) обеспечение надежности и отказоустойчивости - стоимость информации может во много раз превышать стоимость компьютера;

г) обеспечение защиты от несанкционированного доступа;

д) обеспечение совместного доступа к файлам так, чтобы пользователю не приходилось прилагать специальных усилий по обеспечению синхронизации доступа;

е) обеспечение высокой производительности.

Важный аспект организации файловой системы - учет стоимости операций взаимодействия с вторичной памятью. Процесс считывания блока диска состоит из позиционирования считывающей головки над дорожкой, содержащей требуемый блок, ожидания, пока требуемый блок сделает оборот и окажется под головкой, и собственно считывания блока. Для этого требуется значительное время (десятки миллисекунд). В компьютерах обращение к диску осуществляется примерно в 100 000 раз медленнее, чем обращение к оперативной памяти. Таким образом, критерием вычислительной сложности алгоритмов с внешней памятью, является количество обращений к диску.

Правила именования файлов зависят от ОС. Многие ОС поддерживают имена из двух частей (имя+расширение). Тип расширения файла позволяет ОС организовать работу с ним различных прикладных программ в соответствии с заранее оговоренными соглашениями. Обычно ОС накладывают некоторые ограничения как на используемые в имени символы, так и на длину имени файла. В соответствии со стандартом *POSIX*, популярные ОС оперируют удобными для пользователя длинными именами (до 255 символов).

Важный аспект организации файловой системы и ОС - следует ли поддерживать и распознавать типы файлов. Если да, то это может помочь правильному функционированию ОС, например, не допустить вывода на принтер бинарного файла.

Основные типы файлов: регулярные (обычные) файлы и директории (справочники, каталоги). Обычные файлы содержат пользовательскую информацию. Директории - системные файлы, поддерживающие структуру файловой системы. В каталоге содержится перечень входящих в него файлов

и устанавливается соответствие между файлами и их характеристиками (атрибутами).

Классификация файловых систем по предназначению:

– для носителей с произвольным доступом (например, жёсткий диск): *FAT32*, *HPFS*, *ext2* и др. Поскольку доступ к дискам в несколько раз медленнее, чем доступ к оперативной памяти, для прироста производительности во многих файловых системах применяется асинхронная запись изменений на диск. Для этого применяется либо журналирование, например, в *ext3*, *ReiserFS*, *JFS*, *NTFS*, *XFS*, либо механизм *soft updates* и др. Журналирование широко распространено в *Linux*, применяется в *NTFS*. *Soft updates* - в *BSD*-системах;

– для носителей с последовательным доступом (магнитные ленты): *QIC*;

– для оптических носителей - *CD* и *DVD*: *ISO9660*, *HFS*, *UDF* и др.;

– виртуальные файловые системы: *AEFS* и др.;

– сетевые файловые системы: *NFS*, *CIFS*, *SSHFS*, *GmailFS* и др.;

– для флэш-памяти: *YAFFS*, *ExtremeFFS*, *exFAT*;

– специализированные файловые системы: *ZFS* (собственно файловой системой является только часть *ZFS*), *VMFS* (т.н. кластерная файловая система, которая предназначена для хранения других файловых систем) и др.

Контрольные вопросы

1 Какие виды структур файловых систем вы знаете?
2 Назовите функции файловых систем.
3 Какие файловые системы вы знаете?

Лекция №14. Иерархическая структура. Монтирование в Linux

Цель лекции: ознакомить студента с операциями в логической организации файловой системы.

Содержание лекции: операции над файлами; права в файловой системе.

Иерархическая файловая система. Все файлы файловой системы построены в структуру, которая называется деревом. В корне дерева находится, так называемый, корень файловой системы. Если узел дерева является листом, то это файл, который может содержать данные пользователя, либо являться файлом-каталогом.

Рассмотрим в качестве примера основные файловые операции *Unix*:

- создание файла, не содержащего данных. Смысл данного вызова - объявить, что файл существует, и присвоить ему ряд атрибутов. При этом выделяется место для файла на диске и вносится запись в каталог;

- удаление файла и освобождение занимаемого им дискового пространства;

- открытие файла. Перед использованием файла процесс должен его открыть. Цель данного системного вызова - разрешить системе проанализировать атрибуты файла и проверить права доступа к нему, а также считать в оперативную память список адресов блоков файла для быстрого доступа к его данным. Открытие файла является процедурой создания дескриптора или управляющего блока файла. Дескриптор (описатель) файла хранит всю информацию о нем. Иногда, в соответствии с парадигмой, принятой в языках программирования, под дескриптором понимается альтернативное имя файла или указатель на описание файла в таблице открытых файлов, используемый при последующей работе с файлом. Например, на языке Си операция открытия файла *fd = open (pathname, flags, modes*); возвращает дескриптор *fd*, который может быть задействован при выполнении операций чтения (*read*(*fd,buffer,count*);) или записи;

- закрытие файла. Если работа с файлом завершена, его атрибуты и адреса блоков на диске больше не нужны. В этом случае файл нужно закрыть, чтобы освободить место во внутренних таблицах файловой системы;

- позиционирование. Дает возможность специфицировать место внутри файла, откуда будет производиться считывание (или запись) данных, то есть задать текущую позицию;

- чтение данных из файла. Обычно это делается с текущей позиции. Пользователь должен задать объем считываемых данных и предоставить для них буфер в оперативной памяти;

- запись данных в файл с текущей позиции. Если текущая позиция находится в конце файла, его размер увеличивается, в противном случае запись осуществляется на место имеющихся данных, которые, таким образом, теряются.

- есть и другие операции, например, переименование файла, получение атрибутов файла и т.д.

В ОС принято разбивать диски на логические диски (это низкоуровневая операция), иногда называемые разделами (*partitions*). Бывает, что, наоборот, объединяют несколько физических дисков в один логический диск (например, это можно сделать в ОС *Windows NT*).

В ОС *Linux* предполагается наличие нескольких архивов файлов, каждый на своем разделе, один из которых считается корневым. После запуска системы можно смонтировать корневую файловую систему и ряд изолированных файловых систем в одну общую файловую систему. Это осуществляется с помощью создания в корневой файловой системе специальных пустых каталогов. Специальный системный вызов *mount* ОС *Unix* позволяет подключить к одному из этих пустых каталогов корневой каталог указанного архива файлов. После монтирования общей файловой системы именование файлов производится так же, как если бы она с самого начала была централизованной. Задачей ОС является беспрепятственный проход точки монтирования при получении доступа к файлу по цепочке имен.

Если учесть, что обычно монтирование файловой системы производится при загрузке системы, пользователи ОС *Unix* обычно и не задумываются о происхождении общей файловой системы.

В основе механизмов разграничения доступа лежат имена пользователей и имена групп пользователей. Создают классификации пользователей, например, в ОС *Linux* все пользователи разделены на три группы:

– владелец (*Owner*);

– группа (*Group*). Набор пользователей, разделяющих файл и нуждающихся в типовом способе доступа к нему;

– остальные (*Univers*).

В индексном дескрипторе каждого файла записаны имя так называемого владельца файла и группы, которая имеет права на этот файл. При создании файла его владельцем объявляется тот пользователь, от чьего имени запущен процесс, создающий файл. Группа тоже назначается при создании файла - по идентификатору группы процесса, создающего файл. Владельца и группу файла можно поменять в ходе дальнейшей работы с помощью команд chown и chgrp.

Это позволяет реализовать конденсированную версию списка прав доступа. В рамках такой ограниченной классификации задаются только три поля (по одному для каждой группы) для каждой контролируемой операции.

Контрольные вопросы

1 Какие операции над файлами вы можете назвать?
2 Что такое логический диск?
3 Какие три группы пользователей файлов вы знаете?

Лекция №15. Современные аспекты развития Linux

Цель лекции: ознакомить студента с азами обеспечения безопасности в операционной системе, перспективы развития операционных систем и *Linux*.

Содержание лекции: базовые принципы безопасности; возможные решения безопасности.

Базовые принципы безопасности *Linux*:

– минимальный уровень привилегий на доступ к данным;

– комплексный подход к обеспечению безопасности;

– баланс надежности защиты всех уровней;

– использование средств, обеспечивающих максимальную защиту при атаке (например, полная блокировка автоматического пропускного пункта при его отказе, полная блокировка входа в сеть и др.);

– единый контрольно-пропускной путь - весь трафик через один узел сети (*firewall*);

– баланс возможного ущерба от угрозы и затрат на ее предотвращение;

– ограничение служб, методов доступа для лиц, имеющих доступ в Интернет и из Интернета во внутреннюю сеть предприятия. Политика доступа к службам Интернет и политика доступа к ресурсам внутренней сети.

ОС с момента своего появления являлись ядром информационной технологии и оказывали влияние на всю ИТ-индустрию. Каждое поколение ОС формирует программную платформу для разработки приложений и в итоге создает новые сценарии использования ИТ. Понимание текущих тенденций в развитии операционных систем дает видение путей развития информационных систем в целом. Мобильные приложения на сегодняшний день не покрывают полный спектр потребностей пользователя персонального компьютера, их применение сосредоточено в области потребления контента, развлечений и вспомогательных программ.

Особняком стоит *Microsoft*, которая предлагает парадигму универсальной ОС на базе *Windows* 10, решающей полный спектр задач. Эта ОС обладает интерфейсом, характерным для мобильных устройств с сенсорным экраном.

Еще один вариант развития программных платформ и ОС предлагает *Google* с *Chrome OS*, состоящей из браузера. Все приложения, которые можно использовать это веб-приложения (сайты). При этом все данные пользователя хранятся в облаке, никаких уникальных данных на компьютере нет. Основные эффекты, достигаемые такой архитектурой: возможность использованию любого устройства без настройки и переноса личных данных, быстрая загрузка и низкая стоимость аппаратного обеспечения.

Свое видение программной платформы будущего предлагает компания *Canonical* (разработчик дистрибутива *Ubuntu Linux*). Проект называется *Ubuntu for Android*. Идея этой системы составляет запуск стандартного немодифицированной ОС *Ubuntu Linux* на *Android*-устройствах. Этим решается проблема синхронизации множества устройств между собой (данные, настройки, приложения). Для удобства работы на стационарном рабочем месте предлагается использовать полноразмерную периферию (монитор, клавиатура, мышь). Основанием для такой модели служит вычислительная мощность современных мобильных устройств, сравнимая с ПК 10-летней давности.

Основные направления эволюции ОС - в улучшении эргономики интерфейса и увеличении эффективности использования аппаратных ресурсов. Расширение информационного потока между пользователем и системой росло с каждым годом. Количество информации и скорость обмена информацией увеличивалась с применением новых технологий. Интерфейс и дальше будет меняться, вслед за открывающимися возможностями физики и электроники. Смысл всех изменений интерфейса - обеспечить максимально широкий и надежный канал связи между мозгом пользователя и ядром ОС. Полная интеграция электронных гаджетов с мозгом – дело далекого будущего и есть причины для того, чтобы этого не произошло никогда. Причина не в

технических или биологических ограничениях. Актуальные тенденции развития ОС и платформ:

– развитие ОС для устройств, отличных от ПК. Основные направления: мобильные устройства и бытовые приборы (плееры, телевизоры);

– изменение основного канала распространения ПО на подконтрольный разработчику ОС каталог ПО;

– развитие модели монетизации ПО в сторону бесплатных приложений с рекламой и продажи расширенной функциональности;

– создание кросплатформенных веб-приложений также актуально для достижения широкого круга пользователей с минимальными затратами;

– интеграция между ОС, каталогом приложений и магазином контента (книги, музыка, видео) - централизация каналов дистрибуции большинства цифровых товаров;

– усиление проблемы синхронизации различных устройств с ростом их количества. Растет потребность в универсальном личном устройстве переработки информации;

– использование веб-приложений как эталона при разработке ПО;

– количество экземпляров мобильных ОС значительно превышает настольные системы;

– ОС трансформировались в конечный продукт, определяющий не только программную платформу, но и канал дистрибуции ПО и контента.

Контрольные вопросы

1 Каковы дальнейшие перспективы развития эволюции ОС?
2 Каковы тенденции появления новых ОС?
3 Какие тенденции последнего пятилетия стали популярными?

Список литературы

Основная

1 Таненбаум Э. Современные операционные системы. – СПб.: Питер, 2013. – 1120 с.

2 Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы: Уч-к для Вузов. - СПб.: Питер, 2013. - 943 с.

3 Дейтел Харвин М. Операционные системы. Основы и принципы. Т. 1. 3-е изд.- М.: БИНОМ, 2011. - 1024 с.

4 Эви Немет, Гарт Снайдер и др. UNIX и Linux. Руководство системного администратора. - Киев: Вильямс, 2012. – 1312 с.

Дополнительная

5 Национальный открытый университет ИНТУИТ. Основы операционных систем http://www.intuit.ru/studies/courses/1088/322/info

6 Национальный открытый университет ИНТУИТ. Операционные среды, системы и оболочки http://www.intuit.ru/studies/courses/492/348/info

7 Национальный открытый университет ИНТУИТ. Основы работы в ОС Linux http://www.intuit.ru/studies/courses/91/91/info

Дополнительный план 2017 г., поз. 12

Зуева Екатерина Александровна

ОПЕРАЦИОННАЯ СИСТЕМА LINUX

Конспект лекций
для студентов специальности
5В100200 – Системы информационной безопасности

Редактор Л.Т. Сластихина
Специалист по стандартизации Н.К. Молдабекова

Подписано в печать __ __ _____ Формат 60х84 1/16.
Тираж 30 экз. Бумага типографская №1
Объем 2,9 уч.-изд.л. Заказ № __ Цена 1450 тенге

Копировально-множительное бюро
некоммерческого акционерного общества
«Алматинский университет энергетики и связи»
050013, Алматы, ул. Байтурсынова, 126