Лекция 2

Последовательность выполнения команд процессором определяется логикой программы и результатами выполнения предыдущих команд. Существуют ситуации когда процессор должен приостановить выполнение текущей программы для того чтобы срочно отреагировать наиболее срочный запрос. Для этого существует прерывание.

Прерывание (аппаратное) – это сигналы при поступлении которых нормальная последовательность выполнения программ может быть прервана. При этом система запоминает информацию необходимую для возобновления работы прерванной программы и передает управление подпрограмме обработки прерываний. После завершения обработки управление возвращается прерванной программе.

Прерывание делится на 3 основных типа:

1. Прерывание аппаратное и периферийных устройств
2. Внутреннее (исключение)
3. Программное

В современных ОС обработку всех типов прерываний берет на себя сама система и это влияет на все системы. Типы прерываний разнообразны каждый из них требует особой обработки поэтому большинство процессоров поддерживают векторное прерывание. То есть каждая разновидность прерывания имеет свой номер. Этот номер используется как индекс в массиве который хранит для всех прерываний так называемые векторы прерываний. Каждый вектор содержит адрес соответствующий адрес команд обработки прерываний и доп информацию(дескриптор прерываний). При возникновении прерывания аппаратура по номеру прерывания по номеру определяет адрес подпрограммы обработки и вызывает ее предварительно сохраняется информация которая нужно для последующего возврата к выполнению прерванной программы. Чтобы наиболее ответственный участки программ выполнялись без прерываний система может временной запретить прием большинства прерываний. Этот запрет устанавливается автоматически при начале обработки полученного прерывания. Запрет прерывания должен сохраняться лишь в течении короткого интервалов времени не дольше нескольким мс иначе следубщие прерывания могут остаться необработанными и приводит к ошибкам системы.

Прерывания от периферийных устройств. Этот тип прерываний используется для управления операциями ввода вывода. Устройства генерируют сигнал прерывания в 1 из двух случаев: при переходе в состояние готовности, при возникновении ошибки выполнения операции.

Состояние готовности – это такое состояние устройства, в котором оно готово принять и выполнить команды от процессора. Для устройства ввода готовность означает наличие в устройстве данных который могут быть переданы в процессор. Для устройства вывода готовность это возможность принять процессором данные которые нужно вывести. Для служат сигнал прерывания.

Ошибка операции тоже требует вмешательства системы или пользователя (отсутствие бумаги в принтере, ошибка чтения диска…) Не каждое устройство генерирует прерывание, например монитор не выдает прерываний (может принять данные для отображения). Линии запроса прерывания от различных устройств поступают на входы схемы контроллера прерывания. Каждому устройству в контроллере присваивается номер запроса для вызова подпрограммы обработка. Контроллер формирует запрос на прерывание, он передается в ЦП. Запрос содержит номер вектора прерывания, который определяется на основе программы обработки прерываний. Контроллером может заблокировать передачу менее приоритетных прерываний. В периферийных устройствах жестко определяют номера контроллеров. Число подключаемых устройств ограничивается числом входов контроллера.

Исключения или внутри аппаратное прерывание это сигналы которые сообщают о внештатных ситуациях возникающих при выполнении команд (деление на 0). Обработка исключений не может быть запрещена. Результат обработки исключений либо аварийное завершение программы либо попытка продолжить выполнение но предварительно попытавшись устранить ошибку. Они вызываются выполнением специальной команды программы Но обрабатываются также как остальные типы прерываний. Остальные типы программного прерывания это особый случай вызова подпрограммы но при это вместо адреса подпрограммы указывается номер прерывания обработчик которого должен быть вызван. В большинстве операционных систем программные прерывания исспользуются для перехода из режима пользователя в режим ядра при вызове системных функций.

Синхронные и асинхронные прерывания:

исключения и программные прерывания объединяются общим понятием синхронное прерывание. Он могут возникнуть только при выполнении определенных команд. Они как бы синхронизированы с выполнением текущей программы. А вот прерывания периферийных устройств являются асинхронными, они могут произойти в произвольный момент вне зависимости от текущей программы.

Режимы работы процессора:

Как минимум 2 режима работы:

Привилегированный (режим ядра)

Непривилегированный (режим задач, пользователя)

Программы работающие в привилегированном режиме имеют неограниченный ко всем ресурсам пк. В непривилегированном режиме возможности программы ограничены, не может выполнять запрет разрешения и запрета прерывания и тд. Аппаратное разграничение возможностей это необходимое условие для надежной защиты данных в многопользовательской системе.

Программа пользователя работает в непривилегированном режиме. Не могут произвольно переключаться в режим ядра и выполняться в нем. Переход из режима пользователя в режим ядра только в случае обработки прерывания аппаратного или команды программы. Выбранный номер прерывания и другие параметры определяют какая из системных функций будет выполнена, после выполнения необходимых действий в режиме ядра система возвращает управление программе пользователя переключая процессор в непривилегированный режим. Важная возможность обеспечиваемая различением режимов работы процессора это ограничение доступа пользовательских процессов к областям памяти. Каждый процесс получает в распоряжение собственное адресное пространство, отображаемое аппаратурой на определенную область памяти вычислительной системы. Процесс не видит других процессов которые присутствуют в системе если отображение их адресных пространств на память не пересекаются.

Процессоры intel поддерживают 4 режима работы процессора которые называются кольцами защиты, они отличаются уровнем предоставляемых программам привилегий от максимальных (кольцо 0) до минимальных (кольцо 3). Некоторые операционные (real time system) используют кольцо 0 для ядра системы, кольца 1 и 2 для других системных программ и кольцо 3 для пользователей. Достигается таким образом повышение надежности систем. Большинство операционных систем (винда) не используют кольца 1 и 2.

ВАРИАНТЫ АРХИТЕКТУРЫ ЯДРА:

В практике разработки операционных систем конкурируют разные подходы к определению объема функции которые выполняются ядром. А так же внутренние структуры ядра.

1. Монолитное ядро – это традиционный тип архитектуры ядра. Представляет собой единую и неделимую программу выполняющую все основные функции операционной системы. Отдельные подсистемы составляющие ядро работают в общем адресном пространстве и поэтому могут использовать общие структуры данных и передавать управление друг другу. Объединением функций в монолитные программные структуры приводит к разделению функций системы на отдельные группы достаточно условно.

КРАТКИЙ СПИСОК ДЕЙСТВИЙ, КТОТОРЫЕ ВЫПОЛНЯЮТСЯ СИСТЕМОЙ ПРИ ЗАПУСКЕ ПРОГРАММЫ ПОЛЬЗОВАТЕЛЯ ПРИ ВЫПОЛНЕНИИ:

Лекция 3

КРАТКИЙ СПИСОК ДЕЙСТВИЙ, КТОТОРЫЕ ВЫПОЛНЯЮТСЯ СИСТЕМОЙ ПРИ ЗАПУСКЕ ПРОГРАММЫ ПОЛЬЗОВАТЕЛЯ ПРИ ВЫПОЛНЕНИИ:  
1. Ввод и интерпретация

1. Создание нового процесса
2. Выделение памяти для процесса
3. Поиск и открытие для чтения исполняемого файла программы
4. Обращение к дисковому устройству для чтения кода программы
5. Фиксация всех предпринятых действий в системном журнале
6. Запуск загруженных программ в процессор на выполнение

Все основные функции операционной систем:

Управление процессами, управление памятью, периферийными устройствами, ввод и интерпретация команды, сбор статистики.

Эффективность выполнения системных функций тем выше чем легче переход от функций одной группы к другой. Чем теснее руппы связанны между собой, поэтому монолитное ядро выглядит наилучшим решением.

Но монолитная архитектура имеет недостатки:

1. Проектирование, отладка, сопровождение большой по объему и сложности программы будет затруднено.
2. Сложность отладки большой программы примерно пропорциональна квадрату ее размера, с учетом того что монолит может достигать нескольких миллионов команд
3. Надежность монолитного ядра не высокая. Считается что любая сложная программа даже отлаженная содержит некоторое число не выявленных ошибок. Примерно 1 ошибка на 1000 команд и чем теснее связаны друг с другом отдельные части ядра тем необратимее последствия ошибки. Особенно опасны ошибки это драйверы периферийных устройств которые при загрузки становятся частью ядра. Ошибка в работе драйвера может привести к краху всей системы.

МОДУЛЬНОЕ И МНОГОУРОВНЕВОЕ ЯДРО:

Проблема разработки монолитного ядра смягчается использованием принципа модульности. Разбиение программы на модули которая минимально зависит друг от друга а так же четкое формализованное описание связей сохраняющееся между модулями это условие успешной работы и успешной разработки сложных программных систем, в том числе и ядер ОС. Изменение в одном из модулей системы не должны приводить к существенным переработкам других модулей. При проектировании ядер ОС как многоуровневая структура ядра.

Верхний уровень – библиотека системных вызовов и их выполнение требует обращения к нижележащим уровням и так вплоть до самого нижнего уровня который и взаимодействует с аппоратурой.

Модульная и многоуровневая структура ядра облегчает разработку больших систем но ценой незначительной снижения производительности. Понятная и четкая структура ядра положительно сказывается на надежности. Уменьшается кол-во трудно обнаруживаемых ошибок. Не снимается проблема влияния ошибок в любом модуле ядра на работу системы в целом.

МИКРОЯДРО:

Альтернатива монолиту. В ОС на микроядре большая часть функциональных возможностей системы выносится за пределы ядра и передается вспомогательным системным процессам которые выполняют свои задачи и работают в пользовательском режиме. Такие процессы называют сервисами. К ним относятся драйверы периферийных устройств а так же модули выполняющие управлением процессами, распределением памяти и управлением файлами. Сбой в работе одного из сервисов не приводит к краху всей системы в целом. Система может перезапустить драйвер и восстановить работоспособность. Любое программное или аппаратное прерывание приводит к вызову микроядра, обычно микроядро анализирует причину ядра вызывает для его обработки соответствующий сервис. И самому микроядру остаются функции которые могут быть выполнены только в режиме ядра:

1. Запуск и завершение процессов
2. Переключение работающих процессов
3. Управление отображением адресных пространств процессов
4. Управление привилегиями сервисных процессов. (разрешения драйверам
5. Поддержка обмена сообщениями между процессами.

Операционные системы построенные на основе микроядра имеют многоуровневую модульную структуру, системы более надежные и гибкие чем системы с монолитным ядром.

Узкое место микро ядерных систем это производительность, работа ОС требует тесного взаимодействия различных подсистем и если модули монолитного ядра передают друг другу управление с помощью обычных команд перехода или вызова подпрограмм то в системе с микро ядром отдельные сервисы взаимодействует друг с другом только посредством микроядра. Всякий раз нужно вызывать программные прерывания для переключения. Обрабатывая прерывание микро ядро посылает сообщение сервису, адресату который выполнив запрос должен вызвать микроядро для доставки результатов сервису отправителю. За счёт приемов программирования достигают высокой производительности, она уступает монолиту на 15-20%, для большей части распространенных ОС характерно либо модульно-монолитное ядро(Линуха) либо гибридное ядро(это компромисс между монолитным и микро ядерным подходом) (Windows NT и дальше).

Системные функции:

В качестве программного интерфейса системы то есть средств обращение ПП к службам ОС используется документированный набор системных вызовов или функций API.

Системный вызов - это функции реализуемые непосредственно программами ядра системы. При микро ядерной архитектуры с участием ядра. При их выполнении происходит смена режима на режим ядра и обратно.

Api функции – это функция описанная в документации ОС независимо от того выполняются они ядром или системными библиотеками работающими в режиме пользователя. В windows несколько api обращаются к одному и тому же недокументированному системному вызову но имеют различные обрамляющие части работающими в режиме пользователя.

И там где различия несущественно между понятиями используют нейтральный термин системная функция.

История ОС:

1. Ms-DOS – Однозадачная система
2. Windows – сложная система на базе ms-dos
3. Unix – система по возможности сопоставимая с windows но отличающиеся по основным концепций и методом реализации

MS DOS разработанв 1981 для первого 16 разрядного ПК. Однозадачная система, ведет диалог с пользователем в текстовом режиме рассчитана на обслуживание прикладных программ в текстовом режиме но допускает экран. Работа с мышью обеспечивается самими программами, при нимиальной поддержке ПО. Для размещения ПО пользователя мс дос может 640 КБ памяти, позднее были добавлены средства позволяющие использовать до 4МБ памяти. Интерфейс ms dos с прикладными программами основан на вызовах программных прерываний которые обрабатываются системой, большую часть этих прерываний называют функциями dos. Это стартовая площадка для винды. Все версии Виндовс, обеспечивают выполнение большей части программ на дос.

Винда:

Первоначально она была разработана как графическая оболочка запускаемая под ms dos. Версии windows 1.0 (1985 год) работала на 1 МБ памяти с неперекрывающимися окнами. Версии 3.0 и 3.1 (90,91 год) Использовали 16 МБ памяти. Эти версии оставались надстройками и добавляли собственное управлениями процессами. Комбинация Dos + Windows это многозадачная система с графическим интерфейсом.

1993 Появляется NT это уже полноценная многозадачная и многопользовательская ОС без Ms Dos. Система требовала крутые процессора

1995 Появилась Windows95 и повышение скорости работы путем отказа от многопользовательской защиты и ослаблением надежности системы. (Неаккуратно написанные ПП могла привести к краху, но правильная программа переносились на WindowsXP). Винда дает несколько тысяч документированных API функций и распространение ОС способствует простота установки и обилие приложений.

Unix и PostX. В 1973 году система переписана на Си и оставил минимум текста на ссылка. Первая десятилетие система распространялась свободно, поэтому популярность в науке и у студентов. Пробелма, не полные совместимые разные версии.

В 1983 году Unix v5. Тогда решили сделать стандарт. 1988-1990 был разработан набор стандартов ктороые названы Posigs, которые развиваются. Эти стандарты касаются поведения системы а не ее устройства. Они определяют требования к набору системных команд и функций.

В 1990-ие возврат коммерции. Появляется linux, Отличие от unix состоит в большей надежности и эффективности. Исходные тексты свободно распространяются. Таком образом Unix многозадачная многопользовательская система разделения времени.

Достоинства:

Мобильность, крутой интерфейс, хорошее архитектурное решение

Недостатки:

Более низкая эффективность и надежность работы(плата за мобильность системы). Настроить сложнее, linux система для программистов.

Архитектура Linux предназначена для системы разделения времени с одним процессором, стала подходящей для поддержки сетевых систем.

У винды главные окна а у unix главное текст (окна дополнительное, не обязательное). Особенность linux: развитый язык, shell (он нужен для диалога системы, скрипты без обычных языков). Системы Linux лидируют для серверов и суперпк, мобилок. Винда удобна дома

# Лекция 4

Периферийные устройства,

Архитектура устройства – это совокупность особенностей которые влияют на работу программ с устройствами. Кол-во и назначений линий входят в архитектуру если линии могут программо управляться. С устройством поставляется его контроллер или адаптер. Он содержит электронную схему управления устройством, которое вставляется в разъем или расположенно в корпуссе.

Программы работают с устройством последством контроллера. Следовательно нет различия с точки зрения архитектуры устройства и контроллера устройства.

Периферийные устройства объединяют все основные аппаратурные блоки кроме процессора и основной памяти. Для повышения надежности используются разные программно-аппаратные методы. Это дублирование данных, использование защищенных кодов, использование контрольных сумм данных, процедуры выполнения операций с устройствами должны выявлять аппаратные ошибки и сбои а также компенсировать за счет избыточности данных и повторного выполнения операций.

Эффективность использования устройств заключается в сокращении времени на обмен данными. Помимо повышения скорости обменивания, сокращение времени достигается за свет распараллеливания работы Пер. Уст. И процессов. Кроме того сокращения кол-во операций ввода-вывода происходит за счет сохранения данных в памяти и последующего использования. Доп задача которые решают систему управления устройством это хранение данных в сжатом виде, шифрование.

Устройства последовательного и произвольного доступа:

Символьные и блочные устройства.

Для символьных или байтовых устройств наименьшая порция данных есть 1 байт. Для некоторых символьных усьтройств можно за 1операцию произвести ввод-вывод любого кол-во данных.

Для блочных устройств вывод или ввод за 1 обращения к устройству это 1 блок (2^k байтов).

Типичный размер блока 512байтов, 1кб, 4кб.

Примеры блочных устройств: магнитные ленты, магнитные диски.

Для диска понятие блока совпадает с понятием сектор, для ipm совместимых пк. Сектор равен 512байтам.

Блочный ввод-вывод эффективен для высокоскоростных устройств потому что уменьшается время которрое тратится на подготовку и заключительные операции при каждом обращении.

Физические, логические, виртуальные устройства.

Физическое – реально существующий прибор, железо. Для наличия надо знать наболр адресов команд прерывания и других сигналов для операции сдавливания.

Логическое устройство – оно характеризует специальное устройство в данной ОС, загрузочный диск. Устройство стандартного ввода это важные логические устройства ОС, исопльзуется для ввода и вывода по умолчанию, то есть когда в программе явно не указывется другое устройство или файл для ввода вывода. Ему соответствует физ устройство клавиатура, для стандартного вывода – монитор.

Виртуальные устройства – программо реализованный объект ведущий себя как устройства, хотя использует ресурсы других устройств, Виртуальные диски.

Виртуальная память расположена на диске. Виртуальный экран, который предоставляется программе работающей в окне винды. Пустое устройство которому не соответствует никакая аппаратура. Тк драйвер пустого устройства корректо работает, принимает выходной поток символов и выкидывает его. Может использоваться для ввода. Оно нужно когда программа требут указать файл или устройство для вывода объемных но не очень нужных данных. Проверка чтения файла.

Архитектура подсистемы ввода-вывода

Устройства и его контроллер програмно предствалены одним или несколькими регистрами.

Регистр устройства – это адресуемое машинное слово для обмена данными и сигналами между устройствами процесса.

Выделяют два типа регистров:

Регистр данных. Запись в регистр – вывод данных, Чтение из регистра это ввод.

Регистр управления и состояния содержит два типа двоичных разрядов, это биты состояния, они служат для передачи процессу информации о текущем состоянии устройства, флаги готовности и ошибки, сигнал прерывания и биты управления. Они служат для передачи на устройств команд задающих выполняемую операцию для запуска выполнения операций, для установки режимов работы устройства. Используются в различных пк, один из двух способов адресации регистров устрйойств:  
Отображения регистров на память, Для устройств отводится определенная часть адресного пространства памяти а для работы с устройствами используются теже команды что и для работы с основной памятью.

Адресация регистров через порты ввода-вывода, для портов отводится отдельно адресное пространство и для работы с ним существуют специальные команды.

Сравнение способов, 1 удобнее для программирования тк дает возможность использовать более широкий набор команд но его труднее реализовать на аппаратном уровне тк аппаратура должна определять относится ли конкретный адрес к памяти устройства и по разному обрабатывать. Среди конфигурации однопроцессорные вычислительные системы выделяют два основных типа:

1.Система с магистральной архитектура

Основана на подключении всех устройств вместе с процессором и памятью к единой шине. Она объединяет линии передачи данных, устройств, сигналов…

Совместное использование магистрали различными устройствами подчиняется специальным паовилам или протоколам.

2. С радиальной

Она предполагает что каждое из устройств вместе памятью подключается к процессору отдельно не зависимо от других устройств и взаимодействуют с процессором по собственным правилам.

Для программиста с точки зрения архитектуры похуй. Ему важно какие сигналы можно посылать и принимать которая работает.

Основная особенность магистральной архитектуры – это единообразный спосбо подключения всех устройств.

Структура регистров устройств стандартизируется, определяется его разряд. Некоторые типы устройств могут не генерировать прерывания, не сообщать об ошибках но сигналы которые устройства используют должны соответствовать стандарту данной магистрали.

Преимещества магистрального типа:

Простота добавления устройств, удобна для открытых вычислительных систем.

Для радиальной архитектуры, индивидуальный выбор способов подключения, удобный для устройства. Расширение рад системы сложное. Удобно когда кол-во устройств постоянные.

Исходня модель имела радиальную архитектуру и небольшой набор стандартных устройств, в последующих моделях появляется шина, однако они не соответсвуют магистральной архитектуре. Ведь есть много.   
Важный элемнет архитектуры ПК э то контроллер прямого доступа к памяти. Он используется для обмена данными между основными устройствами и память. Если весь обмен данными идет через регистры процессора то ПДП используется. Роль процессора инициировать операцию ввода вывода данных послав команду котроллеру ПДП и далее процессор не учавствует в обмене данных. Завершив операцию контроллер ПДП посылает сигнал прерывания оповещая процессор. Это повышает производительность системы за счет частичной разгрузки процессора и магистрали. Обмен данными с дисковыми устройствами в режиме пдп ускоряется в несколько разв сравнении с режимом процессороного ввода-вывода. При режиме процессорного ввода-вывода прочитанные при вводе данныые сначала одной командой пересылается из регистра устройства в регистр процессора а затем другой командой пересылается в память.

Активное и пассивное ожидание. Различие между способами ввода-вывода по опросу готовности и прерывания. Особенность ввода-вывода по опросу готовности состоит в том что это цикл ожидания. Если выполняется ввод или выводна медленное устройство то этот цикл будет занимать 99% работы процессора. Если ожидание ввода с клавиатуры то процессор не делает ничего пока пользователь не нажимает клавишу, это оправдано если нет полезной работы процессора и возможно однозадачной операционной системы, когда работающая прикладная программа не может продвигаться дальше пока не завершенна операция. В этом случае ввод выод по опросу имеет преимущества, он не связан с обработкой прерываний которые требуют времени и которые замедляют реакцию на переход устройства в состояние готовности.

Способ ожидания программы основанный на постоянной циклической проверки ожидания устройства. Называется активным ожиданием. Так работает не только ввод-вывод, но и в других ситуациях, которые возникают в работе системных и прикладных программ. Если многозадачная ОС в которой несколько активных задач одновременно. То активное ожидание на уровне прикладных программ не приемлемо. Нельзя занимать время других прогамм. Поэтому лучше пассивное ожидание. Это реализация ожидания при которой ожидающая программа не затрагивает процессорного времени. Для пассивного ожидания используются аппаратные прерывания. Иногда это оправдано. Расход времени на пассивное ожидание оказывается значительно больше чем 2 мс и в таких случаях используют активное ожидание.

Синхронный и асинхронный ввод-вывод.

# Лекция 5

Синхронный и асинхронный ввод-вывод.

Система скрывает от приложения детали низкоуровневой работы с устройством. Различия между вводом-выводом по прерываниям отражается на уровне системных функций. В виде функции для синхронного и асинхронного ввода-вывода. Выполнение функции синхронного ввода вывод включает запуск операции ввода-вывода и завершения этих операций. После завершения ввода-вывода функция возвращает управление вызвавшей программы.

Синхронный ввод-вывод, более привычный для программистов способ работы с устройствами. Стандартные процедуры ввода-вывода в ЯП работают именно так.

Вызов функции асинхронного ввода-вывода предполагает только запуск соответствующей операции. После этого функция сразу возвращает управление вызвавшей программе, не дожидаясь завершения операции.

Асинхронный ввод данных – программа не может обращаться к данным пока нет уверенности что их ввод завершен. Программа может заняться другой работой, чтобы не простаивать. Программа приступит к работе с введенными данными после того как асинхронная операция будет завершена. Для этого операционная система предоставляет средства которые делятся на 3 типа:

1. Ожидание и завершение операции

Вторая половина синхронной операции. Программа запустила операцию потом выполнила посторонние действия и ждет окончания операции как при синхронном вводе выводе.

1. Проверка завершения операции. Программа не ожидает а только проверяет состояние асинхронной операции. Если ввод-вывод не завершен, то программа имеет возможность еще что-то делать.
2. Назначение процедуры завершения. Запуская асинхронную операцию, программа пользователя указывает система адрес процедуры которая должна быть вызвана системой после завершения операции. Программа не интересуется состоянием вводом-выводом. Система сама напоминает ей об этом в нужный момент вызвав указанную функцию. Самый гибкий способ, тк в процедуре завершения пользователь сам решает что делать.

В винде есть все 3 способа. В unix асинхронный функций ввода-вывода нет, но эффект асинхронности достигается за счет доп процесса.

Асинхронное выполнение ввода вывода в некоторых случаях повышает производительность работы и обеспечивает доп функциональную возможность. Без простой формы асинхронного ввода как ввод с клавиатуры без ожидания невозможны пк тренажеры. Однако логика программ использующей асинхронные операции сложнее чем при синхронных операциях.

Драйвер устройств – это системная программа которая под управлением ОС выполняет все операции с конкретных периферийным устройством. Перед драйвером стоят 2 задачи:

1. Обеспечить возможность стандартного обращения к любому устройству скрывая от остальных частей ОС специфические особенности других устройств.
2. Добиться максимально эффективного использования всех функциональных возможностей и особенностей конкретных устройств.

Возможность стандартными средствами работать с разными устройствами желательна с точки зрения архитектуры ОС и удобства разработки. В большинстве ОС различаются 2 разных типа драйвера: для символьных и блочных устройств. Обращаясь к драйверу ОС указывает функцию которую надо выполнить, список этих функций общий для драйверов различных устройств. При этом каждый драйвер может реализовать только те функции которые имеют смысл для данного устройства. Наиболее общие функции: Чтение данных, запись данных, инициализация утройства(вызывается системой при подключении-запуска), открытие и закрытие устройства.

Для блочных устройств имеют смысл такие функции как: форматирование и поиск сектора. Для символьных устройств ввода; функции не разрушающие ввода то есть проверки очередного символа без его изъятия из входного потока. Для того чтобы учесть все разнообразие возможных операций в число функций драйвера вводят такую как выполнение специальных функций. И уже для каждого устройства определен свой набор этих спец функций.

Примерная структура драйвера:

Типичный драйвер устройства содержит как минимум 3 блока:

1. Заголовок драйвера
2. Блок стратегии
3. Блок прерываний

Заголовок содержит различную информацию о данном драйвере о об управляемом устройстве: Имя устройства, тип устройства, число однотипных устройств, обслуживаемых одним драйвером, объем памяти…

В заголовке лежат также адреса блок стратегий и прерываний.

В обязанности блока стратегий входят: прием запроса на выполнение операции, ведение очереди запросов (многозадачная система), а также при асинхронных операциях выполнения могут дожиться несколько операций, запуск операций и ее завершение.

Блок прерывания – он выполняет алгоритм называнием вводом-выводом по прерыванию. Система вызывает этот блок, когда получает сигнал прерывания от устройства обслуживаемого драйвером. Закончив выполнения запроса блок прерываний возвращает управление к блоку стратегий для завершения операции.

Кроме этих 3-х основных блоков в драйверах могут быть: блок инициализации (используется при загрузке ОС, а затем может быть выгружен из памяти), блок изменения параметров драйвера и др. Это низкоуровнение драйвера(да сука)

Высокоуровневые драйвера – усложнения периферийных устройств сделала популярных многоуровневую схему использования драйвера. Высокоуровневые драйвера лежат между драйверами аппаратуры и другой часть ОС. В них нету блока прерываний. Он принимает заявки от системы, преобразует данные, а затем вызывает низкоуровневый драйвер для работы с устройством.

Шина usb, драйвер usb:

Традиционный подход к подключению периферийных устройств предполагает что каждое устройство подключается к определенному разъему. По этому при разработке драйвера необходимо знать и учитывать назначение каждой линии разъема. Шина usb требует другой подход, в частности в разработке. С точки зрения архитектуры системы шина usb состоит из главного контроллера, то есть host контроллера к которому могут подсоединяться 127 устройств.

Весь обмен данными по шине инициируется запросами от контроллера на которые отвечает периферийные устройства. При подключении устройства хост контроллер шлет запрос в ответ на который устройства должно доложить о своем классе и ряде доп атрибутов. Есть большое число предопределенных классов устройств среди которых есть: устройство памяти, интерфейст с пользователем, звуковая карта,веб камера, принтер… В зависимости от прочитанного класса устройства host-контроллер определяет какие методы передачи данных будут использоваться.

Таких методов всего 4:

1. Управление управляющей передачи для управления устройствами и считывании информации о состоянии.
2. Передачи по прерыванию. Для ввода-вывода небольших порций данных. До 64 байт.
3. Передача массивов данных, при чтение записи с флешки.
4. Изохронные передачи для которых обеспечивается требуемая скорость в реальном времени, иногда ценой части данных. Для демонстрации видео, аудио.

Host-контролер опрашивает с интервалом 1 мс. И может вести обмен с несколькими устройствами при этом на изохронные передачи прерывания отводится до 90% интервала. На управляющие передачи резервируется 10%. То что осталось на передачу массивом данных. Устройства подключенные к шине юсб не посылают сигналы прерываний в ЦП. Вместо этого контроллеры устройств формируют пакеты данных для перидачи по прерываниям или выполняютдругие типы передач. Работа ОС с шиной юсб обеспечивают 3 типа драйвера:

1. Драйвер host-контроллер (опрос устройств и обмен данными).
2. Драйвер шины usb. Управление шиной, которые не зависят от конкретного контроллера. Нумерация и настройка, распределение времени обмена. Взаимодействие с другими программами ОС
3. Драйвер Устройств usb, формируют специфичные запросы на выполнения операции, передают их драйверам шины и ожидают завершения.

Символьные устройства.

# Лекция 6

Клавиатура – Ос может либо полностью взять на себя обработку сигналов посылаемых устройством. Либо воспользоваться услугами bios в состав которого входят программы.

Мышь и сенсорный экран.

Мышь посылает сигналы прерывания при перемещении в любом направлении при нажатии или отпускании кнопок, при повороте колесика. Обработка прерываний всегда лодится на драйвер мыши в соотвествующей ОС. Драйвер суммирует перемещение мыши вычисляет ее текущие координаты и может выдать по запросу по запросу ОС эти координаты и текущее кнопок. Учитывается что программы ожидающие пользовательского ввода отслеживают действия с мыши должны одновременно учитывать возможность нажатия клавиш на клавиатуре. В винде драйвер мыши генерирует сообщения о перемещения мыши и нажатии и отпускании кнопок. А ОС обеспечивает доставку этих сообщений соответсвующим окнам. В случае устройств которые распознают сразу несколько точек касания – сенсорные жкраны – драйвер сенсорного экрана по запросу ОС выдает коориднаты всех точек.

Управление устройствами винды. Драйверы устройств винды.

ТК винда многозадачная система она исключает для прикладных ПО прямое обращение к портам ввода-вывода или обработку аппаратных прерываний. Взаимодействие с аппаратурой нанизком уровне выполняется только системными программами работабщими в привелигированном режиме. Основную роль играют драйверы устройств. В винде используется многоуровневая система драйверов, в ней высокоуровневые драйверы играют роль фильтров, котоорые выполняют специальную обработку данных полученных драйверами низкого уровня или передаваемых такому драйверу. Например отделение драйвера управляющего шиной от р=дравйверов конкретных устройств подключенных к шине.

Структура драйверов все уровнейн подчинена единым стандартам, WDN. Но высокоурорвневые драйверы в отличие отт низкоуровневых незанимамются обработкой аппаратных прерываний.В видне хп низкоуровневые драйверы не самый низкий уровень управления устройствамию. Еще ближе к аппаратуре лежит уровень аппаратных абстракций. Его роль скрыть от остальных модулей ОС в тч от драйверов некоторые детали работы с аппаратурой зависящие от конкретных шин типа мат. Платы, способы подключения. Например уровень аппаратных абстракций предоставляет дравйверам возможность обращаться к регистрам устройств по их логическим номерам не зная подключен ли регистр к порут процессора или отображет на память.

Можно выделить несколько типов специальных драйверов которыетотличаются функциональными значениями.:

1. Драйверы GDI – представляют собой высокоуровневые графических устройств (мониторы, плоттеры, принтеры), они выполняют трансляцию графических вызово винды, провести линии … в команды выполняющие соответсвующие действия на конкретном устройстве. Выдача этих команд на устройство выполнятеся другим низкоуровневым драйвером. И благодаря драйвером этого типа одна и таже программа может выдавать графические изображения на разные утсройства.
2. Драйверы клавиатуры и мыши. Кроме стандартных для драйвера операций они выполняют доп работу – генерируют сообщения о событиях на устройстве и помещают ихв системную очередь сообщений. Затем системы переплавляет сообщение процессу клоторому оно предназначено для дальнейшей обработки.
3. Драйверы виртуализации устройства. Онги служат для того чтобы разделять устройства между процессами и создавать иллюзию того что процесс монопольно владеет устройством. На самом деле драйвер организует очередь заявок подпроцессов переключает устройства в нужный для очередного процесса режим. (пример, драйве вритуализации монитора)

Доступ к устройству

В большинстве случаев програмы не работают непостредственно с устройством. Для выполнения операций Используются API функций более высокого уровня,а обращения выполняется по мере необходимости. Программист может непосредственно работать с устройством.

Управление устройствами в Unix.

Изза существенного различия между символьногои блочными устройствами различают 2 основных типа дарйверов:

1. Символьный
2. Блочный

Для символьных устройств используют только символьные драйверы.

Для каждого блочного устройва два разных драйвера, блочный и символьный

Блочный драйвер позволяет выполнять только с целым числом блоков.

Символьный драйвер позволяет это более высокоуровневая программа которая имитирует выполнения операции чтения и записи произвольного кол-во байт и на самом деле используте обращение к блочному драйверу.

Кроме драйвера релаьных физических устройств система может включать жрайверы псевдоустройв.Драйвер обеспечивающий обращение программ к системной памяти. При загрузке формируются 2 таблицы Для символьных и блочный жрайверов. Строки таблицы соответсвуют конкретным драфйверам а столбцы соответсвуют функциям которые долженвыполнять драйвер. Так что в ячейках содержатся адреса по которым вызываются функции драйверов. Набор фукнций для символьных иблочных драйверов различаются поэтому используют 2 разные таблицы.

Наиолее важной фукнцией драйверов:

1. Открытие утсрйоства, при этом увеличивается счетчик текущих обращений к устройствам. Это позволяет ставить обращение в очередь если устройство занято. Некоторые устройства при открытии могут выполнять еще какие-то начальные дейтвия.
2. Закрытие устройства. Это операция противоположная открытию.
3. Обработка прерываний. Она выполняет ввод или вывод очередной порции данных когда устройство переходит в состояние готовности.
4. Опрос устройства. Функция выполняется для тех устрйоств котороые не генерируют прерывания или если при рпзработке драйвера по каой-либо причине решено не использовать прерывания от устройств. Опрос выполняется не постоянно, а с некоторым периодом по прерывания от таймера.
5. Чтение данных с утсройства.
6. Запись данных на устройво
7. Вызов стратегии. Это способ выполнения ввода-вывода характерный для блочных утсройств. При этом запрос может быть поставлен в очередь. Запрос моет быть удовлетварен с помощью обращения к диссковому кэшу без выполнения чтения ли изаписи на устройство.
8. Выполнение специальных функций. Набор этих функций зависит от конкретного устрйоства. Это может быть опрос или устсановка текущего режима работы устройства. Форматирование дорожек диска.

Устройства как специальный файл

Отличительная особенность unix заключается в том что для работы с периферийными устройствами прикладные программы должны использовать теже свойства то и для работы с файлами. Устройства unix представлены как специальные файлы вписанные в котолог файловой системы, на равне с обычными файлами. Каждому драйверу устройства соответсутет отдельный специальный файл символьный или блочный в зависимости от типа драйвера. Для того чтобы начтаь работать с устройством программам должна вызвать функцию открытия файла указав ей имя спец файла. Приэтом происходит обращение к функции открытия из драйвера соответсвующего устройства. С каждым специальным файлом связаны два числа которые называются Старшим и младшим номерами устройства. Старший номер определяет номер строки в таблице символьных или блочный драйверов. Млачший номер передается драйверу как доп параметр. Он может озгначть наприммер номер конкретного дисового устройства.

Дисковые устройства, диска, магнитные диски.

Основной класс периферийных устройств. Для долговременного хранени инфорации – различные виды дисковых накопителей. Диски – это основной и почти единственный вид блочных устройств. Устройствва произвольного доступа подразумевает диски, все что не лента. В 60-70 годах в качестве дисковых устройств использовались накопители на жестких и гибких магнитных дисках. ПОзжепоявились дисковые устройства которые работают наиных физических принципах. Магнито-оптиеские и оптические диски, dvd, cd. В это же время распростаняют твердотельные накопители, flash,ssd и что-то еще.

Для этих типов периферийных устройств понятие диск является условным тк ностиели не имеют форму диска. Но тк новые устройства используют теже аппарвтные интерфейсы то с точки зрения программной архитектуры физические различия не существенные.

Операция которую делаают с магнитным диском: низкоуровневое форматирование, оно заключается в разметке поверзности на дорожке магнитной записи разделенные на сетки. Расстояния между дорожками определяется шагом чтения и записиа разбиение на секторы выполняется программно с помощью записи данных на дорожки в моменты рассичитанные на основании известной скорости вращения.

Для всех операций с диском кроме форматирования сектор это минммальная единица чтения или записи данных. Совокупность дорожек одинакового радиуса на всех поверзнсотях диска называется цилиндром. Сектор состоит из заголовка, блока данных, контрольной информации.

Заголовок сектора содержит адрес и размер. Между секторами и внутри секторов есть промежутки использующитеся при поиске заданного сектора. Размер блока данных не может быть очнеь маленьким тк при этом большую часть дискового пространства занималибы промежутки.

Размер сектора на IBM совместимых пк всегда равен 512 байт.

Реалтзация кэширования.

Это такая организация вводавывода при которой данные не передаются непосредственно с усстройва заданной области памяти или из области памяти на устройство. А предварительно направляются во вспомогательную область памяти называемую буффером. Организуемой системы буффером не видимой для прикладного программиста. Он получает данные как готовый результат.Данные могут проходить через нессколько буффероф разного назначения. Существует несколько причин для использования буферизации:

1. Сглаживание неравномерности скоростей процессора. Часто в работе операционной системы возникают функции когда олин процесс порожаетданные котоыре должны оперативно обрабатываться другим процессом. Скорстьданных не равномерна. Интервалы времени интер=нсивного поступления данных перемешиваются спростоями. Обработка данных ПО не идет с постоянной скоростью В результате хотя средняя скорость обработки номральная не искоючено что в некоторые моменты обрабатывающая программа будет переполнена это может приветси к потере части данных не успевших пройти обработку. И решение данной ситуации это использовать буффер размер которого достаточно велик.
2. Распараллеливание ввода и обработки. В системах есть аппааратная возможность совмещения ввода-вывода и обработки данных процессором. Для реализации этого используют асинхронного ввода-вывода. А аднные буферизируются. После заполнения буффера его данные обрабатываются. Ихобработка выполняется параллельно с накоплением следдующей порции данных. Более эффективная схема это схема с 2 переключаемыми буфферами.

Согласование размеров логической и физической записи.

# Лекция 7

Логическая запись – это порция данные указанная в операторе ввода-вывода. Размер логической записи определяется логикой работы программы или логической структурой бд. При фактическом выполнении чтения или записи на блочное устройство обрабатывается порция данных которая называется физической записью или блоком. Размер физической записи определяется особенностями устройства и никаким образом не связан с логикой программ.

Кэширование дисков, специфическая форма буферизации. Ознчает использование небольшого объема но быстродействующей памяти для уменьшения кол-во числа обращений к более медленной памяти большого объема.Идея кэширования основывается на гипотезе о локальности ссылок. Эта гипотеза о если в какой-то момент времени произошло обращение к куску памяти то в ближайшее время можно с высокой ввероятностью ожидать повторение обращения к тем же самым данным. Локальность ссылок нельзя считать законом но практика показывает что эта гипотеза оправдывается для подавляющего большинства программ. При работе программы выполняется чтение очередных команд и обращение к ячейкам переменных. Но адрес очередной команды чаще всего равен адресу предыдущей команды + ее длина до тех пор пока не встретится команда перехода. Обращение к переменной редко бывает однокатным и часто значение переменной читается, изменяется и записывается обратно. Обработка массива данных обычно строится в виде цикла в котором плавно изменяется адрес очередного элемента массива и при этом часто происходит обращение к параметром. В этих ситуациях гипотеза подтверждается. В современных пк может использоваться нессколько уровней кеширования.

К работе ОС имеет отношение программное кэширование устройств произвольного доступа.

Гипотеза имеет следующий вид, если программы выполнила чтение или запись данных из некоторого блока диска то вероятно в скором будущем последует еще операции чтения или записи рядом с предыдущем блоком.

В роли быстродействующей памяти выступает массив который размещен в системной памяти. Каждый буффер состоит из заголовка и блока данных, соответствующего размера блоку то есть сектору диска. Заголовок буфера соержит адрес блока диска копия которого в данный момент содержится в буфере. И несколько флагов которые характерезуют состояние буффера. Когда система получает запрос на чтение или запись определенного блока данных диска то она проверяет не содержится ли в данный момент копия этого блока в одном из буфферов кэша.

В случае записи данных следует также в заголовке буффера отметить с помощью спец флага что содержимое буффера не соответствует данным на диске. Если требуюмый блок диска не найден то выделяем буффер. Общее кол-во буфферов ограничено. Чтобы отдать один из них под требуемый блок надо вытеснить один из блоков при этом он должен быть очищен, то есть записан на диск. При вытистнении чистого блока никаких операций с диском выполнять не надо.

Необходимо решить следующий вопрос: какой из блоков хранящийся в кэше. Если вопрос решится неправильно, то работа системы затормозится.

Алгоритм LFU. Выбирать блок который давно не используектся.

Другой вариант: Все буферы связаный в линейный список. Некоторые буферы уходят в конец списка. Меняет только ссылки в заголовках без перемещения данных. ПО этому принципу в говоле хранится дольше всего не использованный буфер.

Запись блока из кэша на диск возможно в 3 случаях:

1. Выбор блока для вытеснения из кэша
2. Закрытие файла
3. Операция принудительной очистки всех буфферов.

В unix Очистка всех буфферов выполняется каждые 30 секунд. Кэширование операции записи на диск в отличие от кэширования чтения создает опасность потери данных. При случайном сборе системы, отключении питания, некорректного извления флэшкти и др. Может быть важная информация не будет сохранена.

Программа требующие высокой надежности данных. Записывают данные сразу на диск и кэш не используется. Либо в кэш буффер заносится копия данных.

Посик требуемого блока данных в кэше, для этого система просматривает заголовки буфферов. В unix каждый буффер может входить одновременно в два линейных списка. Один из них список свободных блоков LFU, который используется для определения блоков подлежащего вытеснению. Для доступа к элементам спсика есть два указателя.

Хэш цепока – используется для ускорения поиска нужного блока. При записи в буффер данных соответсвующий блок диска номер хэш цепочки который будет помещен определяется как осток от деления номера блока на n, где n кол-во хэш цепочек. Когда система ищет блок она по номеру блока в какой из цепочек этот блок должен находиться.

Особенность: кеширования диском в unix.

При обнаружении в начале списка свободных блоков грязного буфера система запускает процесс их очистки и не ждет конца а выбирает для вытеснения первый чистый блок.

Блоки возвращаются в начало списка свободных блоков и остаются первыми кондидатами на вытеснение.

Опережающее чтение. Если обработка данных ведется последовательным образом от начала к концу, от кеширование не дает значительного эффекта. Обработанные из одного блока , то блок в кэше бесполезный.

Оно заключается в том что при обращении к некоторому диска система выполнив чтение некоторого блока считывает затем еще несколько блоков

В винде программа открывающая файл может указать системе для какого способа доступа желательно оптимизировать механизм буферизации.

При загрузках ОС и запусках приложений система выполняет ожидаеме операции чтения еще до того как они будут запрощены.

ФАЙЛЫ, основные задачи управления данными

# Лекция 8

Управление данными – понимается как управление файлами. Файл – набор данных хранящийся на периферийном устройстве и доступный по имени, при этом конкретное расположение данных на устройстве не интересует пользователя. До изобретения файла пользователь должен был обращаться к своим данным указывая их адреса на диске или на магнитной ленте.

Понятие файловой системы означает стандартизированную совокупность структур данных а также алгоритмов и программ данных которые обеспечивают хранение файлов и выполнение операций с ними. Мощность ОС поддежривает использование нескольких разных файловых систем и наоборот.

Задачи решаемые подсистемой управления данных:

1. Выполнение операций создания, удаления, переименования, поиска, чтения и записи файла, а также ряд вспомогательных функций.
2. Обеспечение эффективного использования дискового пространства и высокой скорости доступа к данным.
3. Обеспечения надежности хранения данных и их восстановления в случае сбоя
4. Защита данных пользователя от несанкционированного доступа
5. Управление одновременным доступом к данным со стороны нескольких процессов

Характеристики файлов и архитектура файловой системы:

С каждого файла связан набор атрибутов. Состав атрибутов сильно различается для разных ФС. Примерный список атрибутов:

1. Имя файла, макс длина имени около 250 символов.
2. Расширение имени. Правая часть имени отделенная точкой. Иногда он не часть имени и хранится отдельно. Используется для выбора программ которые должны обрабатывать данные этого типа.
3. Тип файла. Некоторые ОС выделяют разные типы файлов: символьные, двоичные, данные, файл программы
4. Размер файла
5. Временные штампы. Отметки даты и возможные времени суток. Важнейший из временных штампов: создание, последнее изменение.
6. Номер версии. Не прижился… В винде есть теневые копии, которые хранят в сжатом виде изменения.
7. Владелец файла. Нужен многопользовательским системам для защиты файлов. Обычно владелец – создатель.
8. Атрибуты защиты. Они указывают какие права доступа к файлам имеют различные пользователи, в том числе и владелец.
9. Тип доступа. Допустимый тип доступа: последовательный, произвольный, индексные типы. Которые обеспечивают быстрый поиск данных в файле. Сейчас используют подход в одинаковыми типами доступа а ускорение поиска занимается другое ПО (БД или другое)
10. Размер записи.
11. Флаги, битовые атрибуты. Наиболее распространенный – только для чтения.
12. Данные о размещении файла на диске. Пользователь не знает о размещении.
13. Мета-данные

Сетевая структура файловой системы в отличие от иерархической требует доп средств для организации связи между каталогами и файлами. Такими средствами являются жесткие и символические связи. Жесткая связь – это имя и файл. Особенность сетевой архитектуры в том что любой файл может иметь несколько жестких связей. Польза от нескольких имен в том что если использовать часто какой-то файл данных лежащий глубоко в одной из ветвей дерева каталогов вместо того чтобы указывать длинный путь можно создать новую жесткую связь. Все жесткие связи или имена одного файла равноправны. Символическая связь – это файл который содержит полное имя другого файла. Когда файл символической связи используется как аргумент системной команды ОС автоматически подставляет нужный файл.

Если администратор системы решил заменить файл оставив тоже самое имя файла. Если некоторые пользователи хранили жесткие связи на прежнюю версию. То они так и будут ей пользоваться пока не удалят ее имя и не создадут связи на новую версию. Символические связи не требуют особых изменений в организации хранения атрибутов файлов, по сравнению с иерархической структурой. Имя файла и все прочие атрибуты могут храниться вместе непосредственно в каталогах.

Записи с атрибутами файлов должны храниться в специально отведенных для них местах. Достаточно хранить только имя файла и ссылку на запись его атрибута.

Размещение файлов.

Область файлов – область данных диска отведенная для хранения файлов можно представить как линейную последовательность адресуемых блоков или секторов. Размещая файлы в этой области ОС должны отвести для каждого файла необходимое кол-во блоков и сохранить информацию о них. Информация сохраняется о том в каих именно блоках размещен данный файл. Существуюьт 2 основных способа использования дискового пространства для размещения файла:

1. Непрерывное размещение.
2. Сегментарное размещение. Доступ к любой части файла выполняется быстро.

При создании файла нужно знать его размер. Последующее возможное увеличение размера файла. Снижается надежность хранения данных тк ошибка при чтении или записи более вероятна чем порча данных лежащих на диске.