08.02.2023

Цвет учебника – digital

Примеры простых файловых систем CP/M

|  |  |
| --- | --- |
| 0xFFFF | BIOS |
|  | CP\M |
|  | Оболочка |
|  | Программа пользователя |
| 0x100 0 | Нулевая страница |

В файловой системе всего один каталог, содержащий записи фиксированного размера 32 байт. В этом каталоге перечисляются всей файлы системы. Размер битового массива для 150-КБ диска всего 23 байта, постоянно хранится в оперативной памяти.

Работы системы:

Пользователь набирает команду. Оболочка копирует ее в буфер в нижние 256 байт памяти. Затем ищет вызываемую программу и загружает ее в память по адресу 256 (0x100). Команда обнаруживает свои параметры в буфере командной строки. Команде разрешается использовать память оболочки. Закончив работу, программа сообщает ОС, что следует перезапустить оболочку.

Формат каталоговой записи:

* Код пользователя – владелец файла
* Имя файла – 8 символов верхнего регистра
* Расширение (необязательно) – до трех символов
* Счетчик байтов – размер файла в единицах по 128 байт

Файловая система MS-DOS

Формат описателя файла системы MS-DOS:

* Имя файла – 8 байт
* Расширение – 3 байт
* Атрибуты – 1 байт
* Зарезервировано – 10 байт
* Время – 2 байт
* Дата – 2 байт
* Номер – 2 байт
* Размер – 4 байт

Windows98

Формат каждого фрагмента длинного имени (LFN):

* Последовательность – 1 байт
* 5 символов – 10 байт
* Атрибуты – 1 байт
* 0 – 1 байт
* Контрольная сумма – 1 байт
* 6 символов – 12 байт
* 0 – 2 байт
* 2 символа – 4 байт

Файловая система Unix

В Unix всегда применялась многопользовательская файловая система. Имена файлов могут содержать до 14 символов: любых, кроме «\0». Каталог Unix использует по одной записи для каждого файла каталога. Запись состоит из 2 полей: имени файла и номера i-узла.

Запись каталога файловой системы UNIX V7:

* Номер узла – 2 байта
* Имя файла – 14 байт

I-узлы содержат атрибуты и дисковые адреса.

15.02.2023

Особенности распределенных систем

Отличием распределенных систем является межпроцессорная взаимосвязь. В централизованных имеется разделяемая память, в распределенных она отсутствует. Для взаимодействия используется передача по сети. Используется 2 основных примитива: отправка и получение сообщений.

Способы адресации

Для отправки сообщений необходимо указать адрес получателя. Один из вариантов адресации – использование сетевых адресов. Использование сетевого адреса в качестве адреса получателя приводит к ограничению: на каждой машине должен выполняться только один процесс.

Альтернативная система – 2 части: номер машины, номер процесса. Недостаток – нельзя работать с другим сервером.

Еще один вариант – задание адреса, не связанного с адресом машины. Для этого используется централизованный механизм распределения адресов процессов, который работает как счетчик. При получении запроса на выделение адреса, он возвращается значение счетчика и наращивает его на единицу. Недостаток – не обеспечивает в достаточной степени расширяемость системы.

Еще один подход – выбор собственного идентификатора из большого адресного пространства 64-битных чисел. Проблема – как процесс-отправитель может узнать номер машины процесса-получателя. В сети, поддерживающей широковещательный режим, отправитель широковещательно передает пакет, содержащий идентификатор назначений. Все ядра получают эти сообщения, проверяют адрес процесса, и если он совпадает с идентификатором одного из процесса, то отправляют ответного сообщение с сетевым адресом машины. Недостаток – перегрузка сети. Выход – выделить в сети специальную машину для отображения высокоуровневых символьных имен, называемую сервером имен.

Блокирующие и неблокирующие примитивы

Примитивы бывают блокирующими и неблокирующими. Блокирующий – процесс, выдавший запрос, останавливается до полного завершения примитива. Неблокирующий – управление возвращается вызвавшему процессу немедленно до того, как требуемая работа будет выполнена. Недостаток – отправитель не может модифицировать буфер сообщения, пока сообщение не отправлено, а узнать, отправлено ли сообщение, отправитель не может. Два выхода: либо заставить ядро копировать сообщение в свой внутренний буфер, а затем разрешить процессу продолжение выполнение, либо прерывать процесс отправителя после отправки сообщения для информирования, что буфер снова доступен.

Буферизируемые и небуферизируемые примитивы

Небуферизируемые – вызов получить, сообщает ядру машины, на котором он выполняется, адрес буфера, в которое следует поместить прибывающее сообщение. Возникает проблема, когда вызов «послать» происходит раньше вызова «получить». Вариант выхода – отказ от сообщения. Второй выход – хранить сообщение в ожидании нахождения отправителя.

Надежные и ненадежные примитивы

3 подхода:

1. Система не берет на себя никаких обязательств по поводу доставки сообщений
2. Ядро принимающей машины отправляет квитанцию подтверждения ядру отправляющей машины на каждое сообщение
3. Использование ответов в качестве подтверждения в тех системах, в которых запрос всегда сопровождается ответом

Вызов удаленных процедур

Идея удаленного вызова процедур состоит в расширении хорошо известного механизма передачи управления данных внутри программы, выполняемой на одной машины, на передачу управления и данных через сеть. Основные характеристики локальных процедур – асимметричность и синхронность. В связи с отсутствием общего адресного пространства возникают проблемы при передаче параметров и результатов. При реализации RPC участвует минимум 2 процесса – по одному на каждой машине. При аварии вызывающей процедуры удаленно вызванные процедуры станут осиротевшими, а при аварии удаленных процедур станут обездоленными родителями.

Динамическое связывание

Семантика RPC в случае отказа:

1. Клиент не может определить местонахождения сервера
2. Потерян запрос от клиента к серверу
3. Потеряно сообщение от сервера к клиенту
4. Сервер потерпел аварию после получения запроса
5. Сервер потерпел аварию после отправки запроса

22.02.2023

Лень – двигатель программирования. Unix

Обзор системы Linux

01.03.2023

Процессы в системе Unix

Единственными активными сущностями в системе Unix являются процессы. Каждый процесс запускает одну программу и изначально получает один поток управления. У процесса есть один счетчик команд, указывающий на следующую исполняемую команду процессора. Большинство версий Unix позволяет процессу после его запуска создавать дополнительные потоки. Unix – это многозадачная система, т.е. несколько независимых процессов могут работать одновременно. При загрузке системы автоматически запускаются десятки фоновых процессов, называемых демонами.

Процессы создаются:

1. Системный вызов fork создает точную копию исходного процесса, называемого родительским процессом. Новый процесс называется дочерним. У родительского и дочернего есть свои собственные образы памяти. Если родительский процесс изменяет впоследствии какие-либо свои переменные, изменения остаются невидимыми для дочернего процесса, и наоборот. Открытые файлы совместно используются родительскими и дочерними процессами. Системный вызов fork возвращает системному процессу число 0, а родительскому – отличный от нуля PID. Процессы распознаются по своим PID.

В Unix процессы могут общаться друг с другом с помощью разновидности обмена сообщений. Можно создать канал между 2 процессами, в который один процесс может писать поток байтов, а другой читать. Эти каналы называются трубами. Синхронизация процессов достигается блокированием процесса при попытке чтения из пустого канала. При помощи каналов организуются конвейеры оболочки.

Процессы могут общаться другим способом: при помощи программных прерываний. Один процесс может послать другому сигнал. У процесса получателя есть выбор: проигнорировать сигнал, перехватить или убить процесс. Если выбран перехват, должна быть указана процедура обработки сигнала, ее должен указать тот, кто послал. Управление передается обработчику. После обработки управление передается туда, откуда пришел сигнал. Процесс может посылать сигналы только членам его группы процессов, состоящей из прямого родителя, всех прародителей, братьев и сестер, а также детей, внуков и правнуков. Процесс может послать сигнал сразу всей своей группе за один системный вызов.

Системные вызовы управления процессами

Предназначены для управления процессами.

Работа оболочки:

1. Считывает команды с терминала
2. С помощью системного вызова fork выполняет введенную команду
3. Ждет окончания работы дочернего процесса
4. Считывает следующую команду

Если процесс уже завершил свою работу, а родительский процесс не ожидает этого события, то дочерний процесс переводится в так называемое состояние зомби, то есть приостанавливается.

Обработчик сигнала может выполняться сколь угодно долго. Когда процедура обработки сигнала завершает работу, она возвращается к той точке, в которой ее прервали.

Системные вызовы управления потоками

В первой версии системы не было потоков. Системные вызовы управления потоками стандартизованы в POSIX. POSIX не определяет, должны ли потоки реализовываться в пространстве ядра или пользователя.

Синхронизация потоков осуществляется при помощи мьютексов. Чтобы гарантировать, что только один поток в каждый момент времени имеет доступ только к общему ресурсу, предполагается, что потоки блокируют (захватывают) мьютекс перед обращением к ресурсу и разблокируют, когда ресурс им более не нужен.

Мьютексы предназначены только для кратковременной блокировки. Для долговременной синхронизации представляются переменные состояния. Один поток ждем, когда переменная примет определенное значение, а другой сигнализирует ему изменением этой переменной.

15.03.2023

Реализация процессов в Unix

У каждого процесса есть пользовательская часть, в которой работает программа пользователя. Когда один из его потоков обращается к системному вызову, происходи эмулированное прерывание с переключением в режим ядра. После этого поток начинает работу в контексте ядра с отличной картой памяти и полным доступом к ресурсам машины.

Ядро поддерживает 2 ключевые структуры данных: таблицу процессов и структуру пользователя. Таблица процессов является резидентной, она не выгружается. Структура пользователя выгружается на диск, освобождая место в памяти. Информация в таблице дескрипторов процессов подразделяется на категории:

1. Параметры планирования
2. Образ памяти
3. Сигналы
4. Разное

В структуре пользователя в контексте процесса содержится информация, которая не требуется, когда процесса физически нет в памяти и он не выполняется. Информация о сигналах хранится в таблице процессов, постоянно находящейся в памяти, даже когда процесс отсутствует в памяти.

Данные, хранящиеся в структуре пользователя, содержат:

1. Машинные регистры
2. Состояние системного вызова
3. Таблица дескрипторов файлов
4. Учетная информация
5. Стек ядра

Процессы создаются:

1. Когда выполняется системный вызов fork, вызывающий процесс обращается в ядро и ищет свободную ячейку в таблице процессов, в которую можно записать данные о дочернем процессе
   1. Если свободная ячейка находится, системный вызов копирует туда информацию из ячейки родительского процесса, затем он выделяет память для сегментов данных из стека дочернего процесса и копирует туда соответствующие сегменты родительского процесса. Структура пользователя копируется вместе со стеком. Сегмент кода может либо копироваться, либо использоваться совместно, если доступен только для чтения.

Потоки Unix

Реализация потоков зависит от того, поддерживаются они ядром или нет

Потом в системе Linux

Для реализации потоков в Linux используется системный вызов clone. Он создает поток либо в текущем процессе, либо в новом. Если новый поток находится в текущем процессе, он совместно с другими потоками использует адресное пространство и любое изменение, произведенное одним потоком, видно остальным.

Планирование в Unix

У алгоритма 2 уровня. Низкоуровневый алгоритм выбирает следующий процесс из набора процессов в памяти и готовых к работе. Высокоуровневый перемещает процессы из памяти на диск и обратно. В низкоуровневом алгоритме используется несколько очередей. С каждой очередью связан непересекающийся диапазон значений приоритетов. Процесса, выполняющиеся в режиме пользователя, имеют положительные значения приоритетов, в режиме ядра – отрицательные. Отрицательные значения приоритетов считаются наивысшими.

Планировщик ищет очередь, начиная с самого высокого приоритета (наименьшее отрицательное значение), пока не находится очередь, в которой есть хотя бы один процесс. Выбирается первый процесс, ему разрешается работать в течение некоего кванта времени или пока не заблокируется. Если использован весь квант времени, процесс помещается в конец очереди. Раз в секунду приоритет каждого процесса пересчитывается.

Jiffy – 10 мс. Значение показателя nice – от -20 до +20.

В Linux 3 класса потоков:

1. Потоки реального времени FIFO. Имеют наивысший приоритет и не могут прерываться другими потоками, кроме тех же потоков FIFO
2. Потоки реального времени, обслуживаемые в порядке циклической очереди. Могут прерываться таймером
3. Потоки разделения времени

У каждого потока есть приоритет планирования, значение по умолчанию – 20. Может быть изменено с помощью системного вызова nice. Помимо приоритета с каждым процессом связан квант времени, в течение которого процесс может выполняться. По умолчанию это 10 миллисекунд. Этот интервал называется jiffy.

Добродетель реального времени – 100 + приоритет

Загрузка в Unix

Когда компьютер включается, в память считывается и исполняется первый сектор загружаемого диска. Этот сектор содержит 512-байтовую программу, позволяющую найти и запустить загрузчик с загрузочного устройства. Программа-загрузчик сначала копирует саму себя в фиксированный адрес памяти в старших адресах. Загрузившись, считывает корневой каталог с загрузочного устройства. При этом она должна понимать формат файловой системы и каталога. Затем считывается ядро ОС и передает ему управление.

Управление памятью в Unix

У каждого процесса в системе Unix есть адресное пространство, состоящее из 3 сегментов:

1. Текст – программный сегмент, содержит машинные команды, образующие исполняемый код программы. Обычно разрешен только для чтения. Не изменяется ни по размеру, ни по содержанию
2. Данные – содержит переменные, строки, массивы и другие данные. Состоит из 2 частей:
   1. Инициализированные – данные заданы при запуске программы
   2. Неинициализированные – обычно устанавливаются в ноль. Вместо неинициализированных переменных компилятор помещает одно слово, содержащее размер области неинициализированных данных в байтах

Сегмент данных может модифицироваться. Unix разрешает сегменту расти и уменьшаться.

1. Стек – обычно начинается у старших адресов и растет к нулю.

Большинство систем Unix поддерживает текстовые сегменты совместного пользования. На некоторых компьютерах аппаратное обеспечение поддерживает разные адресные пространства для команд и данных.

Многими версиями поддерживается отображение файлов на адресное пространство памяти. Преимущество отображение файла на память является, что 2 или более процессов могут одновременно отобразить на свое адресное пространство один и тот же файл. Отображение на адресное пространство памяти временного файла представляет собой механизм реализации общей памяти для нескольких процессов, у которого будет высокая пропускная способность.

Системные вызовы управления памятью

22.03.2023

Реализация управления памятью в Unix

Большинство систем основывалось на свопинге подкачки, когда загружалось больше процессов, чем могло поместиться на памяти, некоторые из них выгружались на диск. Выгружаемый процесс всегда выгружался на диск целиком. Перемещение данных между памятью и диском управлял верхний уровень планировщика, называемый свопером.

Свопер сначала рассматривал блокированные процессы. Если такие находились, из них выбирался процесс с наивысшим значением суммы приоритета и времени пребывания в памяти. Если блокированных процессов не было, тогда по тем же критериям выбирался готовый процесс.

Для загрузки процессов свопер исследовал список выгруженных процессов. Если процессы в состоянии готовности обнаруживались, из них выбирался процесс, дольше всех находящийся на диске. Затем свопер определял легкий или тяжелый свопинг. Легкий – не требовалось дополнительное высвобождение памяти. Тяжелый – для загрузки требовалось выгрузить один или несколько процессов. Процесс повторялся до тех пор, пока на диске не оставалось процессов, готовых к работе, или в памяти не оставалось места для новых процессов.

Постраничная подкачка в системе Unix

Чтобы работать, процессу не нужно целиком находиться в памяти. Страницы с сегментами текста, данных и стека загружаются в память динамически. Для этого используются структура пользователя и таблица страниц.

Страничная подкачка реализуется частично ядром, частично – новым процессом, называемым страничным демоном. Если демон обнаруживает, что количество страниц в списке свободных страниц мало, он освобождает дополнительные страницы. При этом карта памяти содержит информацию о содержимом страничных блоков.

Алгоритмы замещения страниц

Демоном просыпается раз в 250 миллисекунд, сравнивает количество свободных блоков с параметром обычно «четверть объема памяти». Если число свободных блоков меньше этого параметра, демон переносит страницы из памяти на диск, пока не достигнет этого параметра. Страничный демон использует модифицированную версию алгоритма часов. При большом объеме памяти используется алгоритм с двумя стрелками.

Каждые несколько секунд свопер проверяет, есть ли на диске готовые процессы. Каждому процессу на диске присваивается значение, зависящее от времени пребывания в выгруженном состоянии, размера, значения, использовавшегося при обращении к системному вызову, и того, как долго процесс спал перед тем, как бы выгружен на диск.

Управление памятью в Linux

Каждый процесс получает 3 ГБ виртуального адресного пространства для себя и 1 ГБ памяти для страничных таблиц. Виртуальное адресное пространство делится на однородные непрерывные области, выравненные по границам страниц. Размер страниц фиксирован. Каждая область описывается записью, все структуры связаны в список.

Система Linux поддерживает динамически загружаемые модули. Динамическая память может получить участок памяти произвольного размера, используя дружественный алгоритм.

Текстовые сегменты и отображаемые на адресное пространство памяти файлы выгружается из соответствующих файлов на диске. Все остальное выгружается либо в область подкачки, либо в файл подкачки, которых может быть до 8.

29.03.2023

Файловая система UNIX

Файл в системе Unix – последовательность байтов произвольной длины, содержащая произвольную информацию. Значение битов в файле целиком определяется владельцем файла. Разрешается использовать все символы.

Многие программы ожидают, что имена файлов должны состоять из основного имени и расширения, разделяемого точкой. Расширения могут иметь произвольную длину, файлы могут иметь несколько расширений. Для удобства файлы могут группироваться в каталоги. Каталоги на диске хранятся в виде файлов. Каталоги могут содержать подкаталоги. Корневой каталог – «/» – обычно содержит несколько подкаталогов.

Существует 2 способа задания имени файла:

1. Задание абсолютного пути – длинные и неудобные.
2. Unix позволяет пользователям обозначить каталог, в котором они работают, как рабочий или текущий. Путь файла, заданный относительно рабочего каталога, называется относительным

Unix позволяет использовать связи, представляющие собой записи каталога, указывающие на другие файла. Все устройства представлены специальными файлами.

Одним из свойств файловой системы Unix является блокировка. Стандартом POSIX. Представляется механизм, позволяющий процессам за одну неделимую операцию блокировать даже единственный байт файла или целый по желанию. Механизм блокировки требует от процесса указать блокируемый файл, начальный байт и количество байтов.

Стандартом определены 2 типа блокировки: с и без монополизации. Если часть файла содержит блокировку без монополизации, то повторная установка блокировки без монополизации на это место файла разрешается. Но попытка установки блокировки с монополизацией будет отвергнута. Если какая-либо область файла содержит блокировку с монополизацией, то любые попытки заблокировать любую часть этой области будут отвергаться, пока не будет снята монопольная блокировка.

При установке блокировки процесс должен указать, хочет ли он сразу получить управление или будет ждать, пока не будет установлена блокировка. Если выбран вызов с ожиданием, то он блокируется до тех пор, пока не будет снята блокировка, установленная другим процессом, после чего процесс активизируется и получается сообщение, что блокировка установлена. Если без ожидания – он немедленно получает ответ об успехе или неудаче.

Вызовы файловой системы в Unix

Дескриптор файла – небольшое неотрицательное число.

Реализация файловой системы Unix

Все системы Unix могут поддерживать несколько дисковых разделов, каждый со своей файловой системой. Раздел содержит загрузочный блок, суперблок, i-узлы и блоки данных. Блок 0 не используется системой и содержит программу загрузки. Блок 1 представляет собой суперблок. В нем хранится информация о размещении файловой системы: количество i-узлов, количество дисковых блоков, а также начало списка свободных блоков диска. За суперблоком располагаются i-узлы, нумеруются от 1 до максимального числа. Узел имеет длину в 64 байта и описывает один файл. За i-узлами располагаются блоки с данными.

05.04.2023

Когда ядро получает управление, ему передаются 3 параметра: количество байт для записи, начало, размер буфера. Все остальные параметры можно получить из внутренних таблиц, относящихся к пользователю. Одной из таблиц является массив дескрипторов файла. Он проиндексирован по номерам дескрипторов файла и содержит по одной записи для каждого открытого файла. По дескриптору файловая система должна найти соответствующий i-узел соответствующего файла.

Вводится в обращение новая таблица – таблица открытых файлов между таблицей дескрипторов файлов и таблицей i-узлов. Работающие процессы получают доступ к позиции в файле и к i-узлу файла. I-узел содержит адреса первых 10 блоков файла. Если позиция в файле попадает в первые 10 блоков, то считывается нужный блок. Для поддержки файлов, длина которых превышает 10 блоков, в i-узле содержится дисковый адрес одинарного косвенного блока. Для файлов, размер которых превышает 256 КБ, используется двойной косвенный блок.

В файловой системе Беркли реорганизованы каталоги, длина имен увеличена до 255 символов. Каждый каталог состоит из целого количества дисковых блоков. Внутри каталога записи файлов и каталогов не отсортированы, и в конце каждого блока может оказаться несколько неиспользованных байтов.

Поиск в каталогах производится линейно, поэтому занимает много времени. В Беркли проведено разбиение дисков на группу цилиндров, у каждой из которых был собственный супер-блок, i-узлы и блоки данных. Идея данного метода – хранить i-узлы и блоки данных ближе друг другу.

В Беркли используются блоки не одного, а двух размеров.

Файловые системы Linux

В файловой системе ext2 диск делится на группы блоков независимо от того, где располагаются границы цилиндров. Каждая группа блоков начинается с супер-блока, в котором хранится информация о том, сколько блоков и i-узлов находится в группе. Затем следует описатель группы, содержащий информацию о расположении битовых массивов, количестве свободных блоков и i-узлов в группе, а также количестве каталогов в группе. В 2-битовых массивах ведется учет информации о количестве свободных блоков и свободных i-узлов. Далее располагаются сами i-узлы размером 120 байт каждый. В отличие от Беркли, в Linux используются дисковые блоки одного размера в 1 КБ.

Особенности ext3 и ext4

Журнал поддерживает 3 основные структуры данных: запись журнала, описатель атомарной операции, транзакция

Ext3 можно настроить таким образом, чтобы она хранила журнал либо всех изменений, либо метаданные файловой системы.

Основные достоинства ext4:

* Увеличение максимального размера до одного экзабайта
* Прирост производительности за счет реализации механизма пространственной записи файла

Еще одной файловой системой является /proc. Для каждого процесса файловой системы создается подкаталог в каталоге /proc. Имя каталога формируется из id процесса в десятичном формате. В каталоге располагаются файлы, которые хранят информацию о процессе: командную строку, строку окружения, маски сигналов.

В основе файловой системы NFS лежит представление о том, что пользоваться ею может произвольный набор клиентов и серверов. Каждый сервер файловой системы NFS экспортирует один или несколько ее каталогов, предоставляя доступ к ним удаленным клиентам. Как правило доступ к каталогу предоставляется вместе с подкаталогами. Клиенты получают доступ к каталогам системы, монтируя эти каталоги.

Протоколы файловой системы NFS

Протокол – набор запросов, посылаемых клиентами серверам, и ответ серверов клиентам. Протоколов два. Первый управляет монтированием каталогов. В большинстве версий Unix поддерживается автомонтировка. Она позволяет ассоциировать с локальным каталогом несколько удаленных каталогов. Ни один из каталогов не монтируется при запуске системы.

Второй протокол предназначен для доступа к каталогам и файлам. В NFS поддерживается большинство системных вызовов за исключением «открыть» и «закрыть» файл. Преимущества: серверу не нужно думать об открытых соединениях между обращениями к нему. Такие серверы называются серверами без состояния.

12.04.2023

Безопасность в Unix

У каждого зарегистрированного пользователя есть свой уникальный UID. Это целое число от 0 до 65 535. Идентификатором владельца помечаются файлы, процессы и другие ресурсы. П умолчанию владельцем является пользователь, создавший файл. В процессе работы пользователя можно поменять.

Пользователи могут организовываться в группы, которые тоже нумеруются 16-разрядными целыми числами, называемыми GID. Вначале пользователь мог принадлежать только одной группе, но в некоторых версиях Unix он может принадлежать нескольким.

Когда создается файл, он получает UIDи GID создающего его процесса. Файл также получает набор прав доступа. Разрешения определяют доступ к файлу для владельца, для других членов группы владельца и прочих пользователей. Для каждых выделяется 3 вида доступа: чтение, запись и исполнение. Возможность исполнения имеет смысл, если файл – исполняемый двоично.

Пользователь, UID которого равен 0, является суперпользователем. Суперпользователь может писать и читать все файлы в системе. Процесс с UID, равным 0, также обладает возможностью обращаться к небольшой группе системных вызовов, доступ к которым запрещен обычным пользователям. У специальных файлов есть те же самые биты защиты.

Реализация безопасности в Unix. Когда пользователь входит в систему, программа регистрации запрашивает его имя и пароль. Затем она хеширует пароль и ищет его в файле паролей. Если пароль введен верно, программа регистрации считывает имя программы оболочки пользователя. Затем программа регистрации устанавливает для себя UID и GID. После этого открывается клавиатура для стандартного ввода и экран для стандартного вывода. Далее выполняется оболочка, которую указал пользователь.

Уровни программирования в Windows

Windows можно разделить на 3 семейства:

1. MS DOS
2. Windows 95, 98, ME
3. Windows NT

MS DOS

MS DOS была 16-битной однопользовательской ОС с интерфейсом командной строки.

Windows 95, 98, ME

Эти ОС были 32-разрядными, но содержали большие куски 16-разрядного ассемблерного кода. Продолжали использовать файловую систему MS DOS. Основное отличие 98 от 95 – был интегрирован интерфейс интернета в рабочий стол пользователя. Windows 98 стала многозадачной. У каждого процесса было 4-ГБ адресное пространство, в котором 2 ГБ принадлежали процессу, а 1 ГБ совместно использовался всеми процессами системы.

Windows на базе NT

Система проектировалась с возможностью переноса на другие процессоры. Упор делался на безопасность и надежность, а также на совместимость с версией MS DOS.

19.04.2023

Уровни программирования в Windows

Несмотря на наличие POSIX, практически весь написанный для Windows код используется напрямую либо для win32, либо поверх win32. Ядром ОС является программа режима ядра NTOS, которая обеспечивает традиционные интерфейсы системных вызовов, на которых построена остальная часть системы. Большинство приложений Windows написано с использованием win32. Подсистема «клиент-сервер» входит в состав ОС Windows NT и представляет собой часть пользовательского режима подсистемы win32.

Когда процесс пользовательского режима вызывает функцию с участием консольных окон, создание процесса потока и др., вместо запроса системного вызова обращаются к процессу CSRSS путем быстрого локального межпроцессного вызова, а затем CSRSS делает большую часть реальной работы без опасности компрометировать ядро. Вызовы к оконному менеджеру обрабатываются драйверами ядра напрямую.

Перехватчик win32 определяет подсистему, которая требуется программе, затем делает запрос к подсистеме управления сеансами на запуск процесса подсистемы CSRSS. После этого за загрузку программы отвечает процесс подсистемы.

Ядро NT спроектировано таким образом, что оно имеет множество средств общего назначения, которые можно использовать для написания специфических систем, однако для правильной реализации любой системы окружения необходим специальный код. Собственные системные службы используют собственные системные вызовы ядра и его основных служб. Собственные системные вызовы содержат межпроцессные средства для управления адресами, потоками, описателями, а также исключениями для программ, написанных под определенную систему окружения.

В отличие от win32, система .net не реализована как официальная подсистема окружения на собственных интерфейсах ядра NT.

Собственный интерфейс прикладного программирования

Windows имеет набор системных вызовов. Они реализованы на уровне исполняющего модуля NT ОС. Системные вызовы используются программами низкого уровня, которые поставляются как часть ОС, а также драйверами устройств, работающих в режиме ядра. Собственные системные вызовы мало меняются от версии к версии и обеспечивают совместимость до уровня MS DOS.

В интерфейсе прикладного программирования NT есть вызовы для создания новых объектов режима ядра и обращения к существующим. Каждый вызов возвращает вызывающей стороне описатель, который используется для выполнения операций с объектом. Каждый агент имеет связанный с ним дескриптор безопасности, отражающий типы операций, которые возможно проводить с данным объектом.

Для представления названий в пространстве имен в NT диспетчер использует юникод и не делает различий между верхним и нижним регистром, но регистр сохраняет.

Для управления устройством загружается драйвер устройства и создается объект драйвера. В отличие от Unix, которая помещает корень своего пространства имен в файловую систему, корень пространства имен NT находится в пространстве имен ядра. При этом при каждой загрузке системе воссоздается свое пространство имен верхнего уровня. Именованный объект может быть отмечен как постоянный, т.е. существовать до удаления или перезагрузки системы.

Интерфейс прикладного программирования win32

Вызов функции win32 называется интерфейсом прикладного программирования. Windows поддерживает 2 специальные среды выполнения, которые называются windows on windows (wow). Wow64 позволяет 32-битным приложениям работать на 64-битных системах.

В win32 есть вызовы, относящиеся к безопасности. Каждый поток ассоциируется с объектом режима ядра, называемым маркером (токеном), который предоставляет информацию об идентификаторе и привилегиях потока. Каждый объект может иметь список управления доступом, который подробно указывается, какие пользователи могут обращаться к объекту и какие операции с ним выполняют. Модель безопасности способна к расширению и позволяет приложениям добавлять новые правила безопасности.

Реестр

Реестр – это центральная база данных, содержащая информацию для загрузки, конфигурирования системы, настройки под определенного пользователя. Реестр состоит из набора каталогов, каждый из которых содержит либо подкаталоги, либо записи. Реестр организованы в отдельные тома, называемые ульями.

Каждый улей хранится в отдельном файле. В нижней части иерархической структуры располагаются записи, называемые значениями. У каждого значения 3 части – имя, тип и данные. Имя представляет строку формата юникод. Тип может быть одним из 11 стандартных типов.

26.04.2023

Структура системы Windows

Центральный уровень – ядро. Загружается из файла ntoskrnl.exe. Часть nt, работающая в режиме ядра, называемая ntos, имеет два уровня: исполняющий и меньший по размеру, называемый ядром.

Верхний уровень – системная библиотека, содержит специальные точки входа, используемые ядром для инициализации потоков, а также диспетчеризации исключений и вызовов APC пользовательского режима.

Задача HAL – представить остальной части ОС некое абстрактное оборудование, которые скрывает специфические подробности, версию процессора, чипсет и пр. При использовании служб HAL и отсутствии прямых обращений к оборудованию для драйверов ядра требуется меньше изменений при переносе на новые процессоры.

Уровень ядра

Над уровнем HAL находится ntos, состоящий из 2 уровней: ядро и исполняющая система. Уровень ядра отвечает за планирование и синхронизацию потоков. Планировщик ядра отвечает за то, какие потоки выполняются на процессорах системы.

Каждый поток выполняется до тех пор, пока прерывание таймера не сигнализирует о том, что пора переключиться на другой поток, или до тех пор, когда потоку нужно ждать какого-то события, либо до тех пор, пока работоспособным не станет поток с более высоким приоритетом. При переключении с одного потока на другой планировщик обеспечивает сохранение регистров и прочего состояния оборудования, затем планировщик выбирает для выполнения на процессоре другой поток и восстанавливает сохраненные значения для выбранного потока. Кроме этого уровень ядра представляет поддержку низкого уровня для 2 классов механизма синхронизации: объектов управления и диспетчеризации.

Отложенный выбор процедур

Объекты управления включают объекты примитивов для потоков, прерываний, таймеров, синхронизации, профилирования, а также 2 специальных объекта: DPC и APC. Объекты DPC отложенного выбора процедуры используются для уменьшения времени выполнения TSR.

Оборудование системы присваивает прерыванием аппаратный уровень приоритета. Процессор реагирует только на те прерывания, которые имеют более высокий приоритет, чем используемые в данный момент. Если процедура прерывания выполняется слишком долго, то обслуживание прерываний низкого приоритета будет отложено, что может привести к потере данных. Для уменьшения времени обработки прерываний выполняются только критические операции такие, как запись результатов операций ввода-вывода и повторная инициализация устройств. Дальнейшая обработка прерываний откладывается до тех пор, пока уровень приоритета процессора не снизится и не перестанет блокировать другие прерывания.

Другой управляющий объект ядра APC, в отличие от DPC, который работает в контексте конкретного процесса, выполняется в контексте конкретного потока, который выполняется при возникновении исходного прерывания.

Объекты диспетчеризации

Объект диспетчеризации – это любой из обычных объектов режима ядра, на который пользователи ссылаются при помощи описателей, и содержащий структуру данных под названием заголовок диспетчеризации.

Исполняющая система

Исполняющая система находится ниже ядра ntos. Все компоненты разделены на внутренние и внешние структуры данных и интерфейсы. Большая часть выполняется в режиме ядра.

Драйверы устройств

Драйверы устройств – динамически связываемые библиотеки, динамически загружаются.

03.05.2023

Процессы и потоки в Windows

В Windows процессы являются контейнерами для программ. Они содержат виртуальные адресные пространства, описатели процессов и потоки. Каждый процесс имеет системные данные пользовательского режима, называемые блок среды процесса. Он включает список загруженных модулей, область память со строками окружения, текущий рабочий каталог и данные для управления кучами процесса.

Потоки – абстракции ядра для планирования процесса Windows. Каждому потоку присваивается приоритет. Потоки могут быть аффинизированными, чтобы выполняться только на определенных процессорах. Каждый поток имеет 2 отдельных стека вызовов: один для выполнения в пользовательском режиме, другой – для выполнения в режиме ядра. Также есть блок среды потока, который хранит специфические для потока данные пользовательского режима, в том числе области хранения для потока. Кроме этого существует еще одна структура данных, которую режим ядра использует совместно со всеми процессами, – это совместно используемые данные пользователя.

Процессы

Процессы создаются из объектов сегментов, каждый из которых описывает объект памяти. При его создании создающий процесс получает описатель для процесса, который позволяет ему модифицировать новый процесс посредством отображения сегментов, выделении виртуальной памяти, записи параметров данных и окружения, дублирования дескрипторов файлов в таблицу описателей.

Задания и волокна

Windows может группировать процессы в задания с целью применения ограничения к содержащимся в ним потокам, так как ограничения с помощью совместно используемый квоты или применения маркера ограниченного доступа не позволяет обращаться к системные объектам. Главное свойство заданий: если процесс оказался в задании, все созданные потоками процессы также будут находиться в задании. Процесс может находиться внутри только одного задания. Волокна создаются путем выделения места в стеке и структуре данных волокна в пользовательском режиме. Процессы содержат потоки, но потоки не содержат волокна. Потоки преобразуются в волокна, однако волокна могут создаваться и независимо от потоков. Потоки могу переключаться на то волокно, которое уже выполняется, поэтому программисты должны предусматривать синхронизацию. Издержки переключения между волокнами ниже, чем между потоками.

Для упрощения взаимодействия между потоками полезно создавать ровно столько потоков, сколько имеется процессоров, а также аффенизировать потоки.

Потоки

Любой процесс обычно начинается с одного потока, однако можно динамически создать дополнительные. Потоки являются основой планирования процессора. Поток имеет состояние, а процессы не имеют состояния планирования. Потоки можно создавать динамически при помощи вызова win32. Каждый поток имеет идентификатор потока, который выбирается из того же пространства, что и идентификатор процесса. Идентификаторы процессов и потоков кратны четырем с использованием специальной таблицы описателей. Таблица описателей не имеет ссылок на объекты, но использует поля указателя для указания на процессы и поток.

Обычно поток выполняется в пользовательском режиме, однако, когда делается системный вызов, он переключается в режим ядра и продолжает выполняться с теми же самыми параметрами. Каждый поток имеет 2 стека: один в пользовательском режиме, другой в режиме ядра.

Потоки являются концепциями планирования, а не владения ресурсами. Windows имеет несколько системных потоков. Они работают в специальном процессе, называемом системным процессом.

Межпроцессный обмен

Потоки могут вести обмен самыми разными способами: каналы, именованные каналы, почтовые слоты, сокеты, вызов удаленных процедур, совместно используемые файлы. Каналы имеют 2 режима: байтовый и режим сообщений. Именованные каналы имеют также 2 режима, но могут использоваться по сети. Почтовые слоты односторонние, их можно использовать по сети, не обеспечивают гарантированной доставки, позволяют посылающему процессу транслировать сообщение множеству адресов. Сокеты объединяют процессы на разных машинах. Удаленный вызов процедуры RPC – это способ процесса А сделать так, чтобы процесс Б вызвал процедуру в своем адресном процессе и вернул процедуру А. Существует ALPC – расширенный вызов локальных процедур – средство передачи сообщений в исполняющем режиме ядра, оптимизировано для локального компьютера, по сети не работает.

Синхронизация

Windows предоставляет множество механизмов: семафоры, мьютексы, критические области и события. Они работают с потоками. Семафоры – объекты режима ядра, поэтому имеют дескрипторы безопасности и описателей. Семафор можно инициализировать с заданным значением и максимальным значением. Мьютексы – также объекты режима ядра, не имеют счетчиков. Критическая секция является локальной для адресного пространства создающего потока. События – объекты режима ядра, имеют два вида: события уведомлений и события синхронизации. Может быть в одном из двух состояний: сигнализируемом и несигнализируемом.

Планирование

Windows не имеет центрального потока планирования. Поток входит в режим ядра и вызывает планировщик, чтобы увидеть, на какой поток переключиться.

10.05.2023

Управление памятью

В Windows NT6 каждый пользовательский процесс имеет собственное виртуальное адресное пространство. <…> 2 гигабайта адресов выделяет для пользовательского режима каждого процесса. Остальные 2 гигабайта используются в режиме ядра.

В компьютерах x86 и x64 виртуальное адресное пространство имеет замещение страниц по требованию со страницы фиксированного размера 4 КБ, но может применяться по 4 МБ посредством использования каталога страниц и обхода соответствующей таблицы страниц.

Нижний и верхний 64 КБ виртуального адресного пространства не отображаются. С отметки 64 КБ начинаются пользовательские приватные код и данные. Граница доходит до 2 ГБ. Верхние 2 ГБ – это виртуальная память ядра, доступная только для режима ядра.

Страница виртуальных адресов может в одном из 3 состояний: недействительная, зарезервированная, зафиксированная. Недействительная страница не отображается на объект раздела памяти, и ссылка на нее вызывает страничную ошибку. После того, как код и данные отображаются на виртуальную страницу, страница называется зафиксированной. Зарезервированная страница недействительная, но эти виртуальные адреса никогда не будут выделяться диспетчером памяти для других целей. Также могут быть атрибуты: читаемое, записываемое, исполняемое.

Файл подкачки

Резервное хранилище для зафиксированных страниц, которые не имеют соответствия конкретным файлам, называются файлами подкачки.

Стратегии выделения файлов подкачки:

* Простая – каждой виртуальной странице назначается страница из файла подкачки на диске (во время фиксации виртуальной страницы)
* Синхронная – ее использует Windows, зафиксированным страницам не выделяет места в файле подкачки до момента, когда их необходимо вытеснить в файл подкачки. Это используется для повышения производительности записи модифицированных страниц в файл подкачки. Когда хранящиеся в файле подкачки страницы считываются в память, они сохраняют свое место в файле подкачки до первой модификации. Если страница не модифицируется, она попадает в специальный список свободных физических страниц, называемый списком резервирования, из которого она может быть взята для повторного использования без необходимости обратной записи на диск. Если она модифицируется, то диспетчер памяти освобождает страницу в файле подкачки и единственный экземпляр страницы находится в памяти (маркируется только для чтения).

Windows поддерживает до 16 файлов подкачки, распределенных по нескольким дискам. Исполняемый код и данные только для чтения могут отображаться на пространство адресов любого используемого процесса. Поскольку эти страницы не модифицируются, их не нужно вытеснять в файл подкачки. При модификации страниц, помеченных только для чтения, Windows поддерживает страницу типа копирования при записи. То есть в начале на является обычной отображаемой страницей. При модификации любой части диспетчер памяти создает приватную копию для записи, затем обновляет таблицу страниц, чтобы она указывала на эту приватную копию и дает возможность потоку повторить запись.

Реализация управления памятью

На процессорах x86 поддерживает одно линейное адресное пространство размером 4 ГБ с подкачкой по требованию. Сегментация не поддерживается. Размер страниц обычно фиксирован и составляет 4 КБ. ОС может использовать страницы в 4 МБ для увеличения эффективности буфера быстрого преобразования адресов.

В отличие от планировщика (выбирает потоки для выполнения, не заботится о процессах), диспетчер памяти полностью занимается процессами и не заботится о потоках. Когда выделяется область виртуального адресного пространства, диспетчер памяти создает дескриптор VID, в котором содержится диапазон отображаемых адресов, секция представления файлов резервного хранения и смещение его отображения, а также разрешения.

Обработка страничных ошибок

Подкачка по требованию в страничной памяти управляется страничными ошибками. При каждой ошибке происходит прерывание.

Каждая страничная ошибка может быть отнесена к одной из 5 категорий:

1. Страница не зафиксирована,
2. Попытка обращения к странице с нарушением разрешений,
3. Попытка модификации страницы типа копирования при записи,
4. Необходимо увеличение стека,
5. Страница зафиксирована, но в данное время не отображена.

Если диспетчер памяти может отработать страничную ошибку посредством нахождения нужной страницы в памяти (а не считывать ее с диска), то ошибка называется мягкой. Если нужна копия с диска – жесткой. Когда в таблице страниц не отображено не одного процесса, она входит в один из списков: свободных, модифицированных или резервных.

Алгоритм замещения страниц

Каждый процесс имеет рабочий набор, размер и состав рабочего набора меняется по мере работы потоков процесса. Начальный набор – 20-50 страниц минимум, 40-345 – максимум. В условиях дефицита памяти диспетчер памяти втискивает процессы в рабочие наборы. При этом диспетчер рабочих наборов запускается каждую секунду и запускается из потока балансировщика.

24.05.2023

Файловая система NTFS