

Философия ОС UNIX

Виктор Назаров

25 февраля 2004 года

UNIX© — зарегистрированная торговая марка .
 Linux© — зарегистрированная торговая марка
 Линуса Торвальдса (Linus Torvalds).
 Plan 9© — зарегистрированная торговая марка Bell Laboratories.
 IBM© —
 Microsoft© —
 Microsoft DOS© —
 Intel©

В данном тексте приняты следующие соглашения о компьютерных терминах:

Слова *вычисление* и *алгоритм* используются как синонимы, под алгоритмом подразумевается алгоритм в смысле Маркова [1].

Для обозначения составляющей программы на каком-либо формальном (не псевдо-код) языке программирования используется слово *инструкция*, это позволяет избежать путаницы возникающей при использовании термина оператор, широко распространенного в русскоязычной литературе. Слово оператор в данном тексте, кроме специально оговорённых случаев, обозначает знак операции (напр. в выражении „ $2 + 2$ “, „+“ — оператор, обозначающий необходимость выполнить сложение.)

Слова *разработчик* и *пользователь* обозначают, того кто что-то (в зависимости от контекста) делает и того для кого это делается, т. е. того, кто пользуется продуктом деятельности разработчика. Подразумевается что удобство пользователя — главная задача разработчика (даже если это одно лицо).

В тексте в основном используется язык программирования *C* (*C*) [2], но знание этого языка не обязательно.

Слова *ЭВМ* и *компьютер* в тексте обозначают одно и то же. Слово ПК если используется подразумевает IBM PC совместимые компьютеры на базе микропроцессоров совместимых с Intel 8086 с установленной ОС (Операционной системой), совместимой с Microsoft DOS.

Слова программа и приложения обозначают в основном одно и то же. Под приложением понимается признанная полезность программы для какой-либо деятельности, в то время как просто программа может, вывести на экран Hello World и умереть, если вы не слишком одиноки и сентиментальны, то вряд ли это можно признать полезной деятельностью.

Оглавление

1	Вычисления	7
1.1	Пользователи, разработчики и „лестница” разработки	7
1.2	Пользовательские приложения	8
1.3	Память	8
1.4	Основные уровни разработки	8
1.5	Место операционная системы	9
2	История ОС UNIX	11
2.1	Рождение UNIX	11
2.2	UNIX выходит в свет	12
2.3	GNU	13
2.4	Linux	13
3	Основные концепции UNIX	15
3.1	Файлы	15
3.2	Процессы	15
3.3	Описывающие объекты. Закрытые типы данных.	16
4	Файлы	21
4.1	Имена файлов	21
4.2	Информация о файле	22
4.3	Операции над файлами	22
4.3.1	Обращение к файлам со стороны процессов	22
4.3.2	Типы файлов	24
4.3.3	Права доступа к файлам	25
4.4	Работа с файловой системой.	28

Введение

Системе UNIX посвящено множество книг и статей. Данная книга не призвана заменить какую-нибудь из них. Книга написана, чтобы заполнить пробелы в знаниях людей уже знакомых с системой, которые захотят узнать некоторые подробности. Также книга призвана заполнить пробел, образовавшийся среди русскоязычной литературы. Я старался дать представление о непонятных терминах, которые часто встречаются в документации.

Здесь так же содержится абстрактное введение в операционные системы вообще, которое возможно будет полезно, людям очень слабо знакомым с компьютером.

В общем, в отличие от других русскоязычных книг, предметом данной является функционирование ядра (ядер) UNIX систем. Я стараюсь всегда отвечать не только на вопрос как, но и почему, также я стараюсь сам придумать альтернативы решениям принятым в UNIX и обсудить сравнительные достоинства и недостатки (я стараюсь, но на это не всегда хватает времени и сил).

Читателям незнакомым с системой UNIX книга может быть полезной для понимания работы какой-либо другой системы, т. к. большинство современных операционных систем унаследовало множество особенностей UNIX.

Несмотря на широкое использование, ОС UNIX устарела — я пытаюсь уделить внимание тем вещам которые и в будущем останутся прежними. Так же я стараюсь проследить тенденции развития новых ОС, таких как непосредственный наследник UNIX'а ОС Plan 9.

...

Глава 1

Вычисления

Для чего нужны ЭВМ? ЭВМ — электронные устройства для выполнения вычислений. ЭВМ должны выполнять определённое множество (очень большое) алгоритмов, определяемое заранее известными параметрами машины (объём памяти, быстродействие, срок эксплуатации, экономические факторы). Выбор (ввод) алгоритма должен выполняться максимально удобно для пользователя. Для каждого алгоритма существуют строго определённые множества входных данных и выходных, причем алгоритм должен быть определён, т. е. для одних и тех же входных данных производить одинаковые выходные.

1.1 Пользователи, разработчики и „лестница” разработки

В программировании используется принцип декомпозиции, т. е. алгоритм разбивается на конечное число (определяемое параметрами машины) стадий (промежуточных вычислений), а промежуточные вычисления разбиваются на уровни. Каждый разработчик отвечает за свой уровень, т. е. он разбивает программируемое вычисление на промежуточные, следующего, более низкого уровня.

Если, допустим, вы работаете на уровне k , то человек отвечающий за $(k+1)$ -й уровень является для вас пользователем, а отвечающий за $(k-1)$ -й уровень является разработчиком для вас. Причём вы не обязаны задумываться о существовании или контактировать с человеком на $(k+n)$ -ом, где $|n| \geq 2$ уровне. Такие контакты могут считаться дурным тоном и вообще вредной деятельностью — другому человеку лучше знать о том что надо на соседнем с ним уровне чем вам, а вам надо лучше удовлетворять потребности ваших соседей. На одном уровне могут работать несколько человек, т. е. для них разработчики и пользователи одни и те же, в таком случае они разделяют работу по какому-нибудь другому признаку (напр. по жребью, хотя обычно деление довольно осмысленное).

Последовательность уровней и принадлежность людей каждому из них мы назовём „лестницей” разработки.

1.2 Пользовательские приложения

Мною было продемонстрировано, что понятия пользователь и разработчик могут приводиться только вместе в понятном контексте, т. к. эти понятия относительны. Но в литературе можно встретить упоминание о пользовательских приложениях, что же это такое. Имеется в виду те программы, которые используют люди, решающие с помощью ЭВМ задачи для внешнего, по отношению к данной машине, мира (Дизайн, сочинительство, проектирование (материальных объектов), развлечения и т. п.). Уровень на лестнице разработки на котором находятся эти люди называется пользовательским уровнем. Именно эти люди оплачивают труд и затраты разработчиков, более низких уровней.

1.3 Память

При определении алгоритма мы упомянули о входных и выходных данных, все вычисления производимые на компьютере получают входные данные из памяти и пишут выходные данные в память. При этом говорится что в памяти храниться информация.

Существует разделение памяти по способу использования, быстродействию и физическому устройству. Память для ЭВМ обладает тем свойством, что чем больше размер памяти, тем она медленнее, а чем она быстрее, тем дороже её производство. Следует упомянуть следующие виды памяти:

оперативная память , достаточно быстрая память объёмом порядка сотен мегабайт.

память на магнитных дисках , очень медленная по сравнению с оперативной памятью, размеры порядка десятков гигабайт.

Отличительной особенностью оперативной памяти является то, что программы описанные машинным языком могут выполняться только если они находятся в оперативной памяти.

1.4 Основные уровни разработки

Давайте перечислим и разберём основные уровни на лестнице(дереве, если принять во внимание дальнейшее логическое деление уровней) разработки. Мы увидели, что каждый уровень определяется кругом пользователей и разработчиков, а также просто всеми теми, кто работает на данном уровне. Для каждого уровня мы приведём: название или описание данного уровня, примерный список людей, которые на нем работают, а так-же список

абстракций с которыми приходится иметь дело этим людям, в этом списке основное место уделяется единицам памяти к которым производится обращение.

Уровень	Люди работающие на этом уровне	Абстракции
Дизайн ЭВМ	Инженеры	триггеры, биты,
Программирование микропроцессоров	Системные программисты	Регистры проц
Программирование	Программисты, Системные программисты	Переменные, к
Уровень „продвинутых” пользователей	Системные администраторы, пользователи	программы, фа
Пользовательский уровень	Нормальные пользователи	пользовательск

1.5 Место операционная системы

Место ОС на этой лестнице, по мнению автора, находится между уровнем программистов и системных администраторов. Т. е. в задачу ОС входит максимальное упрощение системного администрирования, всего того, что должен сделать администратор, чтобы конечные пользователи чувствовали себя комфортно.

Глава 2

История ОС UNIX

2.1 Рождение UNIX

В 1965 году фирмы Bell Telephone Laboratories (часть корпорации AT&T) и General Electric совместно с Массачусетским технологическим университетом (MIT) приступили к разработке новой операционной системы, получившей название Multics. Перед системой были поставлены задачи — обеспечить одновременный доступ к ресурсам ЭВМ большого количества пользователей, обеспечить достаточную скорость вычислений, и хранение данных и дать возможность пользователям в случае необходимости совместно использовать данные. Хотя первая версия системы была запущена в 1965 году на ЭВМ GE 645, она не обеспечивала выполнение главных вычислительных задач, для решения которых она предназначалась. Фирма Bell Laboratories прекратила своё участие в проекте по причине экономической неэффективности.

Группа сотрудников исследовательской лаборатории Bell, во главе которых Кен Томпсон (Ken Thompson) и Деннис Ритчи (Dennis Ritchie), понимая что они не смогут использовать Multics, продолжили работать над системой, которая должна стать удобной средой для программирования. В одной из бесед был набросан проект файловой системы, этот проект затем был набран и отпечатан службой Белл, распечатка этого проекта стала рабочим документом по разработке UNIX.

Bell Laboratories отказались финансировать эту разработку, не смотря на то, что представленный Томпсоном проект должен был максимально снизить стоимость системы. Выходом из ситуации стало применение для разработки мало используемой ЭВМ PDP-7. Кеном Томпсоном было написано ядро и небольшой набор утилит. В начале применялся кросс-ассемблер для переноса системы с GE 645, но вскоре для системы был написан ассемблер, оболочка (shell) и система больше не нуждалась в поддержке со стороны другой машины.

Первым применением системы UNIX стало её использование в патент-

ном бюро Bell Labs для подготовки документации, с этой задачей система успешно справилась.

Томпсон написал систему на языке ассемблер но хотел переписать её на языке высокого уровня. В начале он хотел написать её на Фортране, но вместе с Деннисом Ритчи занялся языком Би (B) интерпретируемым языком произошедшем от языка BCPL. Для языка Би был написан компилятор с объявлениями переменных, процедурами и т. д. Новый вариант языка получил имя Си (C). Первая попытка переписать систему на языке Си не удалась. Около года Ритчи улучшал компилятор, добавил поддержку структур (записей), в итоге система была полностью переписана на этом языке.

В систему UNIX было введено несколько концепций, которые получили огромное признание у пользователей. В системе были использованы трубы (pipes), с помощью которых пользователь мог получать результат сложного действия, соединяя трубами простые задания, вскоре сотрудниками Белл был придуман способ записи, такой последовательности, используя знак |. Это нововведение породило новую философию в программировании, была рождена концепция инструментов — программ, которые должны выполнять одно конкретное действие и делать это хорошо. Так же в UNIX были стандартизованы руководства (manuals), с помощью которых пользователь может получить информацию о команде в стандартизованном виде.

2.2 UNIX выходит в свет

Кену Томпсону было поручено, провести обучение сотрудников Калифорнийского Университета Беркли, то чему он их учил была ОС UNIX. Университет Беркли получил грант DARPA, и таким образом UNIX заняла своё место у истоков того, что сейчас называют Internet. Университет Беркли предоставлял свою версию системы BSD, при этом лицензия принадлежала AT&T.

Позднее, так как система поставлялась в исходниках, и её было очень легко переносить на другие компьютеры, она приобрела огромную популярность, многие фирмы стали производить свою версию системы. В конце концов версий UNIX стало очень много и надо было подумать о совместимости. AT&T и Sun Microsystems объединили усилия для создания системы обладающей лучшими чертами других версий. В тоже время сообщество пользователей UNIX было обеспокоено этим шагом, т. к. Sun могла получить преимущество перед другими. В результате усилий двух сторон были получены две версии системы, которые теперь известны как System V и BSD соответственно. Сейчас существует стандарт POSIX, которому стараются следовать все производители UNIX-подобных систем.

2.3. *GNU*

13

2.3 GNU

...

2.4 Linux

...

Глава 3

Основные концепции UNIX

3.1 Файлы

Учитывая то, что основная деятельность пользователей — запуск приложений, с известными входными данными, система должна как-то организовать все доступные пользователю программы и их входные данные. Эта организация достигается за счёт концепции файлов. В файлах хранятся программы их входные и выходные данные. (Основная концепция UNIX, как мы далее увидим заключается в представлении всего к чему может захотеть обратиться пользователь в виде файлов).

3.2 Процессы

В большинстве компьютеров присутствует только один процессор (устройство для выполнения программ написанных на машинном языке), это влечёт за собой то что одновременно может выполняться только одна программа. Пользователю необходимо чтобы выполнялось несколько логически независимых задач (многие из которых — повседневное обслуживание запросов. Например организация очереди для вывода на принтер). Вероятно можно просто перемешать в оперативной памяти инструкции программ на машинном языке обеспечивающих выполнение необходимых задач, но такое решение очень сложно воплотить в жизнь, такой код очень сложно модифицировать, очень сложно изменить относительное время выполнения задач и т. д.

В системе UNIX машинные инструкции перемешиваются не в памяти машины, а во времени, таким образом каждая задача работает в отведённой для неё участке памяти, отдельные задачи легко загружать и выгружать из памяти. Разделение времени между задачами реализуется благодаря инструкциям перехода в машинном языке (типа `goto` в C). Программисты не должны заботиться о переключении между задачами, при написании программы для какой-то одной цели, в UNIX все выполняемые программы на

машинном языке (а следовательно и интерпретируемые программы) написаны так, как будто это единственная выполняемая программа на компьютере. Всю заботу о переключении между задачами берет на себя ОС — это реализуется по средствам аппаратных прерываний и поддержки виртуальной памяти со стороны ЭВМ. Аппаратные прерывания (прерывание от таймера) позволяет переключиться на выполнение кода ядра ОС, посреди работы приложения. Если ОС решит переключиться на другое приложение, она сохраняет необходимую информацию о состоянии процессора, когда же она переключается на приостановленное приложение она восстанавливает состояние процессора, так что приостановленная программа не замечает переключений. ОС также помнит информацию о занятой программой оперативной памяти, гарантируя сохранность информации на время переключения. Плюс к этому, UNIX хранит информацию об обращениях к файловой системе, так же гарантируя для них незаметность переключения.

Виртуальная память позволяет программе думать, что в оперативной памяти нет ни одной другой программы, а также защищать память от опасных ошибочных обращений. Всё это реализуется на аппаратном уровне.

Вся вышеперечисленная информация хранимая ОС для выполнения задачи, независимо от других задач, называется *процессом* и часто отождествляется с самой выполняемой задачей.

3.3 Описывающие объекты. Закрытые типы данных.

Существует некоторая особенность сознания человека, с которой приходится считаться в программировании. Можно проследить эту особенность на примере человеческого общения. Люди при своём взаимодействии используют последовательности звуков. Люди похожих культур умеют разбивать эти последовательности на подпоследовательности, называемые предложениями, словами, высказываниями и т. д. Допустим, что два человека используют похожие последовательности звуков, которые эти два человека называют словом „лошадь”. Дело в том, что говоря о лошади, каждый говорит о своём понимании этого слова, о своих воспоминаниях, впечатлениях, размышлениях, связанных с этим словом. У каждого эти размышления индивидуальны и неповторимы, следовательно и слово „лошадь” обозначает разные индивидуальные чувства. Удивительным фактом является то, что говоря о различных вещах люди изменяют оба понятия слова лошадь одновременно, понятие одного человека о лошади влияет на понятие другого человека и наоборот, при этом единственным пересечением этих понятий друг с другом является слово „лошадь” в окружении других слов. Таким образом каждый человек выделяет особую группу своих чувственных ощущений и мыслей (группу нейронных цепочек) и связывает её с новым объектом, словом, с помощью этого объекта (шевелия ртом и издавая странные звуки) люди коренным образом меняют нейронные цепи друг-друга. Груп-

3.3. ОПИСЫВАЮЩИЕ ОБЪЕКТЫ. ЗАКРЫТЫЕ ТИПЫ ДАННЫХ. 17

пу мыслей и чувственных ощущений мы назовём *понятием*, а какой-либо объект, благодаря которому понятие одного индивидуума ставится в соответствие понятию другого, мы назовём *описывающим объектом* (от англ. descriptor), такими объектами могут быть звуки, числа, знаки. Можно сказать, что для каждого понятия описывающим объектом является последовательность других понятий, каждое из которых описывается другими и т. д. В конце концов некоторые из понятий упираются в чувственные ощущения, связанные с этими понятиями. Т. е. ощущая что-либо человек непосредственно получает набор понятий, из них он получает другие более абстрактные и т. д. В общем механизм получения абстрактных понятий не известен, одной из причин является то, что любая мысль является сама по себе ощущением дающим новые понятия, которые одновременно являются новыми мыслями, дающими другие понятия и т. д. В этом проявляется дуализм сознания, являющегося одновременно и субъектом и объектом. Несмотря на окружающие эту тему философские проблемы принято считать, что чем большим слоем других понятий (являющихся описывающими объектами для данного) данное понятие отделено от органов чувств (от непосредственных чувственных ощущений), тем оно более абстрактное. И чем больше таких понятий, тем более абстрактное мышление было применено индивидуумом.

Понятия программиста отражаются в тексте программы на языке программирования, обычно через прослойку слов натурального языка, принятого при общении. В случае написания отдельных модулей каждый из них содержит замкнутое множество понятий. Части программы несмотря на разделение продолжают активно взаимодействовать, поэтому по мнению автора именно сужение множества понятий, отражённого в тексте модуля, и является главной причиной искусственного деления (вряд ли причина в скорости трансляции или уменьшении занимаемой памяти при редактировании). Поэтому выделяется обычно минимально возможное множество описывающих объектов для взаимодействия понятий из обоих модулей. В языках программирования существует несколько подходов к созданию описывающих объектов для общения между модулями.

Один из этих подходов — это использование тех же структур данных (того же взгляда на роль данных), что программист использовал при написании одного из модулей; в другом модуле другой набор понятий, и другой программист уже не может так же воспринимать эту структуру данных как первый (если это не так, значит наборы понятий в обоих модулях сильно пересекаются, а следовательно и преимущества деления на модули теряются). У программиста, использующего модуль с таким описывающим объектом, не остаётся другого выхода, как не смотреть на информацию о структуре данных (напр. забыть о именах полей в описании записи (структуры), представленной как описывающий объект). Не смотреть оказывается довольно просто, поэтому данный подход сильно распространён.

Другой подход это отказ от описания какой либо структуры данных связанной с понятием. Если оставить описание только той части структуры, которая воспринимается одинаково во всех модулях её использующих,

то мы ничего не проиграем и программисты смогут смело использовать всю информацию, имеющуюся в тексте программы. Другое дело, что выяснить одинаково ли два программиста понимают описание структуры данных очень сложно. А какое-либо пересечение понятий в модулях оставляет чувство незаконченности. Поэтому иногда, для того чтобы: программистов не нужно было заставлять не смотреть на описания, улучшить поиск ошибок в программе и не заниматься выяснением того, кто кого правильно понимает, — в качестве структуры описываемого объекта выбирается максимально простая безликая структура данных (обычно базовый тип данных языка программирования, напр. целое число). Такой тип данных каждый модуль связывает с присущим ему понятием, не встречая никаких противоречий со стороны описания типа — оно не опирается ни на какие понятия (кроме стандартизованных языком программирования). Каждый модуль хранит данные, связанные с этим понятием, и смотрит на эти данные в соответствии с набором своих внутренних понятий. Эти данные и понятия практически не пересекаются среди модулей. Тип данных (взгляд на данные со стороны модуля), соответствующий понятию модуля, к которому обращаются при помощи безликого описываемого объекта, называется закрытым типом данных.

У каждого модуля хранятся данные, соответствующие его взгляду на понятие. Поэтому перед таким модулем встают следующие задачи: выделить место для данных и поставить в соответствие данные, понятие и описывающий объект этого понятия. Обычно модуль предоставляет одно понятие (и следовательно полный набор операций с ним), другие модули определяют свои понятия через это понятие. Им не нужно что либо помнить об этом понятии, они знают как с ним оперировать, что бы на основе этого и других понятий построить новое, своё. Таким образом место под данные нужно выделить только одному модулю, поэтому ему можно самому выбрать описывающий объект и одновременно установить соответствие между описывающим объектом и данными. Обычно необходимые действия выполняются следующими операциями:

1. Создание или выделение — создание нового объекта-понятия и предоставление пользователю нового описываемого объекта;
2. Дублирование — предоставление нового описываемого объекта для, существующего объекта-понятия, пользователь получает возможность обращаться к одному и тому же понятию, используя два (или более, если дублирование проводилось раньше) различных объекта проводника.
3. Освобождение — заставляет модуль предоставляющий понятие забыть о данном описываемом объекте, пользователь больше не сможет обращаться к понятию при помощи этого описываемого объекта. Если это был единственный объект описывающий понятие, то пользователь больше не сможет обращаться к этому понятию и системе ничего не

3.3. ОПИСЫВАЮЩИЕ ОБЪЕКТЫ. ЗАКРЫТЫЕ ТИПЫ ДАННЫХ. 19

остаётся как освободить данные связанные с этим понятием (понятие же исчезает как только потерян последний описывающий его объект).

Глава 4

Файлы

4.1 Имена файлов

Файлы — это часть ресурсов, которыми распоряжается UNIX, имена файлов с этой точки зрения являются описывающими объектами.

Каждый файл обладает несколькими (хотя бы одним) полными именами, с помощью этих имён пользователь может обращаться к файлам. Для логической организации имён в реализации системы используются каталоги, это специальные файлы содержащие список имён других файлов. Специальный каталог называемый корневым содержит имена файлов и каталогов, если пройтись по всем ссылкам в корневом каталоге то мы посетим все файлы в файловой системе. Для безопасности в большинстве систем UNIX, каталог может обладать только одним именем, это предотвращает появление циклов в графе файловой системы. Таким образом файловая система образует дерево, листья которого — это ссылки на файлы-не-каталоги. На рис. 4.1 показана организация файловой системы.

Необходимость для файлов иметь несколько имён возникает из-за того, что на один и тот же файл можно смотреть с логически-разных точек зрения.

Полное имя файла определяется последовательностью имён каталогов, отделённых косой чертой '/'. Последняя часть полного имени называется просто именем файла.

Например: /usr/share/doc/bash/README

usr — ссылка в корневом каталоге на каталог usr.

share — ссылка в каталоге usr на каталог share с полным именем /usr/share

...

README — ссылка в каталоге bash на файл README с полным именем /usr/share/doc/bash/README

Рис. 4.1: title

Имена файлов состоят из символов алфавита принятого в системе, и обычно следуя соглашениям принятым в языке C, буквы нижнего и верхнего регистра различаются, т. е. README, readME, readme — три разных имени, вероятно трёх разных файлов.

4.2 Информация о файле

Каждому файлу кроме имён ставится в соответствие один единственный набор информации об этом файле. Эта информация хранится в файловой системе в структуре, называемой inode (индекс — не точный русский эквивалент). inode содержит тип файла (см. 4.3.2), права доступа к файлу (см. 4.3.3), время модификации файла, время создания файла и время изменения inode (т. е. информации о файле), а так же расположение файла на диске.

Современные системы поддерживают так называемую виртуальную файловую систему (VFS), т. е. ОС работает со стандартными, „виртуальными” индексами, а то как информация хранится на диске — дело драйверов файловых систем.

Таким образом для каждого файла не зависимо от имён (которые служат только логической организации файлов) содержится одна информация о том как с этим файлом работать.

Для каждого файла процесс может прочесть информацию о нем с помощью системного вызова `stat` или `fstat`, если это открытый файл (см. 4.3.1)

4.3 Операции над файлами

Пользователь должен уметь запускать программы, а программы должны уметь читать входные данные и записывать результат. В результате мы получаем 3 основные операции: `read` (читать), `write` (писать), `exec` (выполнять).

4.3.1 Обращение к файлам со стороны процессов

Рассмотрим программу на языке C, читающую файл:

```
#include <unistd.h>
#include <fcntl.h>

int main (int argc, char **argv)
{
    int fd;
    char buf[100];

    fd = open ("file.txt", O_RDONLY);
```

```
    read (fd, buf, 100);
    close (fd);
}
```

`fd` и `buf` — переменные в программе. `open`, `read` и `close` — стандартной библиотеки, которые просто вызывают (обычно по средствам программных прерываний) функции ядра ОС, носящие такие же названия (такие вызовы называются системными (`system call`)).

Системный вызов `open` регистрирует среди информации о процессе обращение к соответствующему файлу и записывает информацию необходимую для работы с файлом. Мы назовём это *сессией работы с файлом*. Результатом системного вызова является то, что процесс получает число называемое файловым дескриптором (`file descriptor`), это число идентифицирует обращение к данному файлу среди информации о процессе, т. е. является описывающим объектом для сессии.

Вызов `read` записывает 100 байт в место памяти `buf`, прочитанных из файла, на который ссылается дескриптор `fd`.

Вызов `close` сообщает системе что процесс больше не будет обращаться к файлу по данному дескриптору.

Каждая сессия работы с файлом содержит указатель на место чтения/записи, соответствующие операции производимые с данным обращением читают или записывают информацию в файле начиная с места определяемого этим указателем; операции также изменяют значение указателя, так чтобы последующие операции чтения или записи работали уже на новом месте в файле, с новыми данными.

Процессы работают не с именами файлов а с дескрипторами. Одна и та же сессия работы с файлом может иметь несколько дескрипторов на него ссылающихся, причем одну и ту же сессию могут использовать несколько процессов одновременно, у каждого из которых будет свой дескриптор ссылающийся на эту сессию. Если процессы читают файл одновременно, то каждый прочтет только часть этого файла, причем процессы не будут заранее знать какую часть. Один и тот же указатель на место чтения процессы будут сдвигать независимо друг от друга.

Для сессий работы с файлами у нас есть следующие системные вызовы:

- `open` — создание новой сессии и получение дескриптора для неё.
- `dup` — дублирование, мы получаем новый дескриптор для той же самой сессии.
- `close` — завершение работы с дескриптором. Если это единственный дескриптор ссылающийся на сессию, то она уничтожается, т. к. никто больше не работает с ней и никто не сможет получить доступ к этой сессии (у неё нет дескрипторов).

Для непосредственной работы с открытыми файлами используются следующие системные вызовы:

- **read** — читает данное число байт из файла с позиции содержащейся в сессии в пространство памяти пользовательского процесса и сдвигает указатель позиции чтения/записи на это число.
- **write** — записывает данные из пространства памяти пользовательского процесса в файл, начиная с места определяемого указателем чтения записи, поверх данных которые там были раньше. Если в файл не помещается часть записываемых данных, то файл расширяется до нужного размера.
- **lseek** — передвигает указатель чтения записи на новое место в файле.

4.3.2 Типы файлов

Действия, которые выполняются различными системными вызовами, а именно: `open`, `close`, `exec`, `read`, `write`, `lseek`, — зависят от типа файла. В UNIX эти типы заранее определены (в отличие от других систем, напр. Plan 9), вот они:

- Обычные файлы
- Каталоги
- Трубы
- Символьные ссылки (не во всех системах)
- Блочные устройства (со множеством подтипов, физических устройств)
- Символьные устройства (со множеством подтипов)
- Гнезда (Сокеты) (не во всех системах)

Обычные файлы При записи в обычные файлы информация сохраняется на диске и вы сможете её затем прочитать.

Каталоги Пользователи (даже `root`) не могут писать в каталог или запускать его на выполнение. Пользователи могут только читать каталоги. Система берет запись в каталоги на себя.

Трубы Информация пишется и читается из файла по принципу FIFO (первый зашёл, первый вышел). Т. е. если процессы решат обмениваться данными они могут договориться об имени трубы и передавать по ней данные.

Символьные ссылки Эти файлы не предназначены для какого либо использования, они призваны заменять дополнительные имена файлов, там где невозможно создать новое имя для файла (жесткую связь), для создания дополнительных имён каталогов, а также в некоторых других случаях. Символьные ссылки — файлы, содержащие путь к файлу на который

они указывают. Когда система получает имя файла содержащее имя символьной ссылке она просто подставляет вместо имени ссылки путь который в ней содержится.

Например в имени `/usr/doc/bash/README`, `/usr/doc` — символьная ссылка содержащая путь “`share/doc`”, поэтому при системных вызовах ядро подставляет в строку пути вместо `doc share/doc` и получает `/usr/share/doc/bash/README`.

Блочные устройства Блочные устройства предоставляют выполнять операции чтения и записи, определённые модулем встроенном в ядро, обычно это драйвер устройства. Блочные устройства отличаются тем, что загруженный модуль может выполнять чтение или запись только большими блоками и, обычно, довольно медленно. Ядро кеширует блоки с которыми работает модуль и при этом делает вид, что из данного файла можно производить посимвольное (по-байтовое) чтение. Когда пользователь читает данные определённой длины, ядро само находит в каких блоках находятся данные, читает их, если их нет в кеше, и возвращает пользователю, только необходимую часть блока.

Символьные устройства Модуль ядра сам реализует чтение и запись для таких устройств.

Гнезда (Сокеты) Файлы создаваемые при установлении TCP/UDP соединения, запись в такие файлы равносильна передаче данных используя соответствующий протокол.

4.3.3 Права доступа к файлам

В системе UNIX одновременно могут работать несколько (много) пользователей. Каждый процесс выполняет какую-либо одну пользовательскую задачу, но у разных пользователей могут быть различные задачи. Поэтому присутствие пользователя в системе характеризуется тем, что какой-либо процесс работает от имени этого пользователя. Кроме этого пользователь никак не связан с системой. У каждого процесса есть пользователь от лица которого он работает.

Для ядра системы пользователь это просто число, называемое UID (*user identifier*) или идентификатор пользователя. Для пользователей с некоторыми UID'ами могут быть связаны осмысленные, „человеческие” имена.

Было бы невозможно работать на ЭВМ, если бы пользователи могли изменять, смотреть данные друг друга без разрешения. Для этого в файловой системе хранится набор прав доступа к файлам (см. 4.2). В системе существует понятие группы, объединяющее пользователей обладающих одинаковыми правами.

Итак, для каждого файла в файловой системе хранится идентификатор пользователя — владельца файла, идентификатор группы которой принадлежит файл, и удостоверение права на выполнение 3-х операций: записи, чтения и выполнения, — для 3-х категорий пользователей: владельца, членов группы, и всех остальных, не попадающих в первые две категории.

Владелец файла — пользователь имеющий право изменять индекс файла, т. е. независимо от прав других пользователей только владелец может распоряжаться своими правами и раздавать права другим.

Группа файла — способ выделить группу пользователей, которые будут обладать другими (обычно большими правами, чем остальные). Так как в UNIX все ресурсы представлены (по крайней мере так задумано) в виде файлов, то права пользователя в системе определяются именно правами на доступ к файлам. Если пользователь хочет дать некоторым другим пользователям особые права для доступа к файлу (файлам), то он создаёт группу и устанавливает идентификатор группы файла равным идентификатору данной группы, устанавливает необходимые права доступа для группы. В итоге пользователи входящие в эту группу (а пользователь может входить во много групп) получают установленные права. Таким образом для того чтобы получить набор прав пользователь должен входить в нужную группу. Группы позволяют разъединить права пользователя на отдельные составляющие, а также делить права между пользователями. Возможно было бы задавать права для каждого пользователя в отдельности, но тогда пришлось бы модифицировать при необходимости права каждого, а при использовании групп достаточно изменить права доступа файла для группы и все пользователи входящие в группы сразу получают другие права. Также можно легко давать и лишать прав пользователя, достаточно включить, исключить его из группы.

Каждый процесс содержит идентификаторы пользователя и групп от имени которых он работает. Процесс содержит

- фактический идентификатор пользователя (effective UID)
- действительный идентификатор пользователя (real UID)
- сохраненный идентификатор пользователя (saved UID)

Аналогичные три идентификатора содержатся для группы, также содержится список групп в которые входит процесс. Нас интересуют только фактические идентификаторы, т. к. только они влияют на права доступа к файлам (о других мы поговорим в ??).

Для процесса обращающегося к файлу набор прав выбирается в следующей последовательности:

1. Если фактический идентификатор пользователя процесса совпадает с идентификатором пользователя файла (владельца), то выбирается набор прав для владельца этого файла.
2. Если фактический идентификатор группы процесса совпадает с идентификатором группы файла, то выбирается набор прав для группы.
3. Если идентификатор группы файла содержится в списке групп которым принадлежит процесс, то опять же выбирается набор прав для группы.

4. В остальных случаях выбирается набор прав для „остальных”.

Выделение фактического идентификатора группы из списка групп процесса связано с выполнением других операций о которых мы поговорим позже.

Такая последовательность выбора позволяет дать группе меньшие права, чем остальным, или даже лишить себя (владельца) части прав по отношению к остальным пользователям, правда эти возможности редко используются.

Мы знаем что для некоторых файлов мы не можем выполнить некоторые операции, например запись и выполнение для каталога, зачем же им все права доступа? Для каталога право на запись означает возможность создавать файлы в таком каталоге. А право на исполнение даёт возможность использовать каталог, как часть пути для доступа к файлу. Например пользователь может не иметь право читать каталог, но у него будет право на его исполнение, тогда пользователь не сможет прочитать какие файлы содержатся в каталоге, но сможет обратиться к файлу в каталоге имя которого он знает.

Существует специальный пользователь от лица которого системные администраторы выполняют критические для системы действия. Имя такого пользователя `root`, для него обычно создаётся группа с тем же именем. Особенность этого пользователя в том что он не связан правами доступа к файлам и может выполнять любые физически возможные действия. Обычные пользователи не должны работать от лица `root`'а, обычно у системного администратора есть пароль для входа в систему как `root`, администратор запускает некоторые системные программы от имени `root`'а чтобы внести изменения, касающиеся всех пользователей системы.

Права доступа к файлу обычно записываются в одну строчку. Такую информацию выдаёт команда `ls`:

```
drwxr-xr-x  13 root    root          312 Ноя  8 15:20 usr
```

первая строка символов — это тип и права доступа к файлу, `d` — тип файла, в данном случае каталог (`directory`), следующие три символа (`rw``x`) — права доступа для владельца файла (ему разрешены чтение (`r`, `Reading`), исполнение (`x`, `eXecution`) и запись (`w`, `Writing`)). Следующая тройка символов для группы файла, пользователи из группы могут читать и писать в файл, на месте символа `w` стоит прочерк, означающий запрет на запись. Аналогично для всех остальных пользователей. Число тринадцать — количество ссылок на файл, т. е. количество имён файла в файловой системе плюс количество процессов, которые работают с этим файлом. Первое слово `root` — имя владельца файла, второе — имя группы файла, дальше идёт дата модификации и имя файла.

Изменение прав доступа

Процессы могут изменять права доступа к файлу по своему усмотрению, если они являются владельцами файла. Насколько я знаю только `root`, может изменить владельца файла, таким образом большинство владельцев

файлов являются их создателями. Процесс может изменить группу файла, если это одна из групп в которые входит этот процесс (т. е. это фактический идентификатор группы процесса или один из идентификаторов содержащимся в списке групп процесса). Для изменения группы и/или владельца файла используется системный вызов `chown`. Для изменения прав доступа используется вызов `chmod`.

4.4 Работа с файловой системой.

Литература

[1] Марков, *Книга*.

[2] Керниган и Ритчи *Язык программирования Си*.