Содержание

1 Обзор ОС UNIX: архитектура, вход в систему, файлы и каталоги, ввод и вывод.

1.1 Архитектура UNIX

1.1.1 Определение ОС

Операционная система (ОС) – это программное обеспечение (ПО), которое управляет аппаратными ресурсами компьютера и предоставлет среду выполнения прикладных программ (*application*). Обычно это ПО называют ядром (kernel), т.к. оно имеет относительно небольшой объем и состовляет основу системы (см. рисунок ниже).

1.1.2 Интерфес ядра

Интерфес ядра – это слой ПО, называемый системными вызовами (*system calls*). Библиотеки функций общего пользования (*library routines*) строятся на основе интерфеса системных вызовов, но прикладная программа (*application*) может свободно пользоваться как теми, так и другими. Т.е. прикладная программа может использовать как системные вызовы, так и библиотечные функции.

1.1.3 Коммандая оболочка

Коммандая оболочка (*shell*) – это особое приложение, которое предоставлет интерфес для запуска других приложений.

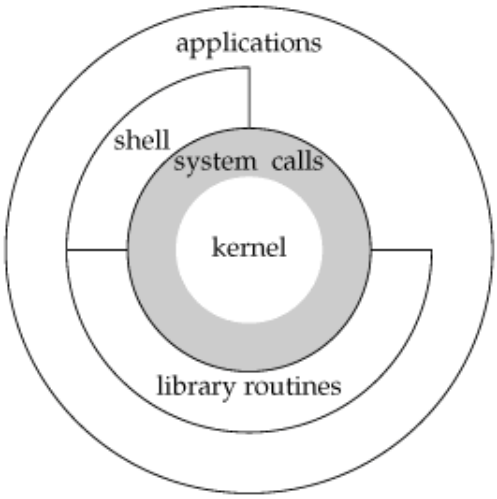
1.1.4 Обобщение

В общем, ОС – это ядро и всё остальное ПО, которое делает компьютера пригодным к использованию. В состов этого ПО входят системные утилиты (например ls, cat), прикладные программы (например GIMP), коммандые оболочки (например bash, fish), библиотеки функций общего пользования (например stdio для Си) и т.п..

1.1.5 Linux

Linux – ядро ОС GNU (или же GNU/Linux). Отдельно Linux – это ядро, называть его ОС не совсем корректно.

1.1.6 Схема, отражающая структуру UNIX



1.2 Вход в систему

При входе в систему UNIX мы вводим имя пользователя и пароль. Послеэ того система отыскивает введенное имя в файле паролей; обычно это файл `/etc/passwd`. Файл паролей содержит записи, каждая из которых состоит из семи полей, разделенных двоеточиями: имя пользователя, зашифрованный пароль, числовой идентификатор пользователя (`205`), числовой идентификатор группы (`105`), поле комментария, домашний каталог (`/home/sar`) икомандный интерпретатор (`/bin/ksh`).
Пример записи:
`sar:x:205:105:Stephen Rago:/home/sar:/bin/ksh`
Все современные системы хранят пароли в отдельном файле.
После входа в систему пользователь получает доступ к коммандной оболочке (например к bash или sh).

1.3 Файлы и каталоги

1.3.1 Файловая система

- Файловая система UNIX представляет собой иерархическую древовидную структуру, состоящую из каталогов и файлов. Начинается она с каталога, который называется корнем (`root`), а имя этого каталога представлено единственным символом - `/`.

- Каталог представляет собой файл, в котором содержатся каталожные записи. Логически каждую такую запись можно представить в виде струк-

туры,состоящей из имени файла и дополнительной информации, описывающей атрибуты файла.

- Атрибуты файла - это такие характеристики, как тип файла (обычный файл или каталог), размер файла, владелец файла, права доступа к файлу (есть ли у других пользователей доступ к файлу), время последней модификации файла.
  Для получения атрибутов файла используются функции `stat()` и `fstat()`.

1.3.2 Имя файла

Имена элементов каталога называются именами файлов. Имя файла не может содержать слеш `/` или нулевой символ `0`. Файл «точка» `.` – это текущий каталог, файл «точка-точка» `..` – родительский. При создании нового каталога в нём создаются файлы «точка» и «точка-точка». Для корневого каталога «точка-точка» – это то же самое что и «точка».
Современные ОС позволяют создавать файлы с длинной имени не менее 255 символов.

1.3.3 Путь к файлу

Последовательность одного или более имен файлов (каталог – это тоже файл), разделенных слэшами, образует строку пути к файлу. Эта строка может также начинаться с символа слэша, и тогда она называется строкой **абсолютного пути**, в противномслучае – строкой **относительного пути**. В случае относительного пути маршрут начинается от текущего каталога.
Также в пути могут присутсвовать «точка» и «точка-точка», например путь `../qwe` – это файл `qwe` в родительском каталоге.

1.3.4 Рабочий каталог

У каждого процесса имеется свой рабочий каталог, который иногда называют текущим рабочим каталогом. Это каталог, от которого отсчитываются все относительные пути, используемые в программе. Процесс может изменить свой рабочий каталог с помощью функции `chdir`.

1.3.5 Домашний каталог

Когда пользователь входит в систему, рабочим каталогом становится его домашний каталог. Домашний каталог пользователя определяется всоответствии с записью в файле паролей.

1.4 Ввод и вывод

1.4.1 Дескрипторы файлов

Дескрипторы файлов - это, как правило, небольшие целые положительные числа, используемые ядром для идентификации файлов, к которым обращается конкретный процесс. Всякий раз, когда процесс открывает существующий или создает новый файл, ядро возвращает его дескриптор, который затем используется для выполнения над файлом операций чтения или записи.

1.4.2 Стандартный ввод, стандартный вывод,Стандартный вывод сообщений об ошибках

По принятым соглашениям все командные оболочки при запуске новой программы открывают для нее три файловых дескриптора: файл стандартного ввода (`stdin`), файл стандартного вывода (`stdout`) и файл стандартного вывода сообщений об ошибках (`stderr`).
`ls > file.out` – перенаправление вывода `stdout` в файл `file.out`
`ls 2> file.err` – перенаправление вывода ошибок `stderr` в файл `file.err`

1.4.3 Небуферизованный ввод-вывод

Небуферизованный ввод-вывод осуществляется функциями `open` , `read` , `write` , `lseek` и `close` . Все эти функции работают с файловыми дескрипторами.
Заголовочный файл `<unistd. h>` , подключаемый из файла `apue.h` , и константы `STDIN_FILENO` и `STDOUT_FILENO` являются частью стандарта **POSIX.**
Константы `STDIN_FILENO` и `STDOUT_FILENO` , определенные в файле `<unistd. h>` ,устанавливают дескрипторы файлов стандартного ввода и стандартного вывода.Обычные значения этих констант - соответственно 0 и 1.

1.4.4 Стандартные функции ввода-вывода

Стандартные функции ввода-вывода предоставляют буферизованный интерфейс к функциям небуферизованного ввода-вывода. Использование стандартных функций ввода-вывода избавляет нас от необходимости задумываться о выборе оптимального размера буфера.
Другое преимущество стандартных функций ввода-вывода в том, что они значительно упрощают обработку пользовательского ввода (например `fgets()` или `printf()`).
`<stdio.h>` содержит прототипы всех стандартных функций ввода-вывода.

2 Обзор ОС UNIX: программы и процессы, обработка ошибок, идентификация пользователя.

2.1 Программы и процессы

2.1.1 Программа

Программа – это исполняемый файл, размещенный на диске. Программа считывается в память и затем выполняется ядром через вызов одной из шести функций семейства `exec`.

2.1.2 Процессы и идентификаторы процессов

Программа, находящаяся в процессе исполнения, называется процессом.
В некоторых ОС для обозначения выполняемой в данный момент программы используется термин задача.
UNIX обеспечивает присвоение каждому процессу уникального числового идентификатора, который называется идентификатором процесса. Идентификатор процесса – всегда целое неотрицательное число.

2.1.3 getpid()

Получить собственный идентификатор процесса можно при помощи функции `getpid()` (get process id)

2.1.4 Управление процессами

Три основные функции отвечают за управление процессами:

- `fork()` – создает копию вызывающего процесса
- `exec()` – замещает дочерний процесс некоторой программой. Функция `exec` имеет шесть разновидностей
- `waitpid()` – ждёт завершения некоторого процесса с определенным `pid`

2.2 Обработка ошибок

2.2.1 errno

Очень часто при возникновении ошибки в любой из функций системы UNIX эта функция возвращает отрицательное число, а в глобальную переменную `errno` записывается некоторое целое, которое несёт дополнительную информацию о возникшей ошибке.

Например, функция open возвращает либо файловый дескриптор – неотрицательное число, либо `-1` в случае возникновения ошибки. Некоторые функции следуют иному соглашению. Например, большинство функций, которые должны возвращать указатель на какой-либо объект, в случае ошибки возвращают пустой указатель (`NULL`).
Определения переменной `errno` и констант всех возможных кодов ошибок находятся в заголовочном файле `<errno.h>`. Имена констант начинаются с символа E (например `EACCES` – возникли проблемы с правами доступа, например при открытии файла).
Стандарты POSIX и ISO C определяют `errno` как символ, раскрывающийсяв изменяемое выражение `lvalue` (то есть выражение, которое может стоять слева от оператора присваивания) целого типа.

2.2.2 Вывод сообщения об ошибке

Для вывода сообщений об ошибках стандарт C предусматривает две функции:

- `char *strerror(int errnum);` – преобразует `errnum` (равный `errno`) в строку сообщения об ошибке и возвращает указатель на нее
- `void perror(const char *msg);` – на основе `errno` выводит сообщение об ошибке с префиксом `msg:`, т.е. `msg: error_text`. Вывод заканчивается символом перевода строки.

2.2.3 Восстановление после ошибок

Ошибки, определенные в `<errno.h>`, могут быть разделены на две категории - фатальные и нефатальные. Восстановление нормальной работы после фатальных ошибок невозможно. Самое лучшее, что мы можем сделать, – это вывести сообщение об ошибке на экран или записать его в файл журнала и завершить работу приложения. Нефатальные ошибки допускают нормальное продолжение работы. Большинство нефатальных ошибок по своей природе носят временный характер (например, нехватка ресурсов), и их можно избежать при меньшей загруженности системы.

2.3 Идентификация пользователя

2.3.1 Идентификатор пользователя

Идентификатор пользователя из записи в файле паролей представляет собой числовое значение, которое однозначно идентифицирует пользователя в системе. Идентификатор пользователя назначается системным администратором при создании учетной записи и не может быть изменен пользователем. Как правило,

каждому пользователю назначается уникальный идентификатор. Ядро использует идентификаторпользователя для проверки прав на выполнение определенных операций.
Пользователь с идентификатором `0` называется суперпользователем, или `root`.
В файле паролей этому пользователю обычно присвоено имя `root`.
Если процесс имеетпривилегии суперпользователя,большинство проверок прав доступа к файлам просто не выполняется. Некоторые системные операции доступны только суперпользователю. Суперпользователь обладает неограниченной свободой действий в системе.

2.3.2 Идентификатор группы

Кроме всего прочего, запись в файле паролей содержит числовой идентификатор группы. Он также назначается системным администратором при создании учетной записи.
Обычно группы используются для распределения пользователей по проектам или отделам. Это позволяет организовать совместное использование ресурсов, например файлов, членами определенной группы.
В системе существует файл групп, в котором указаны соответствия имен группих числовым идентификаторам. Обычно этот файл называется `/etc/group`.
Каждый пользователь может состоять в нескольких группах.

3 Обзор ОС UNIX: сигналы, представление времени, системные вызовы и библиотечные функции.

3.1 Сигналы

Сигналы используются, чтобы известить процесс о наступлении некоторого состояния. Например, если процесс попытается выполнить деление на ноль,он получит уведомление в виде сигнала `SIGFPE` (floating-point exception – ошибка выполнения операции с плавающей точкой). Процесс может реагировать на сигнал тремя способами:

- Игнорировать сигнал
- Разрешить выполнение действия по умолчанию
- Определить функцию, которая будет вызвана для обработки сигнала (такие функции называют перехватчиками (handler) сигналов).

`Ctrl+c` – генерирует сигнал прерывания.
При помощи функции `kill` процесс может послать сигнал другому процессу.

При этом процесс который посылает сигнал должен быть владельцем процесса которому посылается сигнал.

3.2 Представление времени

Исторически в системе UNIX поддерживается два различных способа представления временных интервалов:

- Календарное время. Значения в этом представлении хранят число секунд, прошедших с начала Эпохи: 00:00:00 1 января 1970 года по согласованному всемирному времени (Coordinated Universal Time - UTC)
- Время работы процесса. Оно еще называется процессорным временем и измеряет ресурсы центрального процессора, использованные процессом.Значения в этом представлении измеряются в тактах (ticks). Исторически сложилось так, что в различных системах в одной секунде может быть 50, 60 или 100 тактов. Для хранения времени в этом представлении используется тип данных `clock_t`.

При измерении времени выполнения процессасистема UNIX хранит три значения для каждого процесса:

- Общее время (**Clock time**) – это отрезок времени, затраченный процессом от момента запуска до завершения
- Пользовательское время (**User CPU time**) – это время, затраченное на исполнение машинных инструкций самой программы
- Системное время (**System CPU time**) – это время, затраченное на выполнение ядром машинных инструкций от имени процесса

Сумму пользовательского и системного времени часто называют процессорным временем.

3.3 Системные вызовы и библиотечные функции

Любая операционная система обеспечивает прикладным программам возможность обращения к системным службам. Во всех реализациях UNIX имеется строго определенное число точек входа в ядро, которые называются системными вызовами (system calls, см. картинку в начале).
В ОС Linux имеется от 240 до 260 системных вызовов в зависимости от версии.
В ОС FreeBSD около 320 системных вызовов.
В системе UNIX для каждого системного вызова предусматривается одноименная функция в стандартной библиотеке языка C. Пользовательский процесс вызывает эту функцию стандартными средствами языка C.

С точки зрения разработчика системы между системным вызовом и библиотечной функцией имеются коренные различия. Но с точки зрения пользователя эти различия носят непринципиальный характер. В контексте нашей книги и системные вызовы, и библиотечные функции можно представлять как обычные функции языка С.

4 Стандарты и реализации ОС UNIX: пределы ISO C, пределы POSIX, функции sysconf(), pathconf() и fpathconf(), элементарные системные типы данных.

5 Файловый ввод-вывод: дескрипторы файлов, функция open(), функция creat(), функция close().

6 Файловый ввод-вывод: Функция lseek(), функция read(), функция write()/

7 Файловый ввод-вывод: эффективность операций ввода-вывода

8 Файловый ввод-вывод: совместное использование файлов, атомарные операции, функции dup() и dup2()

9 Файловый ввод-вывод: функции sync(), fsync(), fdatasync(), fcntl(), ioctl(), /dev/fd

10 Файлы и каталоги: функции stat(), fstat(), lstat(), содержимое struct stat.

11 Файлы и каталоги: типы файлов, права доступа к файлу, функция umask().

12 Файлы и каталоги: функции chmod(), fchmod(), chown(), fchown(), lchown().

13 Файлы и каталоги: размер файла, дырки в файлах, усечение файлов, файловые системы, функции link(), unlink(), remove(), rename().

14 Файлы и каталоги: символические ссылки, функции symlink() и readlink().

15 Файлы и каталоги: временные характеристики файлов, функция utime().

16 Файлы и каталоги: функции mkdir() и rmdir(), чтение каталогов, функции chdir(), fchdir(), getcwd().

17 Стандартная библиотека ввода-вывода: потоки и объекты FILE, стандартные потоки ввода, вывода и сообщений об ошибках, буферизация.

18 Стандартная библиотека ввода-вывода: открытие потока, чтение из потока и запись в поток, функции ввода, функции вывода.

19 Стандартная библиотека ввода-вывода: эффективность стандартных операций ввода-вывода, позиционирование в потоке.

20 Стандартная библиотека ввода-вывода: форматированный вывод, форматированный ввод, временные файлы.

21 Управление процессами: идентификаторы процесса, функция fork(), совместное использование файлов.

22 Управление процессами: функция exit(), функции wait() и waitpid().

23 Управление процессами: семейство функций exec().

24 Управление процессами: изменение идентификаторов пользователя и группы, функции setuid(), setgid(), seteuid(), setegid().