УПРАВЛЕНИЕ ТЕРМИНАЛЬНЫМ ВВОДОМ/ВЫВОДОМ

Обзор

Этот раздел обсуждает основы интерфейса для управления асинхронными коммуникационными портами (терминальными портами). Функции, перечисленные на странице руководства TERMIOS(2) используются для доступа и конфигурации аппаратного интерфейса с терминалом. Эти функции и их аргументы будут обсуждаться в этом разделе. Первая секция этого раздела предоставляет информацию, необходимую для понимания характеристик терминала и принципов работы аппаратного и программного терминального интерфейса. Затем будут обсуждаться некоторые аспекты программного интерфейса с терминалом. Приводятся примеры использования функций termios(2) для изменения этих установок.

Характеристики терминального интерфейса

Приблизительно до конца 80х-начала 90х, терминалы были основным средством организации взаимодействия человека с компьютером. Терминал (дословно — оконечное устройство) представляет собой электронную пишущую машинку (телетайп) или устройство, состоящее из клавиатуры и дисплея (видеотерминал). Оба типа терминалов соединены с компьютером последовательным портом (обычно, RS232 или токовая петля); при этом символы, вводимые с клавиатуры, передаются компьютеру, а данные, передаваемые компьютером, показываются на дисплее (в случае видеотерминала) или печатаются на бумаге (в случае телетайпа). Как телетайпы, так и видеотерминалы предназначены для ввода и отображения текстовой информации. С точки зрения компьютера, терминальный порт представляет собой двунаправленный (полнодуплексный) последовательный порт, по которому производится обмен символами кодировки ASCII или национальной кодировки, такой, как КОИ8.

Кроме ASCII, большинство видеотерминалов могут передавать и принимать коды расширения (escape sequence). Обычно это многобайтовые коды, начинающиеся с символа '\0x1B' (ASCII ESC), обозначающие нажатия клавиш, для которых нет соответствующих кодов в ASCII (стрелки, «функциональные» клавиши и т. д.), а также команды терминалу: передвижение курсора, изменения цвета текста и т.д..

Так, на многих видеотерминалах, последовательность символов "\0x1B[A" обозначает нажатие клавиши «стрелка вверх» на клавиатуре, а также команду на перемещение курсора на одну строку вверх.

Поскольку терминалы были основным средством взаимодействия человека с компьютером, в системах семейства Unix в драйвер терминала был встроен ряд функций, не сводящихся к простой передаче данных через порт. Для управления всеми этими функциями, терминальные устройства поддерживали набор специальных команд ioctl(2). Среди этих функций следует упомянуть:

Редактирование ввода: стирание последнего введенного символа, последнего слова и всей строки.

Преобразование ввода: преобразование символов конца строки, замена табуляций на пробелы, и др.

Генерация сигналов: при вводе определенных символов, ядро посылает группе процессов первого плана сигналы.

Поддержка терминальных сессий и управления заданиями.

Было разработано множество программ, рассчитанных на работу с терминалами: экранные текстовые редакторы, интегрированные среды разработки, почтовые клиенты, клиенты gopher, веббраузеры (lynx и links), файловые менеджеры, игры и др. Командные процессоры с управлением заданиями (ksh(1), jsh(1), bash(1)) использовали поддержку со стороны терминала (фоновые группы и группы первого плана, а также сигналы управления заданиями). Кроме того, многие программы, такие как su, sudo, login, использовали некоторые простые терминальные функции, такие, как включение и выключение «эхо» (отображения вводимых пользователем символов). Действительно, при наборе команд, пользователю необходимо видеть на экране набираемые им символы, а при вводе пароля это может быть нежелательно. Поэтому утилиты, требующие ввода пароля, выключают отображение ввода, а потом включают его обратно.

Интерфейс ввода/вывода

Диаграмма показывает отношения между пользовательской программой, ядром системы и портом ввода/вывода при работе с физическим терминалом.

Физические терминалы присоединяются к компьютеру через аппаратный интерфейс ввода/вывода, называемый портом. Порты, сделанные различными изготовителями, устроены по разному, но поддерживают стандартные протоколы обмена, обычно стандарт RS232. В IBM PC--совместимых компьютерах терминальные порты RS232 называются COM-порты. Эти порты обычно конфигурируются или программируются, так что они могут работать с различными терминалами и на различных скоростях.

Пользовательские программы не имеют прямого доступа к портам ввода/вывода. Программный интерфейс с портами предоставляется драйвером устройства. Драйвер устройства — это модуль ядра Unix, который предоставляет набор аппаратно-зависимых функций, которые, в свою очередь, управляют портами и осуществляют доступ к ним. Доступ к драйверу происходит через специальный байт-ориентированный файл терминала. Имена этих файлов находятся в директории, обычно /dev или /dev/term, и доступ к ним может осуществляться так же, как к обычным файлам. Ввод и вывод на терминал осуществляется чтением и записью в соответствующий специальный файл. Многие программы не видят разницы между терминалом и обычным файлом, что позволяет перенаправлять ввод-вывод таких программ на терминал или в файл в зависимости от потребностей пользователя.

В Unix SVR4, терминальные драйверы сами имеют модульную структуру и состоят из нескольких модулей STREAMS, STREAMS — это появившийся в Unix SVR3 асинхронный интерфейс для драйверов последовательных (байт-ориентированных) устройств. API для разработки драйверов в Solaris описано в секции руководства 9 и не будет подробно обсуждаться в этом курсе.

Терминальный драйвер STREAMS состоит из, как минимум, двух модулей: собственно драйвера порта, который обеспечивает прием и передачу данных в физический порт, и модуля терминальной дисциплины ldterm(7m), который и отвечает за специфически терминальные функции и обработку терминальных команд ioctl(2).

Псевдотерминалы

При переходе к графическим дисплеям, возник вопрос, как обеспечить совместимость с программами, ориентированными на работу с терминалом. Для этого в ядре Unix предусмотрен интерфейс для создания специальных псевдоустройств, которые так и называются псевдотерминалами. Псевдоустройство или виртуальное устройство — это виртуальный объект, обслуживаемый специальным драйвером. Такой драйвер поддерживает такие же программные интерфейсы и структуры данных (минорную запись, специальный файл в каталоге /dev), как и обычный драйвер, но, в отличие от обычного драйвера, драйвер псевдоустройства не связан ни с каким физическим устройством, а имитирует все функции устройства программно. Примерами псевдоустройств являются файлы /dev/null, /dev/zero, /dev/random (в Linux). Псевдотерминалы создаются для поддержки сессий удаленного доступа через telnet(1), rlogin/rsh(1) и ssh(1), а также для терминальных сессий xterm(1) и gnome-terminal(1).

Псевдотерминал состоит из двух псевдоустройств, ведущего (/dev/pts/XX) и ведомого (/dev/pty/XX, где X — десятичная цифра). Процедура создания и открытия этих устройств различается в разных Unix-системах и подробно не рассматривается в этом курсе. Соответствующая процедура для Solaris описана на странице руководства pts(7D). Оба устройства, в действительности, обслуживаются одним и тем же модулем ядра и представляют собой нечто вроде трубы: данные, записываемые в дескриптор ведущего устройства, читаются из ведомого, и наоборот. Кроме того, ведомое устройство поддерживает все команды ioctl(2), обязательные для терминала, и все терминальные функции, перечисленные на предыдущей странице.

Рассмотрим использование псевдотерминала терминальным эмулятором xterm(1). xterm(1) представляет собой графическое приложение, использующее протокол X Window. При запуске, xterm(1) создает и открывает окно на локальном или удаленном дисплее, имя которого определяется переменной среды DISPLAY. Это окно представляет собой текстовое окно, по умолчанию имеющее размер 80x25 символов, что соответствует стандартному размеру экрана большинства видеотерминалов. Кроме того, xterm(1) создает пару ведущего и ведомого устройств псевдотерминала, запускает подпроцесс, в этом подпроцессе создает сессию вызовом setsid(2), открывает ведомое устройство на дескрипторы 0, 1 и 2 (при этом, соответствующий псевдотерминал становится управляющим терминалом этой сессии), устанавливает переменную среды TERM=xterm и запускает в этом подпроцессе программу. По умолчанию, имя программы берется из переменной среды SHELL, но, если указать соответствующие параметры xterm(1), можно запустить произвольную программу.

Затем, xterm(1) преобразует нажимаемые пользователем клавиши в коды ASCII или текущей национальной кодировки (в наше время, обычно, UTF-8) или, если это необходимо, в коды расширения, и передает эти символы в ведущее устройство, так что они поступают на стандартный ввод запущенной программы или какого-то из её подпроцессов.. Кроме того, данные, выводимые запущенной программой в дескрипторы 1 и 2, считываются процессом xterm(1) из ведущего устройства и отображаются в текстовом окне так же, как они отображались бы на дисплее видеотерминала. При этом, передаваемые программой коды расширения интерпретируются программой xterm(1) как команды перемещения курсора, изменения цвета текста и т. д., поэтому программы, рассчитанные на работу с видеотерминалом, работают в привычной для них среде.

Программный интерфейс ввода/вывода

Многие из системных вызовов для работы с обычными файлами также используются и для работы с терминальными специальными файлами. Для доступа к терминалам можно использовать следующие системные вызовы:

open(2) Как и регулярные файлы, специальные байт-ориентированные файлы открываются этим системным вызовом. По соглашению, имена всех терминальный файлов находятся в директории /dev или одной из поддиректорий /dev. В Solaris они размещены в /dev/term/XX (физические терминалы) и /dev/pty/XX (псевдотерминалы), где XX — двузначное десятичное число. Кроме того, управляющий терминал вашей сессии доступен вашей программе как /dev/tty.

ioctl(2) Этот системный вызов используется передачи устройствам команд, которые не могут быть сведены к чтению или записи. У терминалов, ioctl(2) используется как для конфигурации физического порта ввода/вывода, так и для управления функциями терминальной дисциплины. Соответствующие команды ioctl(2) не стандартизованы, различаются в разных Unix-системах и не будут обсуждаться в этом курсе. Параметры ioctl(2) для работы с терминалами в Solaris, описаны на странице руководства termio(7I).

termios(3C) Эта страница руководства содержит набор функций, предоставляющих стандартизованный интерфейс для управления терминальными устройствами. Это более предпочтительный интерфейс, чем ioctl(2), потому что он соответствует стандарту POSIX и обеспечивает разработку переносимых программ. В этом разделе будут обсуждаться, главным образом, функции termios(3C).

isatty(3F) Этот системный вызов определяет, связан ли файловый дескриптор с терминальным устройством или с файлом какого-то другого типа. Если isatty(3F) возвращает ненулевое значение, файловый дескриптор поддерживает терминальные ioctl(2) и функции termios(3C).

read(2) Используется для чтения данных из специального терминального файла. read(2) возвращает количество прочитанных байтов, которое может быть меньше запрошенного. По умолчанию, терминал ожидает ввода полной строки, оканчивающейся символом '\n' (ASCII NL) и считывает данные по строкам. Однако не обязательно читать всю строку за один раз. Если буфер read(2) меньше длины текущей строки, будет считано только начало строки.

Чтение с терминала разрушает данные, то есть прочитанные данные не могут быть прочитаны опять. Поэтому если два процесса одновременно читают с терминала, это может приводить к потере данных. Для управления доступом к чтению с терминала используются сигналы управления заданиями и функция tcsetpgrp(3C), которые рассматриваются далее в этом разделе

write(2) Системный вызов write(2) используется для записи символов в специальный байт-ориентированный файл.

poll(2) и select(3C). Эти вызовы часто используются для мультиплексирования ввода-вывода, если процессу необходимо одновременно работать с терминалом и другими устройствами или псевдоустройствами, работа с которыми может привести к блокировке.

libcurses(3LIB) библиотека для генерации кодов расширения терминала в зависимости от его типа.

close(2) Системный вызов close(2) закрывает дескриптор файла, связанный со специальным файлом.

lseek(2), mmap(2) Терминальные устройства эти вызовы не поддерживают.

Библиотека libcurses(3LIB) и другие библиотеки

Видеотерминалы и терминальные эмуляторы поддерживают довольно богатые функции работы с текстом. Одной из главных функций является изменение положения курсора — точки, в которой будет выводиться текст на экран. Перемещая курсор, пользовательская программа может рисовать на экране почти всё, что можно изобразить при помощи символов ASCII: экранные формы, выпадающие меню, диалоговые окна, даже изображения (так называемый ASCII art). Кроме того, многие терминалы поддерживают расширенные символы, например, так называемую «псевдографику». Эти символы позволяю рисовать на экране прямоугольные рамки и таблицы. Многие видеотерминалы и практически все терминальные эмуляторы также поддерживают управление цветом символов и фона.

Доступ ко всем этим функциям осуществляется при помощи кодов расширения (escape sequences) — многобайтовых последовательностей, начинающихся с символа ASCII ESC. Разные модели терминалов поддерживают разные наборы кодов расширения. Существует несколько де-юре и де-факто стандартов этих кодов. Основной стандарт де-юре — это ECMA-48, известный также как ANSI X3.64 и ISO/IEC 6429, соответствующий набор кодов расширения также называется ANSI escape sequences. Наиболее известные стандарты де-факто — это управляющие коды терминалов DEC VT-50, DEC VT-100 и xterm. Набор кодов расширения VT-100 сильно перекрывается со стандартом ANSI и считается одним из первых аппаратных терминалов, реализовавших этот стандарт. Набор кодов xterm эмулирует как VT-100, так и ANSI, а также добавляет ряд функций, характерных для терминального эмулятора — так, код расширения ESC]2;stringBEL (где ESC — это ASCII ESC, а BEL — ASCII BEL) заменяет название окна xterm на string. В системном руководстве bash(1) содержится информация о том, как настроить строку приглашения bash, чтобы она содержала эту последовательность и формировала string так, чтобы эта строка, в свою очередь, содержала интересную для пользователя информацию, например, текущий каталог данной терминальной сессии, имя сетевого узла и т. д.

Так или иначе, многие терминалы, реализующие стандартные наборы команд, поддерживают их не полностью или добавляют какие-то свои расширения. Ядро Unix не предоставляет сервисов для работы с кодами расширения терминалов и передает эти коды терминалу «как есть». Для работы с кодами расширения необходимо использовать библиотеки, например, libcurses(3LIB), которая входит в поставку Solaris.

Библиотека libcurses обеспечивает формирование кодов расширения для выполнения заданных функций, например, для перемещения курсора в заданную точку экрана, а также интерпретацию кодов расширения, посылаемых терминалом, и их перевод в независимые от терминала псевдосимволы. Библиотека определяет тип терминала на основе переменной среды TERM, и использует базу данных, хранящуюся в каталоге /usr/share/lib/terminfo/. В этой базе для каждого известного типа терминала хранится таблица, описывающая поддерживаемые им типы команд и код расширения, соответствующий каждой команда. Libcurses обеспечивает некоторую оптимизацию: так, если требуется переместить курсор в заданную точку и терминал поддерживает команды для установки позиции курсора, библиотека сгенерирует соответствующую команду. Если же терминал такой команды не поддерживает, библиотека сгенерирует последовательность перемещений курсора на одну позицию вверх, вниз или вбок (такая функция есть почти у всех видеотерминалов).

Почти все «полноэкранные» программы, такие, как текстовые редакторы, веб-браузеры или файловые менеджеры, используют libcurses или какой-то из ее аналогов, чаще всего ncurses. Однако в нашем курсе мы изучать эту библиотеку не будем.

Канонический ввод

Терминал имеет две очереди ввода и одна - вывода. Символы, вводимые с клавиатуры терминала, помещаются в "сырую" очередь ввода. Кроме того, если требуется эхо (вывод печатаемых символов на экран), копии этих символов добавляются в очередь вывода.

Если разрешен канонический режим обработки ввода (обычно это делается по умолчанию), символы из "сырой" очереди подвергаются предобработке при копировании в каноническую очередь ввода. Копирование происходит по строкам, когда поступает символ перевода строки (NL). Пользовательская программа читает строки ввода из канонической очереди.

Каноническая предобработка ввода включает обработку символов забоя и стирания строки. Символ забоя ERASE (на видеотерминалах это обычно '\0x8' (Ctrl-H, ASCII BS) или '\0x7F' (ASCII DEL); на телетайпах часто использовался символ #) удаляется вместе с символом, введенным перед ним.

Клавиша «Backspace» на терминале может быть настроена посылать либо ASCII BS, либо ASCII DEL. Клавиша «Del» в xterm или gnome-terminal посылает последовательность ASCII ESC [3~. У gnome-terminal поведение обоих клавиш настраивается на закладке настроек Edit->Profile Preferences->Compatibility.

Символ стирания строки KILL (Ctrl-W по умолчанию) приводит к стиранию всей текущей строки.

Например, предполагая, что символ забоя равен ASCII BS, вы набираете на клавиатуре:

`datx`*`BS`*`e`
Символы, набранные на клавиатуре, записываются в "сырую" очередь по мере ввода. Затем при копировании из "сырой" очереди в каноническую, символы просматриваются. При этом символы BS и стоящий перед ним выбрасываются. Поэтому, программа, читающая с терминала, получит строку:

```
date
```
При выводе, символы, генерируемые вашей прикладной программой, накапливаются в выходной очереди. При этом может происходить требуемая настройками постобработка.

Использование termios(3C)

Существует более пятидесяти различных параметров и флагов, управляющих терминальным интерфейсом. Эти параметры можно изменять через ioctl termio(7I), функции termios(3C) и команду stty(1). Ниже приводятся некоторые из основных характеристик терминала.

Параметры RS232

Используя флаг c_cflag, вы можете управлять скоростью передачи, количеством бит в символе, количеством стоповых битов, обработкой бита четности и т.д. Это необходимо потому, что протокол RS232 не имеет средств автосогласования указанных параметров, и их необходимо настраивать вручную так, чтобы настройки терминала и терминального порта компьютера совпадали. У псевдотерминалов эти параметры сохраняются только для совместимости; изменение большинства из них (кроме количества бит в символе) ни на что не влияют.

Отображение символов.

Вы можете управлять обработкой символов возврата каретки (CR) и перевода строки (NL). Вы можете преобразовывать NL в CR (INLCR) и CR в NL (ICRNL). Вы можете также игнорировать поступающие символы CR (IGNCR). Это полезно для терминалов, которые генерируют последовательность символов CR-NL при нажатии клавиши <RETURN>. На других терминалах, которые генерируют одиночный символ CR, лучше

принимать этот символ и преобразовывать его в NL. Флаги для этих режимов помещаются в c_iflag. Соответственно, флаги ONLCR и OCRNL в c_oflag используются для преобразования NL в CR-NL или CR в NL, соответственно, при выводе.

Для терминалов, которые отображают только буквы верхнего регистра, могут быть установлены флаги XCASE в c_iflag, IUCLC в c_iflag и OLCUC в c_oflag. Буквы верхнего регистра отображаются в нижний регистр при вводе и наоборот - при выводе. Буквы верхнего регистра предваряются символом обратной косой черты (backslash, \).

По умолчанию, некоторые терминалы используют семибитный код ASCII для представления символов. Однако может быть необходима работа с восьмибитными данными. Например, это может потребоваться программам, работающим с национальными алфавитами или UTF-8. Для того, чтобы выключить срезание восьмого бита при вводе, вы должны очистить флаг ISTRIP в

c_iflag, установить флаг CS8 для передачи восьмибитных символов и, возможно, выключить контроль четности очисткой PARENB в c_cflag.

Задержки и табуляции.

Для механических терминалов можно установить различные флаги в c_oflag, управляющие задержками при обработке перевода строки, возврата каретки, горизонтальной табуляции, сдвига каретки назад (backspace), вертикальной табуляции и перевода страницы. Горизонтальная табуляция может преобразовываться в соответствующее число пробелов установкой флага TAB3.

(Продолжение на следующей странице)

Управление потоком.

Вывод на терминал может быть приостановлен нажатием СТОП-символа, по умолчанию CTRL-S, и возобновлен нажатием СТАРТ-символа, по умолчанию CTRL-Q. Эта возможность разрешается установкой флага IXON в c_flag. Иначе, эти символы не имеют специального смысла. Кроме того, если установить флаг IXANY, то любой символ будет возобновлять вывод. Для управления потоком вывода используется функция tcflow(2). Кроме управления потоком вывода, установкой флага IXOFF в c_iflag можно управлять потоком ввода. При этом, если входная очередь приближается к заполнению, драйвер терминального устройства пошлет

СТОП-символ для приостановки ввода, и СТАРТ-символ - для его возобновления. Это может быть полезно, если прикладная программа получает данные из удаленной системы.

Более удобный способ управлять потоком данных при выводе на терминал — это утилита more(1) или её аналоги. Однако надо подчеркнуть, что при использовании more(1), программа осуществляет вывод не на терминал, а в программный канал, направленный на вход more(1); это может повлиять на поведение некоторых программ.

Управляющие символы

При вводе некоторые символы имеют специальное значение. Например, # или ASCII BS

является символом забоя, CTRL-W — символом стирания строки, CTRL-D — концом ввода и т.д. Эти символы хранятся в массиве c_cc[] и могут быть изменены. Например, при использовании дисплея, гораздо удобнее использовать в качестве символа забоя BS (сдвиг каретки назад), чем #

Эхо

Терминалы обычно работают с UNIX-системами в полнодуплексном режиме. Это означает, что данные передаются в обоих направлениях одновременно и что компьютер обеспечивает эхо (отображение на экране или на печати) получаемых символов. Эхо выключается очисткой флага ECHO в c_lflag. Кроме того, стертые нажатием клавиши забоя символы могут затираться установкой флага ECHOE. Если установить флаг ECHOK, то стирание строки символом KILL будет приводить к переводу строки на терминале.

Немедленный ввод

Обычно символы при вводе накапливаются, пока не соберется полная строка, завершенная NL. Только после этого удовлетворяется запрос read(2), даже если он требовал только один символ. Значение, возвращаемое вызовом read(2), равно количеству прочитанных в действительности символов. Во многих прикладных программах, таких как редакторы форм или полноэкранные текстовые редакторы, строки ввода не имеют смысла. В таких программах необходимо читать символы по мере их ввода. При очистке флага ICANON в c_lflag вводимые символы не группируются в строки, и read(2) читает их по мере поступления. Этот режим известен также как режим неканонического ввода. Вместо этого удовлетворение запроса read(2) определяется параметрами MIN (минимальное количество нажатых клавиш) и TIME (промежуток времени

между введенными символами). Если ICANON установлен, то режим ввода называется каноническим.

(Продолжение на следующей странице)

"Сырой" терминальный ввод/вывод

Терминальные порты могут использоваться для подключения не только терминалов, но и других устройств, например, модемов, мышей или различной контрольно-измерительной аппаратуры. При работе с такими устройствами, прикладные программы и драйверы терминальных устройств могут быть вынуждены передавать и принимать произвольные восьмибитные данные. Это могут быть сообщения мыши о передвижении и нажатии кнопок, данные протокола PPP или поступающие с измерительного устройства данные. Для разрешения этого необходимо установить восьмибитные данные, неканонический ввод, запретить все отображения символов и управление потоком, устранить специальные значения всех управляющих символов и выключить эхо. В конце раздела приводится пример использования "сырого" терминального ввода/вывода.

Получение и установка атрибутов терминала

Первые две функции, описанные на странице Руководства termios(3C), используются для получения и установки атрибутов терминала. tcgetattr(3C) получает текущие установки терминального устройства. tcsetattr(3C) изменяет эти установки. Эти функции получают и передают требуемые параметры в виде управляющей структуры termios.

Параметры функций tcgetattr(3C) и tcsetattr(3C) таковы:

fildes дескриптор файла, соответствующий терминальному устройству. Для этого дескриптора вызов isatty(3F) должен возвращать ненулевое значение.

optional_actions Комбинация флагов, определенных в <termios.h>. optional_actions определяет, когда должны быть выполнены изменения и что делать с находящимися в буферах устройства данными при изменении параметров.

termios_p указатель на структуру termios. Эта структура содержит флаги и битовые поля, используемые для управления терминальным интерфейсом ввода/вывода. Флаги и поля этой структуры обсуждаются позже.

После исполнения tcgetattr(3C) рекомендуется сохранить копию этой структуры, чтобы программа могла вернуть начальное состояние терминального интерфейса. Дело в том, что функции tcsetattr(3C) изменяют настройки не файлового дескриптора вашего процесса, а настройки драйвера в ядре Unix. Внесенные вами изменения не откатываются автоматически при завершении программы, поэтому ненормально завершившаяся программа может оставить терминал в непригодном для работы состоянии.

В некоторых случаях, для восстановления параметров терминала можно воспользоваться командой stty(1). В Solaris, команда stty sane пытается привести терминал в режим, пригодный для интерактивной работы.

При изменении настроек терминала, вместо того, чтобы формировать значения полей структуры termios самостоятельно, рекомендуется получить текущие настройки терминала вызовом tcgetattr(3C), затем поменять нужные вам параметры в структуре и установить новые параметры вызовом tcsetattr(3C). При этом вы, по возможности, сохраните те настройки, которые пользователь мог сделать до запуска вашей программы. Кроме того, в последующих версиях Solaris и в других Unix-системах, в структуре termios могут появиться дополнительные поля или флаги. Если ваша программа не будет без нужды изменять незнакомые ей настройки, это значительно облегчит её адаптацию к новым версиям Solaris и ее перенос на другие платформы.

Параметр optinal_actions функции tcsetattr(2)

Параметр optional_actions функции tcsetattr(2) может принимать следующие значения:

TCSANOW Атрибуты изменяются немедленно.

TCSADRAIN Изменения атрибутов происходят только после того, как был передан («осушен») весь вывод в fildes. Этот запрос может быть использован при изменении атрибутов, которые влияют на обработку вывода.

TCSAFLUSH Это похоже на TCADRAIN. Изменение происходит после того, как весь вывод в fildes был передан, а непрочитанный ввод сброшен. Например, этот запрос полезен, если нужно проигнорировать все символы в буфере ввода.

Структура termios

Структура termios используется для представления характеристик терминального устройства. Структура одна и та же для всех терминальных устройств, независимо от изготовителя аппаратуры. Это предоставляет единообразный способ изменения характеристик и поведения терминального устройства. В частности, эта структура используется модулем STREAMS для изменения поведения аппаратного и программного интерфейса ввода/вывода.

Ниже перечислены поля структуры termios:

c_iflag флаги, управляющие предобработкой ввода с терминала.

c_oflag флаги, управляющие системной постобработкой вывода на терминал.

c_cflag флаги, описывающие аппаратные характеристики терминального интерфейса.

c_lflag флаги, управляющие разбиением потока на строки.

c_cc[] массив специальных управляющих символов.

Эти поля будут подробнее обсуждаться далее в этом разделе.

`Ссылка: /usr/include/sys/termios.h`

Управляющие символы

Управляющие символы, определенные в массиве c_cc[] имеют специальное значение и могут быть изменены вызовом tcsetattr(). Далее в этом разделе, приведены символьные константы, определенные в <termios.h>. Эти символьные константы могут быть использованы как индексы в массиве c_cc[]. Значения по умолчанию для соответствующих элементов перечислены в скобках. Некоторые из этих управляющих символов описаны ниже.

Следует иметь в виду, что в кодировке ASCII, первые 32 символа зарезервированы для выполнения различных управляющих функций (см. ascii(5)). Видеотерминалы и терминальные эмуляторы генерируют такие символы при одновременном нажании комбинации алфавитных (или некоторых неалфавитных) клавиш и клавиши Ctrl. Нажатие клавиши Ctrl приводит к срезанию старших бит кода соответствующего символа. Исключение составляет комбинация Ctrl-?, которая выдает код '\0x1F' (ASCII DEL). Так, комбинация клавиш Ctrl-D — это символ '\0x3' (ASCII EOT).

VINTR (Ctrl-C или ASCII DEL) генерирует сигнал SIGINT, который посылается всем процессам в группе первого плана, связанной с этим терминалом. По умолчанию процесс при получении этого сигнала будет завершен, но он может проигнорировать этот сигнал или перехватить его при помощи функции обработки.

VQUIT (CTRL-\) генерирует сигнал SIGQUIT. Этот сигнал обрабатывается так же, как и SIGINT.

VERASE (Ctrl-H, Ctrl-? или #) стирает предыдущий символ. Он не может стереть символ перед началом строки, ограниченной символами NL, EOF, EOL или EOL2.

VWERASE (CTRL-W) очищает предыдущее "слово". Он не может стереть слово из предыдущей строки, ограниченной символами NL, EOF, EOL или EOL2. Это функция расширения, поэтому для ее использования необходимо установить флаг IEXTEN.

VKILL (Ctrl-U) стирает всю строку, ограниченную символами NL, EOF, EOL или EOL2.

VEOF (CTRL-D) может использоваться для обозначения конца файла при вводе с терминала. Когда получен этот символ, все символы, ожидающие считывания, будут немедленно переданы программе, без ожидания символа новой строки, и остаток строки игнорируется. Таким образом, если в очереди не было символов, то есть EOF был послан в начале строки, read получит ноль символов, что является стандартным обозначением конца файла.

(Продолжение на следующей странице)

VSTOP (CTRL-S) может использоваться для временной приостановки вывода. Это полезно на экранных терминалах, чтобы вывод не исчезал с экрана, пока пользователь его не прочитал. Если вывод уже приостановлен, вводимые СТОП-символы игнорируются и не будут прочитаны.

VSTART (CTRL-Q) используется для возобновления вывода, остановленного СТОП--символом. Если ввод не был приостановлен, символы VSTART игнорируются и не читаются.

VSUSP (CTRL-Z) генерирует сигнал SIGTSTP, который приостанавливает все процессы в группе процессов первого плана этого терминала. Например, этот символ используется для функций управления заданиями в shell.

VDISCARD (CTRL-O) приводит к тому, что весь вывод будет игнорироваться, пока не будет послан еще один символ DISCARD, программа не выведет новые символы или не сбросит соответствующее условие. Это функция расширения, и исполняется, только если установлен флаг IEXTEN.

VLNEXT (CTRL-V) игнорирует специальное значение следующего символа. Это работает для всех специальных символов из массива c_cc[]. Это позволяет вводить символы, которые в ином случае были бы проинтерпретированы системой (такие, как KILL, QUIT). Символы VERASE, VKILL и VEOF могут также быть введены после символа обратной косой черты (backslash, \). В этом случае они также не вызовут исполнения специальной функции.

VREPRINT (CTRL-R или ASCII DC2) печатает символ новой строки и все символы, которые ожидали в очереди ввода (как если бы это была новая строка). Это считается функцией расширения, поэтому работает, только если установлен IEXTEN.

Для изменения управляющего символа, необходимо получить текущие терминальные атрибуты вызовом tcgetattr(3C), присвоить требуемому элементу массива c_cc[] новое значение и изменить атрибуты терминала вызовом tcsetattr(3C). Если значение управляющего символа будет _POSIX_VDISABLE, то функция, ассоциированная с этим символом, будет выключена.

Некоторые флаги режимов

Следующая страница перечисляет некоторые атрибуты терминала, которые могут быть изменены. Флаги, перечисленные во второй колонке таблицы, являются символьными константами, определенными в <sys/termios.h>, и представляют собой значения отдельных битов. Значения флагов хранятся в следующих четырех полях структуры termios:

c_iflag Поле c_iflag описывает режим обработки ввода. Если установлен флаг IGNBRK, то последовательность нулевых бит (break condition, некоторые терминалы или модемы таким образом кодируют разрыв линии) игнорируется, то есть не помещается в очередь ввода и не может быть считано ни одним процессом. Иначе, если установлен флаг BRKINT, условие разрыва генерирует сигнал прерывания и сбрасывает входную и выходную очереди.

Если установлен ISTRIP, то вводимые символы обрезаются до 7 бит, иначе они передаются как 8-битные значения. Если установлен ICRNL, то символ CR переводится в символ NL.

Если установлен IXON, разрешается старт/стоповое управление выводом. Получение СТОП-символа будет задерживать вывод, а СТАРТ-символ - возобновляет его. Все СТАРТ/СТОП-символы игнорируются и не читаются. Если установлен IXANY, любой введенный символ будет возобновлять приостановленный вывод.

c_oflag Поле c_oflag содержит флаги, управляющие обработкой вывода. Если установлен флаг OPOST, выводимые символы подвергаются постобработке в соответствии с остальными флагами, иначе они передаются без изменений.

Если установлен ONLCR, символ NL передается как пара CR-NL. TAB3 и XTABS задают замену символов табуляции пробелами.

c_cflag Поле c_cflag управляет аппаратными атрибутами терминального интерфейса. Биты CBAUD задают скорость передачи. Биты CSIZE задают размер символа в битах как для приема, так и для передачи.

Если CSTOPB установлен, передаются два стоповых бита. Флаги PARENB и PARODD управляют контролем четности.

c_lflag Если установлен ICANON, разрешена каноническая обработка ввода. Допускаются функции редактирования (забой и стирание строки) и объединение вводимых символов в строки, ограниченные символами NL, EOF, EOL, EOL2. Если ICANON не установлен, данные для удовлетворения запросов чтения берутся прямо из "сырой" очереди. Неканоническая обработка будет обсуждаться далее.

Если установлен ECHO, на каждый полученный символ выдается эхо. Если установлен режим ICANON, доступен ряд функций управления эхо. Если установлены флаги ECHO и ECHOE, а ECHOPRT не установлен, эхо для символа забоя выдается как ASCII BS SP BS (сдвиг каретки назад - пробел - сдвиг каретки назад), что очищает последний символ на экране терминала. Если ECHOK установлен, а ECHOKE нет, то после символа стирания строки передается NL, чтобы подчеркнуть, что строка была стерта.

Символ переключения режима (escape), идущий перед символами очистки или стирания строки, лишает эти символы их функции. Если установлен флаг ISIG, вводимые символы проверяются на совпадение с символами INTR, QUIT, SUSP и DSUSP. Если вводимый символ соответствует одному из них, посылается соответствующий сигнал. Если ISIG не установлен, не выполняется никакой проверки.

Если установлен флаг IEXTEN, то над входными данными будут выполняться функции из расширенного набора, зависящие от реализации. Этот флаг должен быть установлен для распознавания символов WERASE, REPEINT, DISCARD и LNEXT.

Запирание терминала - пример

Эта программа запирает терминал пользователя. Она запрашивает пользователя ввести «ключ». Конечно, пользователю было бы удобнее, чтобы ключ совпадал с его паролем, но в современных Unix-системах хэши паролей доступны только пользователю root, поэтому ключ необходимо вводить при запуске программы. Эхо выключено, поэтому ключ не будет показан на терминале. Этот же ключ должен быть введен, чтобы получить доступ к терминалу. Программа работает так:

9 Объявляет структуру termios для сохранения установок терминала.

10 Объявляет переменную tcflag_t для сохранения текущих значений c_lflag.

11 Объявляет массив символов для сохранения значения первого ключа. Ключ может быть очень длинным, так как BUFSIZ имеет большое значение.

13 Программа получает текущие установки терминала. Первый параметр - дескриптор файла стандартного ввода, полученный макросом fileno, определенным в <stdio.h>. Если stdin переназначен, то программа работать не будет. Второй аргумент - адрес структуры termios.

14 Сохраняется значение поля c_lflag. Это значение затем будет использовано для восстановления состояния терминального интерфейса.

15-16 Запрещается сравнение ввода с управляющими символами INTR, QUIT, SUSP и DSUSP. Эхо выключается. tcsetattr(3C) вызывается с флагом TCSAFLUSH для изменения состояния терминального интерфейса и сброса всех введенных символов.

17 Запоминается ключ, который позднее будет использован для отпирания терминала.

18-25 Для того, чтобы выйти из этого цикла, программа должна получить значение, совпадающее со значением исходного ключа. Если ключи совпадают, терминальный интерфейс возвращается в исходное состояние. Это не делается автоматически при завершении программы. Если программа завершится ненормально, терминал может остаться в странном состоянии.

28 Эта функция возвращает указатель на строку символов, которая содержит ключ, введенный пользователем.

30 Объявляется массив символов для сохранения значения ключа. Он объявлен как static, и указатель на этот массив будет возвращен функцией.

32 Выдается приглашение

33 Первый символ в массиве line[] устанавливается в невозможное значение. Для чего это нужно? Ответ: Если getkey() был вызван во второй раз, и был введен только EOF, не будет считано ни одного символа; содержимое line[] будет тем же, что и раньше, и возвращенная строка совпадет со значением, полученным от первого вызова этой функции, что совершенно неправильно.

34 Строка ввода считывается без эхо. Не делается проверки на совпадение с управляющими символами, такими как INTR, QUIT и т.д. Эти символы обрабатываются так же, как обычные.

Файл: termlock.c

ЗАПИРАНИЕ ТЕРМИНАЛА - ПРИМЕР

```
 1 #include <string.h>
 2 #include <unistd.h>
 3 #include <stdio.h>
 4 #include <termios.h>
 5 static char *getkey(void);
 6
 7 main()          /* lock the terminal */
 8 {
 9      struct termios tty;
10      tcflag_t savflags;
11      char key[BUFSIZ];
12
13      tcgetattr(fileno(stdin), &tty);
14      savflags = tty.c_lflag;
15      tty.c_lflag &= ~(ISIG | ECHO);
16      tcsetattr(fileno(stdin), TCSAFLUSH, &tty);
17      strcpy(key, getkey());
18      for (;;) {
19       if (strcmp(key, getkey()) == 0){
20            tty.c_lflag = savflags;
21            tcsetattr(fileno(stdin), TCSAFLUSH, &tty);
22            break;
23       }
24       fprintf(stderr,"incorrect key try again.\n");
25      }
26 }
27
28 static char *getkey(void)     /* prompt user for key */
29 {
30      static char line[BUFSIZ];
31
32      fputs("Key: ", stderr);
33      line[0] = '\377';  /*change first char for EOF to fgets*/
34      fgets(line, BUFSIZ, stdin);
35      fputs("\n", stderr);
36      return(line);
37 }
```

Неканонический ввод

В обычном (каноническом) режиме символы собираются вместе, пока не будет введена полная строка, завершенная символом NL. Только после этого вызов read(2) возвращает управление, даже если он запрашивал только один символ. Возвращенное вызовом read(2) значение показывает количество символов, которые были прочитаны на самом деле до ввода NL.

Однако в некоторых прикладных программах, таких, как обработка экранных форм или полноэкранных редакторах, "строки" ввода не имеют смысла. Например, эти программы могут требовать символы по мере их ввода с клавиатуры. Если очистить флаг ICANON в c_iflag, вводимые символы не будут собираться в строки и read(2) будет читать их по мере ввода.

Параметры MIN и TIME определяют условия, при которых будет удовлетворен запрос read(2). MIN определяет минимальное количество символов, которые должны быть получены. TIME представляет собой таймер с квантом времени 0.1 секунды, который сбрасывается при вводе каждого символа. Таким образом, TIME кодирует не общее время ввода строки, а межсимвольный интервал. Это сделано для упрощения считывания терминальных кодов расширения, ведь терминалы передают последовательные символы кода быстрее, чем обычный человек может нажимать клавиши.. Символы EOF и EOL в неканоническом режиме не используются, поэтому эти позиции в массиве c_cc[] используются для MIN и TIME соответственно. Ниже описаны четыре возможных сочетания значений MIN и TIME:

MIN > 0, TIME > 0. В этом случае, TIME служит для измерения времени между вводом одиночных символов и стартует после получения первого символа. Счетчик времени сбрасывается после каждого очередного символа. Если до истечения интервала времени будет получено MIN символов, запрос read(2) удовлетворяется. Если, наоборот, время истекает раньше, чем было считано MIN символов, то все введенные до этого момента символы возвращаются пользователю. Замечание: если TIME истекло, то будет возвращен по крайней мере один символ. Если MIN равен 1, значение TIME не играет роли.

MIN > 0, TIME = 0 Если значение TIME равно нулю, таймер не используется. Имеет значение только MIN. В этом случае запрос read(2) удовлетворяется только тогда, когда получены MIN символов.

MIN = 0, TIME > 0 Если MIN равен нулю, TIME больше не является счетчиком межсимвольного

времени. Теперь таймер активизируется при обработке системного вызова read(2). Запрос read(2) удовлетворяется когда поступил хотя бы один символ или истекло время. Если в течении TIME*0.1 секунд после начала чтения не поступило ни одного символа, запрос возвращает управление с нулевым количеством прочитанных символов.

MIN = 0, TIME = 0 В этом случае, read(2) возвращает управление немедленно. Возвращается минимум из запрошенного и имеющегося на данный момент в буфере количества символов, без ожидания ввода дополнительных символов.

Клавиатурный тренажер - Пример

Эта программа может служить в качестве клавиатурного тренажера и является примером неканонического ввода. Она выводит на экран строку текста, и пользователь должен напечатать ее. Символы по мере ввода проверяются. Если введен неправильный символ, программа издает звуковой сигнал и печатает звездочку. Если введен правильный символ, он выводится на экран. Программа работает так:

15-20 Терминальный специальный файл открывается для чтения, и его текущий режим записывается в структуру termios. Библиотечная функция isatty(3F) проверяет, связан ли файловый дескриптор 1 с терминалом. Иными словами, эта программа не должна исполняться с использованием перенаправления стандартного ввода/вывода shell. Функция isatty описана на странице руководства ttyname(3F).

22-24 Терминальный интерфейс переводится в режим неканонического ввода без эхо и без обработки специальных символов.

25 Вызовом стандартной библиотечной функции setbuf(3) запрещается локальная буферизация в библиотечных функциях вывода. Это приводит к тому, что putchar(3) в следующих строках будет немедленно вызывать write(2) в стандартный вывод.

28-37 Этот цикл читает по одному символу и сравнивает его с соответствующим символом строки text. Если символ введен правильно, он немедленно выводится на терминал. Иначе издается звуковой сигнал и на терминал выводится звездочка.

Ниже приведен пример работы программы:

```
$ typtut
Type in beneath the following line

The quick brown fox jumped over the lazy dog's back
The *uick b*own f** jumped over t*e lazy dog's back

number of errors: 5
```

Файл: typtut.c

КЛАВИАТУРНЫЙ ТРЕНАЖЕР - ПРИМЕР НЕКАНОНИЧЕСКОГО ВВОДА

```
 1 #include <unistd.h>
 2 #include <stdio.h>
 3 #include <fcntl.h>
 4 #include <termios.h>
 5 #include <string.h>
 6 #include <stdlib.h>
 7
 8 main()
 9 {
10      char ch, *text =
11      "The quick brown fox jumped over the lazy dog\'s back";
12      int fd, i, errors = 0, len;
13      struct termios tty, savtty;
14
15      fd = open("/dev/tty", O_RDONLY);
16      tcgetattr(fd, &tty);
17      if (isatty(fileno(stdout)) == 0) {
18       fprintf(stderr,"stdout not terminal\n");
19       exit(1);
20      }
21      savtty = tty;
22      tty.c_lflag &= ~(ISIG | ICANON | ECHO);
23      tty.c_cc[VMIN] = 1;     /* MIN */
24      tcsetattr(fd, TCSAFLUSH, &tty);
25      setbuf(stdout, (char *) NULL);
26      printf("Type beneath the following line\n\n%s\n", text);
27      len = strlen(text);
28      for (i = 0; i < len; i++) {
29       read(fd, &ch, 1);
30       if (ch == text[i])
31            putchar(ch);
32       else {
33            putchar('\07');
34            putchar('*');
35            errors++;
36       }
37      }
38      tcsetattr(fd, TCSAFLUSH, &savtty);
39      printf("\n\nnumber of errors: %d\n", errors);
40 }
```

Программа просмотра файла - Пример

Эта программа может использоваться для просмотра файла на терминале и служит другим примером неканонического ввода. Она служит альтернативой CTRL-S и CTRL-Q, которые, соответственно, задерживают и возобновляют вывод. В этой программе любая клавиша приостанавливает или возобновляет вывод на терминал. Например, пробел может использоваться как переключатель. Этот эффект достигается изменением значения MIN

между нулем и единицей. Эта программа работает так:

13-16 Стандартная библиотечная функция fopen(3) открывает файл для просмотра.

17 Считывается текущий режим терминального интерфейса.

18 Этот режим сохраняется. Позднее он будет использоваться для восстановления состояния терминального интерфейса.

19-22 Терминальный интерфейс переключается в режим неканонического ввода. Кроме того, INTR, QUIT и остальные управляющие символы не анализируются и эхо выключено. Чтение с терминала будет ожидать в течении 0.1 секунды, потому что MIN равен нулю, а TIME равен 1.

24-33 Этот цикл считывает строки из файла и выводит их на терминал.

25 read(2) пытается считать с терминала один символ. Так как чтение возвращает управление немедленно (без ожидания), символ будет прочитан, только если он был введен до вызова read(2). Непрочитанные символы накапливаются в буфере. Если считан символ, read(2) возвращает 1 и исполняются операторы 27-31. Если не прочитано ни одного символа,read(2) возвращает 0. Это называется опросом ввода с терминала.

26-27 MIN установлен в единицу, так что запросы чтения с терминала будут ждать ввода.

28 Как только символ введен, запрос read(2) удовлетворяется и возвращает управление.

29-30 Чтение с терминала снова переводится в режим опроса.

34 Восстанавливается исходный режим работы терминала. После нажатия клавиши для приостановки вывода возникает небольшая задержка, во время которой выводится несколько лишних строк. Это связано с буферизацией вывода и низкой скоростью работы терминала.

Как можно прекратить просмотр длинного файла? Например, после чтения в строке 25, мы можем проверять символ на равенство букве q (quit). Если был введен этот символ, программа выходит из цикла. Можно даже использовать символ DEL, так как он читается наравне с остальными символами.

Файл: `lister.c`

ПРОГРАММА ПРОСМОТРА ФАЙЛОВ - ПРИМЕР НЕКАНОНИЧЕСКОГО ВВОДА

```
 1 #include <unistd.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <fcntl.h>
 5 #include <termios.h>
 6
 7 main(int argc, char *argv[])
 8 {
 9     struct termios tty, savtty;
10     char ch, line[BUFSIZ];
11     FILE *fp;
12
13     if ((fp = fopen(argv[1], "r")) == NULL) {
14         printf("Cannot open %s\n", argv[1]);
15         exit(1);
16     }
17     tcgetattr(fileno(stdin), &tty);
18     savtty = tty;
19     tty.c_lflag &= ~(ISIG | ICANON | ECHO);
20     tty.c_cc[VMIN] = 0;     /* no characters */
21     tty.c_cc[VTIME] = 1;    /* wait for 100 msec */
22     tcsetattr(fileno(stdin), TCSANOW, &tty);
23
24     while (fgets(line, BUFSIZ, fp) != NULL) {
25         if (read(fileno(stdin), &ch, 1) == 1) {
26             tty.c_cc[VMIN] = 1;     /* one char */
27             tcsetattr(fileno(stdin), TCSANOW, &tty);
28             read(fileno(stdin), &ch, 1);
29             tty.c_cc[VMIN] = 0;     /* no chars */
30             tcsetattr(fileno(stdin), TCSANOW, &tty);
31         }
32         fputs(line, stdout);
33     }
34     tcsetattr(fileno(stdin), TCSANOW, &savtty);
35     fclose(fp);
36 }
```

Передача двоичного файла - Пример

На следующей странице приведены подпрограммы для установки терминального интерфейса в режим ввода/вывода необработанных данных ("сырой") и восстановления исходного режима. Эти подпрограммы работаюттак:

8 fd присваивается 0 или 1, если setrawio вызывается получателем (recv) или передатчиком (xmit), соответственно.

10-11 Если дескриптор файла 0 или 1 не ассоциирован с терминальным специальным файлом, эта функция возвращает управление немедленно. Это позволяет программе, использующей setrawio, перенаправить свой стандартный ввод/вывод в файл или программный канал (в этом случае, setrawio вообще не нужен). Если же дескриптор ассоциирован с терминальным специальным файлом, то режим интерфейса переключается на "сырой" ввод/вывод.

12-15 Для заданного дескриптора файла считывается значение структуры termios.

16 Копия структуры termios сохраняется для восстановления режима терминального интерфейса, который был до вызова setrawio.

17-22 Флаги в структуре termios устанавливаются для приема и передачи произвольных восьмибитных данных. Срезание старшего бита, отображение вводимых символов и управление потоком ввода выключены. Размер символа установлен равным восьми битам, и выключен контроль четности. Поиск специальных управляющих символов, канонический ввод и эхо в поле флагов локального режима также выключены. Заметьте, что эта функция одна и та же как для ввода, так и для вывода, так как установки флагов для ввода не влияют на вывод, и наоборот.

23-24 Для неканонического ввода MIN устанавливается равным размеру буфера ввода, используемого в вызове read(2). Таймер не используется, поэтому TIME устанавливается в ноль.

25 Режим терминального интерфейса будет изменен после того, как весь вывод будет передан, а ввод - сброшен.

32 Терминальный интерфейс возвращается в то состояние, в котором он находился до вызова функции setrawio.

Файл: setrawio.c

ПЕРЕДАЧА ДВОИЧНОГО ФАЙЛА - ПРИМЕР setrawio.c

```
 1 #include <stdlib.h>
 2 #include <stdio.h>
 3 #include <termios.h>
 4 #include "xmit.h"
 5
 6 static struct termios tty, savtty;
 7
 8 void setrawio(int fd)     /* set "raw" input/output modes */
 9 {
10      if (!isatty(fd))
11       return;
12      if (tcgetattr(fd, &tty) == -1) {
13       perror("tcgetattr");
14       exit(2);
15      }
16      savtty= tty;
17      tty.c_iflag &= ~(BRKINT | ISTRIP | INLCR | ICRNL
18       | IUCLC | IXON);
19      tty.c_oflag &= ~OPOST;
20      tty.c_cflag |= CS8;
21      tty.c_cflag &= ~PARENB;
22      tty.c_lflag &= ~(ISIG | ICANON | ECHO);
23      tty.c_cc[VMIN] = BLOCKSIZE;     /* MIN */
24      tty.c_cc[VTIME] = 0;          /* TIME */
25      tcsetattr(fd, TCSAFLUSH, &tty);
26 }
27
28 void restorio(int fd)     /* restore terminal modes */
29 {
30      if (!isatty(fd))
31       return;
32      tcsetattr(fd, TCSAFLUSH, &savtty);
33 }
```

Сессии и группы процессов

Все процессы объединены в сессии. Процессы, принадлежащие к одной сессии, определяются общим идентификатором сессии (sid). Лидер сессии - это процесс, который создал сессию вызовом setsid(2). Идентификатор процесса лидера сессии совпадает с его sid. Сессия может выделить себе управляющий терминал для того, чтобы дать пользователю возможность управлять исполнением заданий (групп процессов) внутри сессии. При входе в систему создается сессия, которая имеет идентификатор сессии, равный идентификатору процесса вашего входного shell'a. Также, при открытии каждого окна xterm(1) или закладки gnome-terminal(1), создается сессия, идентификатор которой совпадает с идентификатором дочернего процесса, запущенного терминальным эмулятором.

Если ваш командный процессор не предоставляет управления заданиями, все процессы в вашей сессии являются также членами единственной в этой сессии группы процессов, которая была создана вызовом setsid(2). В этом случае, функциональность сессии совпадает с функциональностью группы процессов.

В командном процессоре, предоставляющем управление заданиями (ksh(1), jsh(1), bash(1)), управляющий терминал совместно используется несколькими группами процессов, так как для каждой команды, запущенной с управляющего терминала, создается своя группа процессов. Командный процессор называет такие команды «заданиями» (jobs). Одновременно работающие задания идентифицируются номерами, обычно совпадающими с порядком их запуска.

Каждая группа процессов имеет лидера - процесс, идентификатор которого совпадает с идентификатором группы процессов. Управляющий терминал выделяет одну из групп процессов в сессии, как группу основных процессов (процессов первого плана). Все остальные процессы в сессии принадлежат к группам фоновых процессов. Группа процессов первого плана получает сигналы, посланные с терминала. По умолчанию, группа процессов, связанная с процессом, который выделил себе управляющий терминал, изначально становится группой основных процессов.

Кроме того, если процесс из фоновой группы пытается читать с терминала или выводить на него данные, он получает сигнал, соответственно, SIGTTIN или SIGTTOUT. Оба эти сигнала приводят к остановке соответствующего процесса. Shell при этом выводит сообщение [имя программы] stopped: tty input. Для продолжения исполнения такой программы необходимо перевести соответствующую группу процессов на первый план командой fg.

Обычные команды запускаются как задания первого плана. Shell ожидает завершения лидера группы этого задания и выдает приглашение только после его завершения. Если в конце команды стоит символ &, shell запускает такую команду как фоновое задание и не дожидается ее завершения. Если пользователь во время работы команды первого плана введет символ VSUSP (Ctrl-Z), группа получает сигнал SIGTSTP и останавливается, а shell выдает приглашение. Пользователь может вернуться к взаимодействию с этой группой процессов, введя команду fg, или продолжить её исполнение в фоне командой bg. Если пользователь имеет несколько приостановленных или фоновых заданий, он может выбирать между ними, идентифицируя их по номерам. Так, переключение на задание 3 делается командой fg %3.

Вывести список заданий, их номера и состояния можно командой jobs.

Также, встроенная команда kill у shell'ов с управлением заданиями, может принимать номер задания вместо номера процесса; при этом сигнал будет послан всем процессам соответствующей группы.

Получение/установка идентификатора сессии

Каждый процесс принадлежит к сессии и группе процессов. Сессия создается для вас, когда вы входите в систему. Первый терминал, открытый лидером сессии, который не был уже ассоциирован с другой сессией, становится управляющим терминалом для этой сессии. Если при открытии терминала лидер сессии укажет флаг NOCTTY, терминал не станет управляющим. Это позволяет процессам-демонам выводить сообщения на системную консоль.

Управляющий терминал генерирует сигналы завершения и прерывания (quit и interrupt), а также сигналы управления заданиями. Управляющим терминалом для вашего shell'a является тот терминал, с которого вы вошли в систему. Управляющий терминал наследуется процессом, порожденным при помощи fork(2). Процесс может разорвать связь со своим управляющим терминалом, создав новую сессию с использованием setsid(2).

Если сессия так и не откроет управляющий терминал, соответствующий процесс будет называться «демоном» (daemon). Большинство системных сервисных процессов, таких, как init(1M), svc.startd(1M) crond(1M) или сетевых сервисов, таких, как sshd(1M), запускаются как демоны. Иногда демонами называют также системные процессы ttymon(1M), обслуживающие терминальные порты, хотя эти процессы имеют управляющие терминалы.

Если вызывающий процесс не является уже лидером группы процессов, setsid(2) устанавливает идентификаторы группы процессов и сессии вызывающего процесса равными его идентификатору процесса и отсоединяет его от управляющего терминала. setsid(2) создает новую сессию, превращая вызвавший процесс в лидера этой сессии. Новые сессии создаются чтобы:

1. отсоединить вызвавший процесс от терминала, так что этот процесс не будет получать от этого терминала сигналы SIGHUP, SIGINT и сигналы управления заданиями.

2. позволить процессу назначить новый управляющий терминал. Только лидер сессии может назначить управляющий терминал. Например, ttymon создает новую сессию и, таким образом, назначает управляющий терминал, когда пользователь входит в систему.

getsid(2) возвращает идентификатор сессии процесса с идентификатором, равным pid. Если pid равен нулю, getsid(2) возвращает идентификатор сессии вызвавшего процесса.

Получение/установка идентификатора группы процессов

Группа процессов - это совокупность процессов с одним и тем же идентификатором группы процессов. Управляющий терминал считает одну из групп процессов в сессии группой основных процессов. Все процессы в основной группе будут получать сигналы, относящиеся к терминалу, такие как SIGINT и SIGQUIT.

Новый идентификатор группы процессов может быть создан вызовом setpgid(2). Группы процессов, отличные от основной группы той же сессии, считаются группами фоновых процессов. ksh использует группы процессов для управления заданиями. Фоновые процессы не получают сигналов, генерируемых терминалом. Системный вызов setpgid(2) устанавливает идентификатор группы процессов следующим образом:

pid == pgid создается группа процессов с идентификатором, равным pid; вызвавший процесс становится лидером этой группы

pid != pgid процесс pid становится членом группы процессов pgid, если она существует и принадлежит к этой сессии.

Если pid равен 0, будет использован идентификатор вызывающего процесса. Если pgid равен нулю, процесс с идентификатором pid станет лидером группы процессов. pid должен задавать процесс, принадлежащий к той же сессии, что и вызывающий.

Идентификатор группы процессов - атрибут, наследуемый порожденными процессами. Процесс может определить свой идентификатор группы, вызывая getpgrp(2) или getpgid(2).

Системный вызов waitid(2) и библиотечная функция waitpid(3C) могут использоваться для ожидания подпроцессов, принадлежащих определенной группе. Кроме того, можно послать сигнал всем процессам в заданной группе.

В ksh и bash для каждой исполняемой команды создается новая группа процессов.

В sh все процессы принадлежат к одной группе, если только сам процесс не исполнит setsid(2) или setpgid(2).

Установить идентификатор группы процессов - Пример

Этот пример показывает, как создать группу процессов, используя setpgid(2). Создаются три подпроцесса, и каждый распечатывает значение своего идентификатора группы процессов. Пример демонстрируется так:

```
$ setpgid
[6426] Original process group id: 179
[6426] New process group id: 6426

        [6427] Process group id: 6426

        [6428] Process group id: 6426

        [6429] Process group id: 6426
```

Эта выдача предполагает, что программа запущена из sh. Любой процесс, запущенный с управляющего терминала, принадлежит основной группе. Таким образом, процесс изначально принадлежит к группе основных процессов. Затем, в строке 14, он становится лидером группы процессов. Его подпроцессы наследуют новый идентификатор группы процессов, и принадлежат той же группе, что и их родитель. Эта новая группа процессов будет фоновой, и поэтому не будет получать сигналы, связанные с терминалом. Если программа исполняется из ksh или bash, вывод будет выглядеть так:

```
$ setpgid
[6426] Original process group id: 6426
[6426] New process group id: 6426

        [6427] Process group id: 6426

        [6428] Process group id: 6426

        [6429] Process group id: 6426
```

ksh создает новую группу процессов для каждой исполняемой команды. Поэтому setpgid(2) в строке 14 не делает ничего наблюдаемого, ведь процесс уже является лидером группы. Чтобы добиться более интересного поведения, можно сначала запустить sh, а только потом запускать программу, тогда при запуске программы лидером ее группы будет процесс sh.

`Файл: setpgid.c`

УСТАНОВИТЬ ИДЕНТИФИКАТОР ГРУППЫ ПРОЦЕССОВ - ПРИМЕР

```
 1 #include <sys/types.h>
 2 #include <unistd.h>
 3 #include <stdlib.h>
 4 #include <stdio.h>
 5 #define   NUMCHILD   3
 6
 7 main()
 8 {
 9     int i;
10
11     printf("[%ld] Original process group id: %ld\n",
12     getpid(), getpgid(0));
13
14     if (setpgid(0, 0) == -1) {
15     perror("");
16     exit(1);
17     }
18
19     printf("[%ld] New process group id: %ld\n",
20     getpid(), getpgid(0));
21
22     for (i = 0; i < NUMCHILD; i++ ) {
23     if (fork() == 0) {    /* child */
24         printf("\n\t[%ld] Process group id: %ld\n",
25         getpid(), getpgid(0));
26         exit(0);
27     }
28     }
29 }
```

Получение и установка группы процессов первого плана.

Функция tcsetpgrp(3C) устанавливает для терминала с дескриптором файла fildes идентификатор группы первого плана равным pgid. Помните, что процессы из основной группы получают сигналы, связанные с терминалом, такие как SIGINT и SIGQUIT. Если фоновый процесс попытается сделать tcsetpgrp(3C), он получит сигнал SIGTTOU. Например, если фоновый процесс попытается стать основным процессом, он получит этот сигнал. Фоновый процесс, однако, может проигнорировать или обработать этот сигнал.

Кроме того, фоновые процессы получают сигналы SIGTTIN и SIGTTOU при попытке читать с управляющего терминала или писать на него.

Командный процессор bash под SVR4 традиционно собирают с игнорированием SIGTTOU, поэтому фоновые процессы, запущенные из-под bash, не могут читать с терминала, но могут писать на него, так что их вывод может смешиваться с выводом текущей программы первого плана. Это сделано для совместимости с традиционными Unix-системами, где не было способа заблокировать вывод фоновых процессов.

tcgetpgrp(2) возвращает идентификатор группы основных процессов для терминала с дескриптором файла fildes.

tcgetsid(2) возвращает идентификатор сессии, для которой управляющим терминалом является терминал с дескриптором файла fildes.

Пример - Группа первого плана, связанная с терминалом

Эта программа демонстрирует изменение группы процессов первого плана. Это одна из основных задач при управлении заданиями. Программа работает так:

11 Распечатывается начальный идентификатор группы для этого процесса.

13 Создается подпроцесс.

14-17 При помощи setpgid(2) создается новая группа.

19 Распечатывается новый идентификатор группы процессов.

21 Подпроцесс засыпает на 10 секунд.

26 Распечатывается идентификатор группы основных процессов. Родительский процесс принадлежит группе первого плана.

27 Так как родительский процесс принадлежит основной группе, во время исполнения вызова sleep(2), этот процесс может получать сигналы, связанные с терминалом. Например, пользователь может послать SIGINT и родительский процесс завершится.

28 Группа основных процессов изменяется вызовом tcsetpgrp(2). Теперь порожденный процесс принадлежит к основной группе.

30 Все связанные с терминалом сигналы будут получены подпроцессом и вызовут его завершение.

Из-под ksh или bash эта программа исполняется следующим образом:

```
$ tcsetpgrp
Original PGID: 8260
New PGID: 8376
Foreground PGID: 8260  (terminal signals received by parent)
Foreground PGID: 8376  (terminal signals received by child)
child done
parent done
```

Из-под sh, shell необходимо снова сделать основным процессом прежде, чем наш процесс завершится. Иначе пользователь будет выброшен из системы при завершении программы.

Файл: tcsetpgrp.c

ПРИМЕР - ГРУППА ОСНОВНЫХ ПРОЦЕССОВ

```
 1 #include <sys/types.h>
 2 #include <unistd.h>
 3 #include <stdlib.h>
 4 #include <termios.h>
 5 #include <stdio.h>
 6
 7 main()
 8 {
 9    pid_t pid;
10
11    printf("Original PGID: %ld\n", getpgid(0));
12
13    if ((pid = fork()) == 0) {
14       if (setpgid(0, 0) == -1) {
15       perror("");
16       exit(1);
17       }
18
19       printf("New PGID: %ld\n", getpgid(0));
20
21       sleep(10);
22       printf("child done\n");
23       exit(0);
24    }
25
26    printf("Foreground PGID: %ld\n", tcgetpgrp(0));
27    sleep(5);  /* parent receives terminal signals */
28    tcsetpgrp(0, pid);
29    printf("Foreground PGID: %ld\n", tcgetpgrp(0));
30    wait(0);  /* child receives terminal signals */
31    printf("done parent\n");
32 }
```