1 Управление терминалом

Терминалом мы назовём устройство (или программу), которое позволяет пользователю вводить символы (обычно с клавиатуры) и видеть символы, выводимые компьютером (обычно на экране). Нас интересуют так называемые *алфавитно-цифровые* терминалы, то есть устройства, позволяющие вводить и выводить некоторый (достаточно ограниченный) диапазон символов и не поддерживающие графику. Существует очень большое количество видов (моделей) терминалов, начиная от электрической печатающей машинки, подключённой к компьютеру, и заканчивая современными X-терминалами. Некоторые терминалы являются программами (например, программа telnet для Windows), которые эмулируют работу некоторой модели аппаратного терминала.

Традиционно терминалы подключались к системам по асинхронным последовательным каналам связи, например, по стандарту RS-232. Хотя сейчас таких терминалов почти не осталось, интерфейс управления терминалами в системах Posix содержит средства управления последовательным портом компьютера, которые имеют мало смысла, например, для сетевого соединения.

Работа с терминалом ведётся через открытый файловый дескриптор терминала. Как правило, этот открытый файловый дескриптор процесс наследует от интерпретатора командной строки. И стандартный поток ввода, и стандартный поток вывода у процесса обычно связаны с одним и тем же терминалом.

За время существования системы Unix было разработано большое количество различных типов терминалов. Как правило, производитель каждой марки машины делал свой тип терминалов, который был несовместим с другими типами. Поскольку к машине, особенно, если она включена в компьютерную сеть, может быть одновременно подключено (физически или через сеть) несколько различных типов терминалов, система должна уметь работать с различными несовместимыми типами терминалов. Чтобы определить тип терминала, с которого работает пользователь, используется переменная окружения `TERM`. Например, сразу после входа пользователя в консоль Linux переменная `TERM` установлена в `linux`. После запуска X-сервера и вызова эмулятора терминала xterm тип терминала устанавливается в `xterm`.

Ядро операционной системы, если пользователь работает с физическим терминалом, или эмулятор терминала, если пользователь работает с виртуальным терминалом, специальным образом обрабатывает некоторые вводимые пользователем комбинации клавиш. Например, нажатие Ctrl-C вызывает посылку сигнала `SIGINT` всем процессам основной группы этого терминала (подробнее об этом можно прочитать в разделе «Управление заданиями»). Приложение может отменить специальную обработку таких комбинаций клавиш.

1.1 Очереди ввода/вывода

Поскольку терминал обычно является достаточно медленным устройством (если терминал подсоединён по последовательной линии, типичная скорость связи составляет 9600 или 19200 бод), ядро операционной системы дополнительно буферизует ввод и вывод. Эта буферизация не зависит от буферизации высокоуровневых потоков стандартной библиотеки языка Си.

Очередь ввода терминала хранит символы, которые были получены от терминала, но ещё не были прочитаны никаким процессом. Размер очереди ввода терминала определяется макросами `MAX_INPUT` и `_POSIX_MAX_INPUT`, определёнными в файле `<limits.h>`. Гарантируется, что очередь ввода имеет минимальный размер `MAX_INPUT`, но на самом деле очередь может быть больше или даже динамически менять размер. Если включено управление входным потоком (с помощью бита IXOFF режимов ввода), драйвер терминала передаёт на терминал символы STOP и START, когда входная очередь переполняется. В противном случае вводимые символы могут быть потеряны, если они поступают быстрее, чем успевают обрабатываться. В каноническом режиме ввода все введённые символы остаются в очереди до тех пор, пока не поступит символ перевода строки, поэтому очередь ввода терминала может переполниться, когда пользователь вводит слишком длинную строку.

Очередь вывода терминала аналогична очереди ввода, но она содержит символы, которые были записаны на терминал процессами, но ещё не были на него переданы. Если контроль выходного потока включён (установкой бита IXON режимов ввода), драйвер терминала подчиняется символам START и STOP, переданными терминалом, чтобы приостановить и возобновить пересылку данных.

Очистка очереди ввода терминала — это сброс всех символов, которые были получены от терминала, но ещё не были прочитаны процессами. Аналогично, очистка очереди вывода терминала — сброс всех символов, которые были записаны процессами, но ещё не переданы на терминал.

1.2 Канонический и неканонический ввод

Системы **POSIX** поддерживают два основных режима ввода с терминала: канонический и неканонический.

В *каноническом режиме* ввода с терминала обрабатывается построчно, каждая строка завершается символами перевода строки `'\n'`, EOF или EOL. Процесс не может считать вводимые символы, пока пользователь не введёт целую строку, а функция `read` возвращает самое большее одну строку ввода вне зависимости от того, сколько байт было запрошено.

В каноническом режиме ввода некоторые средства для редактирования строки предоставляет само ядро операционной системы. Некоторые символы, такие как ERASE или KILL, обрабатываются специальным образом и обозначают операции редактирования текущей вводимой строки.

Константы `_POSIX_MAX_CANON` и `MAX_CANON` параметризуют максимальную длину строки, вводимой в каноническом режиме. Гарантируется, что максимальная длина строки не меньше значения этих констант, но на самом деле она может быть больше и даже динамически менять размер.

В неканоническом режиме ввода символы не группируются в строки, а символы ERASE и KILL не обрабатываются. Частота (гранулярность), с которой считываются символы, задаётся параметрами MIN и TIME.

Большинство программ используют канонический режим ввода, поскольку это даёт возможность пользователю редактировать вводимые строки. Неканонический режим обычно используется тогда, когда программа либо обрабатывает односимвольные команды, либо предоставляет свои средства редактирования строки.

Выбор между каноническим и неканоническим режимом управляется флагом ICANON поля `c_lflag` структуры `struct termios`.

1.3 Функции низкоуровневого управления терминалом

В данном разделе рассматриваются некоторые функции низкоуровневого управления терминалом.

```c
#include <termios.h>
#include <unistd.h>

int tcgetattr(int fd, struct termios *termios_p);
int tcsetattr(int fd, int action, struct termios *termios_p);
int tcdrain(int fd);
int tcflush(int fd, int queue);
int tcflow(int fd, int action);
```

Функции `tcgetattr` и `tcsetattr` считывают и модифицируют режимы работы терминала. В качестве параметра им передаётся адрес переменной типа `struct termios`. Эта структура содержит всю информацию о режимах работы терминала и имеет следующие поля:

```c
struct termios
{
  tcflag_t c_iflag;       /* режимы ввода */
  tcflag_t c_oflag;       /* режимы вывода */
  tcflag_t c_cflag;       /* управляющие флаги */
  tcflag_t c_lflag;       /* локальные флаги */
  cc_t c_cc[NCCS];        /* управляющие символы */
};
```

Данное определение структуры не является точным, для каждой конкретной системы структура может содержать дополнительные поля. Типы `tcflag_t`, `cc_t` и макрос `NCCS` определяются системой, и их конкретное значение нас не будет интересовать. Назначение каждого поля будет рассмотрено ниже.

Хотя в функциях `tcgetattr` и `tcsetattr` терминал указывается с помощью файлового дескриптора, атрибуты соответствуют терминалу, а не файловому дескриптору. Это значит, что изменения атрибутов терминала имеют постоянный эффект, если некоторый другой процесс откроет тот же самый терминал, для него будут действовать изменённые атрибуты.

Аналогично, если один и тот же процесс имеет несколько файловых дескрипторов, ассоциированных с терминалом (либо полученных через `open`, либо через `dup`), изменение атрибутов терминала на одном файловом дескрипторе затрагивает ввод и вывод на всех файловых дескрипторах, ассоциированных с этим терминалом. Это значит, что нельзя одновременно открыть один файловый дескриптор для чтения с терминала в нормальном построчно-буферизируемом режиме с эхом (канонический режим), а другой файловый дескриптор для чтения с терминала в посимвольном режиме без эха. Вместо этого необходимо явно переключать терминал между этими двумя режимами.

Функция `tcgetattr` возвращает информацию о терминале по открытому файловому дескриптору `fd`, связанному с терминалом. Адрес структуры, в которую будет записана информация, должен содержаться в параметре `termios_p`. При успешном завершении функция возвращает 0, а при ошибке — `-1`. Переменная `errno` может содержать следующие коды ошибок:

`EBADF` Аргумент `fd` не является допустимым файловым дескриптором.

`ENOTTY` Файловый дескриптор `fd` не ассоциирован с терминалом.

Функция `tcsetattr` устанавливает атрибуты терминала, ассоциированного с файловым дескриптором `fd`. Адрес структуры, содержащей новые атрибуты терминала, передаётся в параметре `termios_p`. Параметр `action` указывает, как нужно поступить с входными и выходными символами, находящимися в очередях терминала. Он может принимать следующие значения:

TCSANOW — Изменения вступят в силу немедленно.

TCSADRAIN — Изменения вступят в силу после того, как все символы в выходной очереди терминала будут переданы на терминал. Этот флаг должен использоваться, когда устанавливаются режимы работы, затрагивающие вывод на терминал.

TCSAFLUSH — Аналогично TCSADRAIN, но дополнительно сбрасывает очередь ввода с терминала.

Функция `tcdrain` приостанавливает выполнение процесса до тех пор, пока все символы из выходной очереди терминала, ассоциированного с файловым дескриптором `fd`, не будут переданы на терминал.

Функция `tcflush` сбрасывает данные из очередей терминала, ассоциированного с файловым дескриптором `fd`. Параметр `queue` может принимать следующие значения:

TCIFLUSH — Сбрасывает входную очередь терминала.

TCOFLUSH — Сбрасывает выходную очередь терминала.

TCIOFLUSH — Сбрасывает и входную, и выходную очереди терминала.

Функция `tcflow` приостанавливает приём данных с терминала или передачу данных на терминал в зависимости от значения параметра `action`.

TCOOFF — Приостанавливает вывод на терминал.

TCOON — Возобновляет вывод на терминал.

TCIOFF — Передаёт на терминал символ STOP, который вызывает приостановку передачи вводимых символов от терминала к системе.

TCION — Передаёт на терминал символ START, который возобновляет передачу вводимых символов от терминала к системе.

1.3.1 Режимы ввода с терминала

В этом разделе рассматриваются флаги, управляющие относительно низкоуровневыми деталями обработки ввода с терминала: обработка ошибок чётности, сигналы прерывания, управление потоком данных и т. д. Все эти флаги — биты в поле `c_iflag` структуры `struct termios`. Это поле имеет один из целых типов, поэтому флаги можно модифицировать побитовыми операциями `&`, `|`, `^`. Никогда не задавайте значение `c_iflag` целиком, изменяйте только нужные флаги, а остальные биты не трогайте.

ISTRIP — Если этот бит установлен, у всех входных байтов обнуляется старший бит.

IGNBRK — Если установлен этот бит, сигнал прерывания игнорируется. Сигнал прерывания в асинхронной последовательной передаче данных определяется как последовательность нулевых бит длины, большей чем байт.

BRKINT — Если этот бит установлен, а бит IGNBRK сброшен, сигнал прерывания сбрасывает входную и выходную очереди терминала и генерирует сигнал SIGINT для основной группы процессов, связанной с этим терминалом. В противном случае сигнал прерывания передаётся в программу как символ `'\0'`.

IGNCR — Если установлен этот бит, символы возврата каретки (`'\r'`) при вводе игнорируются.

ICRNL — Если установлен этот бит, а IGNCR сброшен, символы возврата каретки (`'\r'`) передаются в программу как символы перевода строки `'\n'`.

INLCR — Если установлен этот бит, символы перевода строки передаются в программу как символы возврата каретки.

IXOFF Если этот бит установлен, управление приостановкой/продолжением передачи на входном потоке включено. Другими словами, компьютер посылает символы START и STOP, когда необходимо предотвратить переполнение входной очереди терминала. Оборудование терминала должно приостановить посылку символов при получении STOP и возобновить посылку вводимых символов при получении START.

IXON Если этот бит установлен, управление приостановкой/продолжением передачи на выходном потоке включено. Другими словами, когда компьютер получает символ STOP, он приостанавливает передачу символов до тех пор, пока не получит символ START. Символы START и STOP никогда не будут переданы в прикладную программу. Если бит не установлен, символы START и STOP передаются в программу как обычные символы.

1.3.2 Локальные режимы терминала

В этом разделе рассматриваются флаги поля c_lflag структуры struct termios. Эти флаги, как правило, управляют более высокоуровневыми деталями обработки вводимых символов, чем флаги, описанные в предыдущем разделе.

Все эти флаги — биты в поле c_lflag структуры struct termios. Это поле имеет один из целых типов, поэтому флаги можно модифицировать побитовыми операциями &, |, ^. Никогда не задавайте значение c_lflag целиком, изменяйте только нужные флаги, а остальные биты не трогайте.

ICANON Если этот бит установлен, ввод с терминала происходит в каноническом режиме. В противном случае ввод работает в неканоническом режиме.

ECHO Если этот бит установлен, включён режим отображения вводимых символов на терминале (эхо-режим).

ISIG Если этот бит установлен, распознаются символы INTR, QUIT и SUSP, которые посылают сигналы SIGINT, SIGQUIT и SIGTSTP соответственно основной группе процессов, ассоциированной с данным терминалом. Работа терминала в каноническом или неканоническом режиме не влияет на специальную обработку этих символов.

Использовать запрещение обработки сигнальных символов нужно очень осторожно, поскольку в этом случае программа не сможет быть интерактивно прервана и будет недружественной пользователю. Программа тогда должна предоставлять альтернативные способы, эквивалентные нажатию этих специальных символов.

NOFLSH Обычно символы QUIT, INTR, SUSP вызывают сброс входной и выходной очереди терминала. Если этот бит установлен, очереди не сбрасываются.

TOSTOP Если установлен этот бит, и система поддерживает управление заданиями, сигнал SIGTTOU будет послан фоновой группе процессов, если один из процессов группы пытается записать на терминал.

1.3.3 Специальные символы терминала

В каноническом режиме ввода терминал распознаёт некоторые специальные символы, которые выполняют различные управляющие функции. Например, символ ERASE (обычно <Del>) для редактирования вводимой строки и некоторые другие символы. Символ INTR (обычно Ctrl-C) для посылки сигнала SIGINT и другие символы, генерирующие сигнал, могут работать и в каноническом, и в неканоническом режиме. Все эти символы описаны в данном разделе.

Собственно используемые символы задаются в поле c_cc структуры struct termios. Это поле — массив, в котором каждый элемент задает символ, выполняющий соответствующую функцию. Каждый элемент имеет символическую константу, которая обозначает индекс этого элемента в массиве. Например, VINTR — индекс элемента, задающего символ INTR, поэтому присваивание значения '=' элементу t.c_cc[VINTR] устанавливает символ = как символ прерывания.

На некоторых системах каждый специальный символ можно отменить, присвоив значение _POSIX_VDISABLE соответствующему элементу массива. Это значение не равно никакому возможному коду символа. Этот макрос должен определяться в файле <unistd.h>, и если он определён, такая функция поддерживается системой.

	Символы редактирования строки
VEOF	Это — индекс символа EOF в массиве специальных символов структуры struct termios. Символ EOF распознаётся только в каноническом режиме. Если символ EOF введён не в начале строки, он завершает ввод строки, как символ '\n'. Если символ EOF введён в начале строки, функция read при чтении с терминала вернёт 0, обозначая конец файла, а сам символ EOF будет сброшен. Обычно символ EOF равен Ctrl-D.
VEOL	Индекс символа EOL в массиве специальных символов. Символ EOL распознаётся только в каноническом режиме ввода. Как и символ перевода строки он обозначает конец ввода строки. Символ EOL не сбрасывается, а считывается последним символом вводимой строки. Чтобы сделать <RET> символом конца ввода строки, не обязательно задавать символ EOL, а достаточно установить флаг ICRNL. На самом деле, именно такой режим включён по умолчанию.
VERASE	Индекс символа ERASE в массиве специальных символов. Символ ERASE распознаётся только в каноническом режиме работы терминала. Когда пользователь вводит этот символ, предыдущий введённый символ уничтожается. Сам символ ERASE тоже уничтожается. Символ ERASE не может использоваться для стирания символов в предыдущих введённых строках. Обычно символ ERASE равен <DEL>.
VKILL	Индекс символа KILL в массиве специальных символов. Символ KILL распознаётся только в каноническом режиме ввода. Когда пользователь вводит символ KILL, вся текущая вводимая строка уничтожается. Символ KILL при этом тоже уничтожается. Обычно символ KILL равен Ctrl-U.
	Символы посылки сигналов
VINTR	Индекс символа INTR в массиве специальных символов. Ввод символа INTR вызывает посылку сигнала SIGINT всем процессам основной группы процессов, ассоциированной с данным терминалом. Символ INTR затем сбрасывается. Обычно символ INTR равен Ctrl-C.
VQUIT	Индекс символа QUIT в массиве специальных символов. Ввод символа QUIT вызывает посылку сигнала SIGQUIT всем процессам основной группы процессов, ассоциированной с данным терминалом. Символ QUIT затем сбрасывается. Обычно символ QUIT равен Ctrl-\.

VSUSP	Индекс символа SUSP в массиве специальных символов. Символ SUSP распознаётся, только если система поддерживает управление заданиями. Если пользователь вводит этот символ, всем процессам текущей основной группы процессов, ассоциированной с данным терминалом, посылается сигнал SIGTSTP. Символ SUSP затем сбрасывается. Обычно символ SUSP равен Ctrl-Z.
Старт/стоп-символы	
Эти символы могут распознаваться и в каноническом, и в неканоническом режиме ввода. Их использование управляется флагами IXON и IXOFF режима ввода терминала. Конкретное значение этих символов может диктоваться аппаратурой подключённого терминала, поэтому иногда изменение этих символов не имеет никакого эффекта.	
VSTART	Индекс символа START в массиве специальных символов. Символ START используется, чтобы поддерживать режимы ввода IXON и IXOFF. Если флаг IXON установлен, получение символа START от терминала возобновляет передачу символов на терминал, сам символ START сбрасывается. Если установлен флаг IXANY, получение любого символа возобновляет приостановленную передачу на терминал, при этом символ не сбрасывается, если только это не символ START. Если установлен флаг IXOFF, система может передавать символ START на терминал, чтобы возобновить получение вводимых символов. Обычно символ START равен Ctrl-Q.
VSTOP	Индекс символа STOP в массиве специальных символов. Символ STOP используется, чтобы поддерживать режимы ввода IXON и IXOFF. Если установлен флаг IXON, получение символа STOP вызывает приостановку передачи символов на терминал, сам символ STOP при этом сбрасывается. Если установлен флаг IXOFF, система передаёт на терминал символ STOP, чтобы предотвратить переполнение очереди ввода с терминала. Обычно символ STOP равен Ctrl-S.

1.3.4 Неканонический ввод

В неканоническом режиме ввода специальные символы редактирования, такие как ERASE или KILL не имеют специального значения. Системное редактирование строки отключено, поэтому все вводимые символы, кроме, возможно, специальных символов посылки сигналов или управления потоком данных, передаются прикладной программе точно в том виде, как они набраны. Прикладная программа должна сама предоставлять средства для редактирования ввода, если это необходимо.

Неканонический режим предоставляет специальные параметры MIN и TIME для управления тем, сколько ждать ввода символов, и нужно ли ждать вообще или возвращаться немедленно, даже если нет введённых символов. Параметры MIN и TIME хранятся в массиве c_cc. VMIN и VTIME — имена индексов соответствующих элементов массива.

VMIN — Индекс значения MIN в массиве специальных символов. Значение MIN имеет смысл только в неканоническом режиме ввода. Оно задаёт минимальное количество байт в очереди ввода терминала, необходимое, чтобы read завершился.

VTIME — Индекс значения TIME в массиве специальных символов. Значение TIME имеет смысл только в неканоническом режиме ввода. Оно задаёт максимальное время, которое read будет ждать до возврата из системного вызова. Время задаётся в десятых долях секунды (то есть, 10 означает 1 секунду).

Значения MIN и TIME вместе определяют критерий, когда системный вызов read завершает работу. Точный смысл зависит от того, какой из этих параметров не равен 0. Возможны четыре случая.

- И `TIME` и `MIN` не равны 0. В этом случае `TIME` задаёт, сколько времени после получения очередного введённого символа ждать, придёт ли ещё один символ. После того, как был считан первый символ, `read` ждёт, пока либо не поступят `MIN` байтов, либо за время `TIME` не поступит ни одного нового символа.

  `read` всегда дожидается поступления первого символа, даже если интервал времени `TIME` закончится раньше. `read` может вернуть больше, чем `MIN` символов, если столько символов окажется в очереди ввода терминала.

- И `MIN`, и `TIME` оба равны 0. В этом случае `read` всегда завершается немедленно, возвращая столько символов, столько в этот момент находится в очереди ввода терминала, но не больше чем число байт, заданное в параметре `read`. Если очередь ввода пуста, `read` возвращает 0.

- `MIN` равно 0, но `TIME` не равно 0. В этом случае `read` ждёт поступления символов время `TIME`. Поступления единственного символа достаточно, чтобы удовлетворить запрос на чтение, и завершить `read`. При выходе `read` возвращает столько символов, сколько было доступно в очереди ввода, но не более числа байт, указанного аргументом `read`. Если за время `TIME` в очереди ввода не появилось символов, `read` возвращает 0.

- `TIME` равно 0, но `MIN` не равно 0. В этом случае `read` ждёт, пока в очереди ввода не появится как минимум `MIN` байтов. После этого `read` возвращает столько символов, сколько доступно в очереди, но не более числа байт, указанного аргументом `read`. Таким образом, `read` может вернуть больше, чем `MIN` символов, если столько символов оказалось в очереди ввода терминала.

Например, если параметр `MIN` установлен в 50, а системный вызов `read` запрашивает чтение 10 байт, `read` будет ждать, пока в очереди ввода не появится 50 байт. Затем `read` вернёт первые 10 байт, оставляя оставшиеся 40 в очереди ввода для последующих вызовов `read`.

1.4 Установка режимов терминала

Чтобы установить некоторый режим работы терминала, прежде всего нужно вызвать `tcgetattr`, чтобы получить текущий режим работы этого терминала. Затем нужно изменить только те поля, которые устанавливают нужный режим. После этого нужно установить режим работы терминала вызовом `tcsetattr`.

Не рекомендуется просто инициализировать структуру `struct termios` нужным набором значений и передать её непосредственно `tcsetattr`. Разные операционные системы могут иметь дополнительные поля, которые здесь не задокументированы, поэтому единственный способ, как можно избежать установки этих полей в неправильные значения, — не модифицировать их.

Для правильной работы различные терминалы могут требовать различных установок режимов работы терминала, поэтому не рекомендуется копировать атрибуты с одного терминала на другой терминал.

Когда поле структуры является набором независимых флагов, как поля `c_iflag`, `c_oflag`, `c_cflag` и `c_lflag`, не рекомендуется устанавливать значение поля целиком, поскольку у каждой операционной системы могут быть дополнительные флаги. Вместо этого нужно модифицировать только те биты флагов, которые необходимы программе, оставляя другие в неприкосновенности.

1.5 Пример неканонического ввода

Следующая программа переводит терминал в неканонический режим, отключает эхо и вводит символы до тех пор, пока не будет введён символ с кодом 4 (Ctrl-D).

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <signal.h>

/* переменная для сохранения исходных атрибутов терминала */
struct termios saved_attributes;

void reset_input_mode(void)
{
  tcsetattr(0, TCSANOW, &saved_attributes);
}
void sighnd(int signo)
{
  exit(0);
}
void set_input_mode(void)
{
  struct termios tattr;

  /* проверяем, что вводим с терминала */
  if (!isatty(0)) {
    fprintf(stderr, "Not␣a␣terminal.\n");
    exit(1);
  }
  /* считываем текущий режим работы терминала */
  tcgetattr(0, &saved_attributes);
  memcpy(&tattr, &saved_attributes, sizeof(tattr));
  /* включаем неканонический режим без эха */
  tattr.c_lflag &= ~(ICANON|ECHO);
  /* считываем минимум по одному символу */
  tattr.c_cc[VMIN] = 1;
  /* без ограничения времени ожидания */
  tattr.c_cc[VTIME] = 0;
  /* устанавливаем новый режим работы терминала */
  tcsetattr(0, TCSAFLUSH, &tattr);
}
int main(void)
{
```

```c
  char c;

  set_input_mode();
  atexit(reset_input_mode);
  /* устанавливаем обработку сигналов завершения */
  signal(SIGINT, sighnd);
  signal(SIGTERM, sighnd);

  while (1) {
    read(0, &c, 1);
    if (c == '\004') break;
    write(1, &c, 1);
  }

  return 0;
}
```

1.6 Вывод на терминал

Сам по себе вывод на терминал не имеет никаких особенностей по сравнению, например, с выводом в файл. Можно использовать системный вызов `write`, можно использовать функции высокоуровневого вывода `fprintf` и т. д. При использовании высокоуровневых функции вывода может потребоваться отключить буферизацию потока вывода, чтобы данные передавались на терминал сразу, а не накапливались во внутреннем буфере потока. Для этого можно использовать вызов `setbuf(f, NULL);`, где `f` — имя высокоуровневого потока, связанного с терминалом (обычно это `stdout`). Если буферизация вывода отключена, можно безопасно перемешивать использование низкоуровневой функции `write` и высокоуровневых функций работы с потоком.

Поскольку, как было сказано выше, существует большое количество разных моделей терминалов, основная проблема состоит в том, как сделать программу не привязанной к некоторой конкретной модели терминала. Об этом пойдет речь в следующем разделе.

1.7 Система команд терминала

Разные модели терминалов могут иметь разный набор поддерживаемых режимов вывода символов. Например, матричные принтеры могут поддерживать несколько шрифтов, верхние и нижние индексы и другие возможности. Стандартом де-факто для системы команд матричных принтеров является система команд Epson, но каждая конкретная модель принтера вносит какие-то добавления в эти команды. Точно также, стандартом де-факто для алфавитно-цифровых терминалов с прямой адресацией экрана (то есть, терминалов, позволяющих выводить символ в произвольное место экрана) является система команд VT100, но каждая конкретная модель терминала добавляет какие-то новые команды.

Если программа будет работать только с одним каким-то видом терминала, такая программа не сможет получить широкого распространения. Более того, при разработке программы просто невозможно предусмотреть модели терминалов, которые могут появиться уже после того, как программа будет распространена.

В системах Unix эта проблема решается следующим образом. Вместе с системой поставляется файл или набор файлов, содержащий описание возможностей (terminal capability) всех «известных науке» терминалов. В ранних системах **BSD** файл описания терминалов назывался `/etc/termcap`, сейчас повсеместно используется другая форма описания, когда свойства терминала описываются для каждого терминала в отдельном файле, и все эти файлы находятся в подкаталогах каталога `/usr/share/terminfo` или `/usr/lib/terminfo`. Описание терминала содержит сведения о том, допускает ли терминал прямую адресацию, и какая последовательность символов для этого должна быть передана, сколько у терминала стобцов и строк и пр.

Алфавитно-цифровые терминалы, как правило, работают в побайтовом режиме, то есть каждый байт — это символ, который нужно вывести на текущую позицию экрана, либо некоторая команда управления терминалом. Кроме того терминал может обрабатывать многобайтные последовательности. Такие последовательности всегда начинаются с кода `'\033'` (`ESC`). Например, для терминала **VT100** последовательность `'\033'`, `'['`, `'2'`, `'J'` очищает экран.

Каждый символ, набираемый на клавиатуре, как правило, имеет однобайтный код, соответствующий его коду **ASCII**. Но некоторые клавиши, такие как клавиши перемещения курсора и некоторые другие, посылают многобайтную последовательность, которая тоже начинается с кода `'\033'` (`ESC`). Для того же терминала **VT100** клавиша «стрелка вверх» посылает последовательность байтов `'\033'`,`'['`,`'A'`.

1.8 Работа с библиотекой ncurses

Функции работы с терминалом находятся в библиотеке **curses**. Эта библиотека предоставляет возможности для управления терминалом как для вывода символов на экран, так и для чтения кодов символов с клавиатуры. В современных системах (**Linux**, **FreeBSD**) эта библиотека называется **ncurses**. Чтобы использовать эту библиотеку в программе, она должна подключать заголовочный файл `<ncurses.h>`, а при компоновке должна быть указана опция `-lncurses`.

Основным понятием библиотеки является *окно*, представляющее собой прямоугольную область на экране. При запуске программы создаётся *корневое окно* (root window), занимающее весь экран. Пользователь может создавать новые окна меньшего размера. К корневому окну можно обращаться по имени `stdscr`. При выводе в окно вывод отсекается по правой и по нижней границе окна, то есть, если выводится строка длины большей, чем ширина окна, строка будет обрезана по ширине окна.

Все функции, работающие с окнами, не выводят непосредственно на терминал, а модифицируют буфер окна, находящийся в памяти. Чтобы содержимое буфера отобразилось на терминал, нужно явно вызвать одну из функций перерисовки терминала. При перерисовке окна библиотека старается минимизировать количество данных, пересылаемых на терминал (как было сказано выше, предполагается, что терминал связан с компьютером достаточно медленным каналом), и выполняет оптимизации перерисовки, исходя из своих представлений о текущем содержимом экрана терминала.

Заметьте, что окна сами по себе не имеют иерархии окно-подокно и не поддерживают перекрытия друг друга. Для этих целей применяются *панели*.

1.8.1 Начало работы программы

В начале работы программа, использующая библиотеку **ncurses**, должна выполнить несколько инициализационных вызовов. Типичное начало программы, использующей библиотеку `ncurses`, выглядит следующим образом:

```
if (!(initscr())) return 1;

cbreak();
noecho();
nonl();
meta(stdscr, TRUE);
intrflush(stdscr, FALSE);
keypad(stdscr, TRUE);

if (has_colors()) {
  start_color();
  init_pair(1, COLOR_WHITE, COLOR_BLUE);
  init_pair(2, COLOR_YELLOW, COLOR_BLUE);
  init_pair(3, COLOR_BLUE, COLOR_WHITE);
  init_pair(4, COLOR_YELLOW, COLOR_RED);
  wattrset(stdscr, COLOR_PAIR(1));
  wbkgdset(stdscr, COLOR_PAIR(1));
}
clear();
refresh();
```

Функция `initscr` выполняет начальную инициализацию терминала и внутренних структур данных библиотеки. Если при инициализации возникла ошибка, на стандартный поток ошибок печатается диагностическое сообщение, а функция возвращает `NULL`. При успехе возвращается указатель на структуру, описывающую корневое окно (то есть возвращается значение, равное `stdscr`).

Дальнейшая группа вызовов функций включает «неканонический» режим работы терминала.

- Вызов `cbreak` отключает построчную буферизацию ввода и обработку символов очистки строки и удаления последнего символа.

- Вызов `noecho` отключает эхо-режим, то есть вывод всех вводимых символов обратно на терминал.

- Вызов `nonl` отключает автоматический перевод символа `'\r'` в символ `'\n'` при вводе и перевод `'\n'` в последовательность `'\r'`, `'\n'` при выводе.

- Вызов функции `meta` позволяет изменить режим обработки 8-го бита кодов символов при вводе-выводе. По умолчанию 8-й бит отбрасывается. Если второй параметр функции равен `TRUE`, 8-й бит искажатся не будет. Первый параметр функции игнорируется.

- Вызов функции `intrflush` позволяет изменить режим «быстрой» реакции на символы прерывания (Ctrl-C и др.). Если второй параметр функции равен `TRUE`, при поступлении символа прерывания все данные, ещё не переданные на терминал, будут потеряны. Это может приводить к тому, что после прерывания с клавиатуры внутренний буфер окна будет не совпадать с отображаемым на экране. Если второй параметр равен `FALSE`, используется медленная, но надёжная схема реакции на символы прерывания. Первый параметр функции игнорируется.

- Функция `keypad` позволяет изменить режим трансляции специальных многобайтных кодов, поступающих с терминала (например, кодов функциональных клавиш или клавиш управления курсором). Если второй параметр функции установлен в TRUE, вводимые с терминала последовательности символов, соответствующие специальным клавишам, преобразуются в специальные целые константы, которые возвращаются программе. Например, при нажатии клавиши «стрелка влево», программа получит константу KEY_LEFT. Если второй параметр функции равен FALSE, многобайтные последовательности символов при вводе не транслируются. Первый параметр функции игнорируется.

Следующая группа команд выполняется, если терминал поддерживает переключение цветов символов и фона. В этом случае функция has_colors возвращает ненулевое значение. Функция start_color включает цветной режим терминала (изначально он может быть выключен). Далее вызовы функции init_pair каждой паре (цвет символа, цвет фона) ставят в соответствие номер этой пары в палитре атрибутов. Функция wattrset устанавливает атрибут выводимых символов для последующих операций, а функция wbkgdset устанавливает атрибут, которым будет очищаться окно.

В нашем случае создаётся палитра атрибутов из четырёх элементов. Первый элемент палитры определяет белый цвет текста и синий цвет фона, второй — жёлтый цвет текста и синий цвет фона и т. д. Далее вызовом функции wattrset устанавливается атрибут номер 1 из палитры атрибутов для выводимых в корневое окно символов, то есть все последующие символы будут выводится белыми на синем фоне. Вызов функции wbkgdset устанавливает атрибут номер 1 из палитры атрибутов для очистки корневого окна, то есть при очистке окна (или его части) будет использоваться синий цвет фона.

Вызов функции clear очищает корневое окно (точнее, очищает его буфер в памяти), а вызов функции refresh синхронизирует состояние окна в памяти с состоянием окна на терминале.

1.8.2 Завершение работы программы

В конце работы программа должна восстановить стандартный режим работы терминала. Кроме того, она может очистить после себя экран. Для этих целей можно использовать следующую последовательность вызовов функций:

```
// установить стандартные атрибуты фона для окна stdscr
bkgdset(COLOR_PAIR(0));
// очистить окно stdscr
clear();
// синхронизовать буфер
refresh();
// установить все режимы терминала в исходное состояние
endwin();
```

1.8.3 Соглашения об именовании функций

Для многих действий, которые можно выполнить с окном, предусмотрены два варианта функций: с указанием окна и с окном stdscr по умолчанию. Те функции, которые требуют указания окна имеют префикс w в своём имени. Например, уже упомянутая функция bkgdset устанавливает атрибут фона для корневого окна stdscr, а чтобы установить атрибут фона у произвольного окна, необходимо использовать функцию wbkgdset.

Для функций вывода текста в окно помимо префикса w допускается префикс mv, который указывает, что соответствующая функция принимает дополнительно два параметра, указывающие координату позиции в окне, в которую должен выводиться текст. Координаты отсчитываются, как обычно, от левого верхнего угла окна, и первой всегда указывается строка, а за ней столбец, то есть пара (y,x). Точка начала отсчёта имеет координаты $(0,0)$. Например, функция printw выводит форматную строку в текущую позицию корневого окна, функция wprintw выводит форматную строку в текущую позицию заданного окна, функция mvprintw выводит форматную строку в указанную позицию корневого окна stdscr, и, наконец, функция mvwprintw выводит форматную строку в указанную позицию указанного окна.

1.8.4 Синхронизация окна и терминала

Как уже было упомянуто, для каждого окна в памяти хранится его текущий образ. Все функции работы с окном модифицируют этот образ окна в памяти. Для того, чтобы изменённое содержимое окна было показано на экране, необходим явный вызов одной из функции обновления экрана. Кроме того, для всего экрана терминала в памяти хранится его образ. Поэтому функции обновления окна refresh и wrefresh работают в две стадии: вначале изменённые ячейки окна копируются в виртуальный буфер экрана, затем изменённые ячейки виртуального буфера экрана выводятся на физический экран. Для более тонкого управления обновлением экрана могут оказаться полезными следующие функции:

```
int refresh(void);
int wrefresh(WINDOW *win);
int wnoutrefresh(WINDOW *win);
int doupdate(void);
int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);
```

Функция refresh копирует изменённые ячейки корневого окна stdscr в буфер экрана, затем изменённые ячейки буфера экрана выводятся на физический экран. Функция wrefresh копирует изменённые ячейки заданного окна в буфер экрана, затем изменённые ячейки буфера экрана выводятся на физический экран. Обратите внимание, что даже если содержимое окна изменилось, реального вывода на экран может не произойти, если новое содержимое окна совпадает с тем, что находится в данной ячейке буфера экрана.

При работе программы может возникнуть ситуация, когда реальное содержимое экрана не соответствует буферу экрана. Например, программа может запустить другую программу, которая что-то вывела на экран. В этом случае предусмотрены средства для принудительного обновления физического экрана. Так, функция wredrawln позволяет указать, что строки экрана, соответствующие указанным строкам окна win должны быть обязательно перерисованы, поскольку их содержимое на экране не соответствует содержимому буфера экрана. Параметр beg_line указывает первую строку окна, а параметр num_lines позволяет указать количество строк[1]. Функция redrawwin задаёт, что всё окно win должно быть перерисовано. И функция wredrawln, и функция redrawwin не изменяют физического экрана, а манипулируют внутренними структурами библиотеки **ncurses**. Собственно обновление физического экрана будет выполнено при вызове refresh, wrefresh или doupdate.

[1] Заметим, что информация о рассинхронизации буфера экрана и физического экрана редко бывает такой точной.

Другой способ принудительного обновления экрана состоит в вызове функции `wrefresh` со специальным параметром `curscr`. В этом случае физический экран немедленно очищается и полностью перерисовывается заново.

Когда функция `wrefresh` используется для обновления нескольких окон подряд, операция вывода обновлённых ячеек буфера экрана на физический экран выполняется за короткий промежуток времени несколько раз. Ускорить операции можно, если выводить буфер экрана на физический экран только один раз в конце всех обновлений буфера экрана. В этом случае должны применяться функции `wnoutrefresh` и `doupdate`. Функция `wnoutrefresh` копирует изменённые ячейки заданного окна в буфер экрана, но не выводит буфер экрана на физический экран, а функция `doupdate` выводит все накопленные в буфере экрана изменения на физический экран. Таким образом, вызов функции `wrefresh` эквивалентен последовательному вызову `wnoutrefresh` и `doupdate`.

1.8.5 Очистка окна

Следующие функции очищают окно или его часть. Ячейки окна заполняются символами пробела. Атрибут символа (то есть цвет фона) можно изменить при помощи функции `wbkgdset`.

```
int erase(void);
int werase(WINDOW *win);
int clear(void);
int wclear(WINDOW *win);
int clrtobot(void);
int wclrtobot(WINDOW *win);
int clrtoeol(void);
int wclrtoeol(WINDOW *win);
```

Функции `erase` и `werase` записывают символ пробела в каждую ячейку окна, очищая его.

Функции `clear` и `wclear` делают то же, что и функции `erase`, `werase`, но дополнительно устанавливают флаг принудительного обновления физического экрана. Таким образом, при следующем вызове `refresh`, `wrefresh` или `doupdate` физический экран будет очищен и перерисован.

Функции `clrtobot` и `wclrtobot` очищают окно от текущей позиции курсора и до конца окна. Очищаются все символы справа от курсора, включая текущую позицию курсора, на текущей строке и все строки ниже текущей строки.

Функции `clrtoeol` и `wclrtoeol` очищают текущую строку окна справа от текущей позиции, включая её.

1.8.6 Вывод в окно

Строка выводится в окно с помощью одной из следующих функций:

```
int addstr(const char *str);
int addnstr(const char *str, int n);
int waddstr(WINDOW *win, const char *str);
int waddnstr(WINDOW *win, const char *str, int n);
int mvaddstr(int y, int x, const char *str);
int mvaddnstr(int y, int x, const char *str, int n);
int mvwaddstr(WINDOW *win, int y, int x, const char *str);
```

```
int mvwaddnstr(WINDOW *win, int y, int x, const char *str, int n);
```

Здесь win — окно, в которое выводится строка, y, x — координаты в окне, str — выводимая строка, n — максимальное количество выводимых символов. Если функция не принимает параметр win, строка добавляется в окно stdscr. Если функция не принимает параметры x, y, строка выводится, начиная с текущей позиции.

Если выводимая символьная строка не помещается на текущей строке окна вывода, остаток переносится на следующую строку окна вывода, которое при необходимости прокручивается. Если в качестве параметра n указать -1, будет выведена только та часть строки, которая умещается в текущей строке окна вывода. После вывода строки текущая позиция в окне соответствующим образом изменяется.

Для вывода одного символа используются следующие функции:

```
int addch(const char ch);
int waddch(WINDOW *win, const char ch);
int mvaddch(int y, int x, const char ch);
int mvwaddch(WINDOW *win, int y, int x, const char ch);
```

Функции позволяют задавать окно вывода win, координаты вывода y, x.

Для форматного вывода используются функции:

```
int printw(char *fmt, ...);
int wprintw(WINDOW *win, char *fmt, ...);
int mvprintw(int y, int x, char *fmt, ...);
int mvwprintw(WINDOW *win, int y, int x, char *fmt, ...);
int vw_printw(WINDOW *win, char *fmt, va_list varglist);
```

Параметр fmt — это строка, определяющая форматное преобразование вывода. Спецификации преобразования в точности совпадают со спецификациями преобразования функций семейства printf. Параметр win позволяет указать окно вывода, параметры y, x определяют координаты вывода.

1.8.7 Ввод с клавиатуры

```
int getch(void);
int mvgetch(int y, int x);
int ungetch(int ch);
int has_key(int ch);
```

Функция getch возвращает код очередной нажатой клавиши в буфере ввода терминала. Если включён неблокирующийся режим, и буфер ввода пуст, возвращается значение ERR (константа, определённая как -1). Если неблокирующийся режим не включён, и буфер ввода пуст, программа приостанавливается до поступления информации в буфер ввода. Функция mvgetch перемещает курсор в указанную позицию окна, затем выполняет функцию getch.

Если включён режим распознавания специальных клавиш, соответствующие им последовательности байтов распознаются и заменяются на целые значения, находящиеся вне диапазона допустимых кодов символов. Этим значениям соответствуют констнты KEY_, определённые библиотекой. Некоторые из таких специальных кодов клавиш перечислены в таблице 4. Обратите внимание, что последовательности байтов, кодирующие специальные клавиши, как правило, начинаются с байта \033. Этот же байт используется для кодирования клавиши ESC, поэтому нажатие на клавишу ESC клавиатуры будет передано в программу после

KEY_DOWN	Клавиша «стрелка вниз»
KEY_UP	Клавиша «стрелка вверх»
KEY_LEFT	Клавиша «стрелка влево»
KEY_RIGHT	Клавиша «стрелка вправо»
KEY_HOME	Клавиша **Home**
KEY_END	Клавиша **End**
KEY_IC	Клавиша **Insert**
KEY_DC	Клавиша **Delete**
KEY_NPAGE	Клавиша **Page Down**
KEY_PPAGE	Клавиша **Page Up**
KEY_BACKSPACE	Клавиша **Backspace**
KEY_F*n*	Функциональная клавиша **F**n, $0 \leqslant n \leqslant 63$

Таблица 4: Константы, соответствующие некоторым специальным клавишам

некоторого интервала времени, в течение которого библиотека ожидает поступления других байтов в последовательности, задающей специальную клавишу.

Функция `ungetch` возвращает один код клавиши обратно во входную очередь терминала. Следующий вызов `getch` вернёт этот код.

Функция `has_key` принимает код специальной клавиши, одной из определённых в таблице 4, и возвращает `TRUE` или `FALSE` в зависимости от того, распознаётся ли эта клавиша терминалом.

1.9 Пример программы

Приведённая ниже программа перемещает символ `'*'` по экрану. При компиляции в командной строке редактора связей нужно указать `-lcurses` — подключение библиотеки.

```
#include <curses.h>
#include <locale.h>

int main(void)
{
  int prev_x = 0, prev_y = 0, caret_y = 0, caret_x = 0, c;
  int flag = 1;

  setlocale(LC_ALL, "");

  if (!initscr()) return 1;
  cbreak();
  noecho();
  nonl();
  meta(stdscr, TRUE);
  intrflush(stdscr, FALSE);
  keypad(stdscr, TRUE);

  if (has_colors()) {
    start_color();
    init_pair(1, COLOR_WHITE, COLOR_BLUE);
  }
```

```
  attrset(COLOR_PAIR(1));
  bkgdset(COLOR_PAIR(1));
  clear();

  while (flag) {
    mvaddch(prev_y, prev_x, '␣');
    mvaddch(caret_y, caret_x, '*');
    move(caret_y, caret_x);
    refresh();
    prev_x = caret_x;
    prev_y = caret_y;
    c = getch();
    switch (c) {
    case 033:
      flag = 0;
      break;
    case KEY_UP:
      if (caret_y > 0) caret_y--;
      break;
    case KEY_DOWN:
      if (caret_y < LINES - 1) caret_y++;
      break;
    case KEY_LEFT:
      if (caret_x > 0) caret_x--;
      break;
    case KEY_RIGHT:
      if (caret_x < COLS - 2) caret_x++;
      break;
    }
  }

  bkgdset(COLOR_PAIR(0));
  clear();
  refresh();
  endwin();
  return 0;
}
```