16. Консольный ввод-вывод

Каждому, кто приступает к изучению программирования, консольный ввод/вывод представляется простым делом. Для создания графических программ необходимо сначала изучить язык программирования, а затем – один из интерфейсов программирования графики. Для создания консольных программ достаточно изучить язык программирования. Если вы не шли к изучению программирования окольными путями, то ваши первые программы наверняка были консольными. Однако интерфейс консоли Unix обладает значительно большими возможностями, чем распечатка строк и чтение данных с клавиатуры. О некоторых дополнительных возможностях работы с консолью мы и поговорим.

Богатство функций работы с консолью в Unix объясняется тем, что у Unix долгая история. Когда-то терминалы были настоящими устройствами (в некоторых областях человеческой деятельности такие устройства применяются и сейчас), подключенными к компьютеру через последовательный порт. Часто терминал и компьютер были отделены друг от друга прослойкой в виде пары модемов и телефонной линии.

Последовательный порт и модем считались неотъемлемой частью терминала, и интерфейс управления ими стал частью интерфейса управления терминалом. Кроме того, такие устройства как, например, принтеры, тоже считаются терминалами Unix. Если в графическом программировании принтер рассматривается как устройство вывода данных, подобное графическому дисплею, то вполне логично, что при программировании в текстовом режиме принтер считается разновидностью терминала. Таким образом, с точки зрения Unix, под понятие «терминала» подпадает множество устройств, которые работают совершенно по-разному.

Неудивительно, что для управления всем этим многообразием устройств потребовался сложный интерфейс. И хотя вы вряд ли подключаетесь к своему компьютеру с помощью модема (у вас, возможно, и модема-то уже нет), а для вывода документа на печать вы, скорее всего, используете фильтр PostScript, некоторые возможности управления терминалом, появившиеся в незапамятные времена, все еще могут пригодиться вашим программам.

Предотвращение перенаправления вывода

Вы, конечно, знаете, что в Unix вывод консольной программы, предназначенный монитору, может быть перенаправлен в файл, или в поток ввода другой программы. Часто это бывает не только полезно, но даже необходимо. Теперь можете проверить свою сообразительность. Отодвиньте эту статью на десять секунд и подумайте, в каких случаях программа не должна допускать перенаправление своего вывода? Самый простой пример – интерактивная консольная программа. Интерактивные программы выводят данные небольшими порциями, после чего останавливаются в ожидании реакции пользователя. Если пользователь пытается перенаправить вывод вашей интерактивной программы, значит он, скорее всего, просто не понял, как с ней работать. Желательно предотвратить бессмысленные действия и сообщить пользователю об этом. Рассмотрим сначала простейший пример программы, которая не позволяет перенаправить свой вывод на устройство, не являющееся терминалом (текст этой программы вы найдете на диске, в файле noredirect.c).

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <unistd.h>
#include  <string.h>
int main (int argc, char ** argv)
{
  char * errstr = "I will not redirect my output!\n";
  if (!isatty(fileno(stdout))) {
     write(2, errstr, strlen(errstr));
     return EXIT_FAILURE;
  }
  printf ("Hello!\n");
```

```
  return EXIT_SUCCESS;
}
```

Функция isatty(3) позволяет программе узнать, является ли предоставленное ей устройство ввода/вывода терминалом. Аргументом isatty() должен быть дескриптор файла устройства, который мы получаем из переменной stdout, пользуясь функцией fileno(3). Функция isatty() возвращает единицу, если переданный ей дескриптор соответствует терминалу и 0 – в противном случае. Программа noredirect напечатает строку “Hello!”, только если у нее есть доступ к терминалу. Если же вы скомандуете:

```
$ noredirect > file
```

На экране появится гневное сообщение, а файл file окажется пустым. Впрочем, возможно, вы не всегда будете столь строгим по отношению к пользователю. Вам может потребоваться, чтобы только некоторые части вывода вашей программы не допускали перенаправления. В этом случае можно воспользоваться трюком, который показан в программе noredirect2 (noredirect2.c на диске).

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <fcntl.h>
#include  <string.h>
int main (int argc, char ** argv)
{
  int fd_in, fd_out;
  char * nr_message = "Enter your name, please\n";
  char buf[255];
  fd_in = open("/dev/tty", O_RDONLY);
  fd_out = open("/dev/tty", O_WRONLY);
  write(fd_out, nr_message, strlen(nr_message));
  read(fd_in, buf, 255);
  printf("Your name is %s\n", buf);
  return EXIT_SUCCESS;
}
```

В этой программе мы напрямую обращаемся к устройству, которое соответствует управляющему терминалу ввода/вывода. Вы, конечно, помните, что устройства в Unix/Linux представлены файлами. Файл /dev/tty предоставляет программе доступ к ее управляющему терминалу. Иначе говоря, для каждой программы файл /dev/tty представляет тот терминал, который является управляющим терминалом данной программы (так, по крайней мере, обстоит дело в Linux). Мы открываем устройство /dev/tty дважды – для чтения и записи. Запись в файл /dev/tty соответствует выводу данных на экран терминала, а чтение – вводу данных с клавиатуры.

Полученные дескрипторы, fd_in и fd_out мы будем использовать, соответственно, для ввода/вывода, который нельзя перенаправить. Вся магия перенаправления ввода/вывода основана на том, что программы пользуются потоками ввода вывода, которые предоставляет им среда окружения. Если мы в программе свяжем дескрипторы с конкретным устройством, среда окружения не сможет на это повлиять. Таким образом, строка “Enter your name, please” всегда будет выводиться на терминал. С терминала же программа будет считывать ответ пользователя. В то же время, строка, распечатанная с помощью printf(), может быть перенаправлена на другое устройство, поскольку printf() использует поток вывода, предоставленный окружением (мы можем, в принципе, подменить поток, используемый printf() по умолчанию, но не будем этого делать). Вместо указания имени устройства /dev/tty можно воспользоваться функцией ctermid(3). Будучи вызвана с аргументом NULL, эта функция возвращает указатель на структуру FILE, соответствующую управляющему терминалу.

16.1. Управление терминалом

Управление терминалом осуществляется с помощью структуры termios, которая содержит значения и флаги, влияющие на различные параметры терминала. Прежде чем изучать структуру termios и функции, работающие с ней, необходимо сказать несколько слов о режимах работы терминала.

Поведение терминала во многом определяется тем, в каком режиме он находится – каноническом или неканоническом. Примером программы, использующей терминал в каноническом режиме, может служить оболочка bash. В каноническом режиме терминал передает программе символы, введенные пользователем, построчно, после того, как пользователь нажмет клавишу «Ввод». До тех пор, пока пользователь не нажал «Ввод», он может редактировать вводимую строку, используя клавиши BackSpace, Del, Tab и им подобные.

Если терминал находится в неканоническом режиме, пользователю не требуется нажимать «Ввод» для того, чтобы программа получила напечатанные им символы. В неканоническом режиме (примером использования которого может служить редактор vi) программа может получать символы сразу после ввода или с задержкой, по одному или по нескольку, в зависимости от настроек режима терминала.

Поскольку обычно в неканоническом режиме все введенные символы сразу передаются программе, возможности редактировать строку у пользователя отсутствуют. Независимо от режима, в котором находится терминал, можно установить некоторые другие его параметры, например, скорость передачи данных и функцию контроля ошибок (очень полезно при подключении терминала с помощью модема). Впрочем, некоторые из этих параметров, например отключение отображения вводимых пользователем символов (echo) или изменение времени ожидания ввода в неканоническом режиме, могут с пользой применяться и в современных программах.

Структура termios позволяет управлять флагами и численными параметрами, которые можно разделить на пять групп: ввод, вывод, управление оборудованием, локальные параметры и специальные управляющие символы. Простейшая структура termios состоит из пяти полей, соответствующих перечисленным группам:

```
struct termios {
  tcflag_t c_iflag;  // флаги управления вводом
  tcflag_t c_oflag;  // флаги управления выводом
  tcflag_t c_cflag;  // флаги управления оборудованием
  tcflag_t c_lflag;  // флаги управления локальными параметрами
  cc_t c_cc[NCCS]  // Специальные управляющие символы
};
```

У структуры termios могут быть и другие поля, но нас они не интересуют. Обычно работа со структурой termios происходит по следующему сценарию (все необходимые функции и типы данных определены в файле termios.h): с помощью функции tcgetattr(3) мы получаем копию структуры, описывающую текущее состояние терминала и делаем еще одну копию. Затем мы модифицируем значения полей одной из копий termios, так чтобы изменить нужные нам параметры терминала, и передаем системе новое значение termios с помощью функции tcsetattr(3).

После того, как работа с терминалом в измененном режиме закончена, мы восстанавливаем исходное состояние терминала с помощью сохраненной копии исходной структуры termios и функции tcsetattr(). Первым аргументом функции tcgetattr() должен быть дескриптор файла, соответствующего терминалу. Вторым аргументом является указатель на структуру termios, в которой функция возвращает текущие настройки терминала. Первым параметром функции tcsetattr() также служит дескриптор файла терминала. Второй параметр используется для передачи флагов, определяющих, когда изменения параметров терминала должны вступить в силу. Третьим параметром tcsetattr() является указатель на структуру termios, содержащую новые параметры.

Ключевой момент во всем этом, – модификация полей структуры termios. Первые четыре поля структуры содержат комбинации флагов, определяющих параметры терминала. Пятое поле представляет собой массив значений. Индексам этого массива соответствуют специальные константы, с помощью которых мы можем понять значение элементов массива. Рассмотрим сначала поля termios, содержащие флаги. Полное описание флагов (а их довольно много) вы найдете на странице man, посвященной termios. Я перечислю здесь только некоторые флаги, которые устанавливаются в поле c_lflag, поскольку они представляются мне наиболее интересными.

Флаг ECHO управляет отображением вводимых символов на экране монитора. Если он установлен, символы отображаются, в противном случае – нет. Флаг ECHOE делает то же, что и флаг ECHO, но только для управляющих символов, стирающих другие символы или строки (например, BackSpace). Поскольку неканонический режим не поддерживает редактирование строки, в этом режиме флаг ECHOE игнорируется. Если установлен флаг ICANON, терминал находится в каноническом режиме, в противном случае – в неканоническом. Флаг IEXTEN переводит терминал в режим расширенной обработки вводимых символов. От того, установлен ли флаг ISIG, зависит, будут ли специальные комбинации клавиш, такие как Ctrl-C и Ctrl-Z, инициировать соответствующие им сигналы.

Из констант, соответствующих индексам массива c_cc[], наибольший интерес представляют две – VMIN и VTIME. Чтобы объяснить важность этих параметров, рассмотрим подробнее работу терминала в неканоническом режиме. В каноническом режиме сигналом завершения ввода данных является нажатие клавиши «Ввод». Любая функция, считывающая данные с терминала, вернет управление вызвавшей ее программе только после того, как пользователь нажмет эту клавишу.

При работе терминала в неканоническом режиме дело обстоит сложнее. В этом режиме нет сигнала, который бы оповещал систему о том, что ввод данных окончен, и функция чтения данных должна вернуть управление. Поведение функций, читающих данные в этом режиме зависит от параметров termios.c_cc[VMIN] и termios.c_cc[VTIME].

Параметр termios.c_cc[VMIN] указывает минимальное число введенных символов, после которого функция, считывающая данные, может вернуть управление программе. Параметр termios.c_cc[VTIME] указывает максимальное время ожидания ввода (после ввода первого символа), по прошествии которого функция чтения данных возвращает управление, независимо от того, сколько символов было прочитано.

Если обоим этим параметрам присвоены значения, большие нуля, функция чтения данных вернет управление после того, как будет выполнено требование одного из параметров. Если функция получит количество символов, заданное в параметре termios.c_cc[VMIN] до истечения срока времени, заданного в параметре termios.c_cc[VTIME], она вернет управление и передаст программе соответствующее количество символов.

Если заданный срок времени истечет до того, как функция считает указанное ей количество символов, функция вернет управление и передаст программе те символы, которые она успела считать. При этом, как следует из сказанного выше, функция чтения данных вернет как минимум один символ. Если одному из параметров termios.c_cc[VMIN] или termios.c_cc[VTIME] присвоено нулевое значение, а другому – ненулевое, то условием возврата из функции чтения данных становится значение ненулевого параметра.

Наконец, если оба параметра имеют нулевые значения, функция чтения данных всегда будет возвращать управление немедленно. Если к моменту вызова функции чтения данных в потоке ввода были символы, буфер функции будет заполнен ими, иначе функция чтения данных вернет пустой буфер.

Вернемся к функции tcsetattr(). Во втором параметре этой функции может быть передан один или несколько нижеследующих флагов (для комбинации флагов используется оператор |):

- TCSANOW – изменения параметров терминала вступают в силу немедленно.
- TCSADRAIN – изменения параметров терминала вступают в силу после того, как все данные, записанные ранее в файл устройства, будут переданы самому устройству. Это

значение обычно используется, если новые значения полей termios изменяют параметры вывода данных.

- TCSAFLUSH - изменения параметров терминала вступают в силу после того, как все данные, записанные ранее в файл устройства, будут переданы самому устройству. Все данные, которые в этот момент были введены с клавиатуры, но еще не прочитаны программой, при этом теряются.
- TCSASOFT – этот флаг заставляет функцию игнорировать значения полей c_cflag, c_ispeed и c_ospeed структуры termios. Поля c_ispeed и c_ospeed управляют скоростью передачи входящих и исходящих данных. Но, поскольку мы не рассматриваем подключение терминала с помощью модема, эти поля нас не интересуют.

Настало время освежить теорию практикой. Рассмотрим два примера, в которых изменение свойств терминала может оказаться полезным. Первый пример, это консольная программа, предназначенная для ввода пароля. Как вы знаете, во время ввода пароля такие программы не отображают никаких символов. Теперь вы также догадываетесь, как они это делают – им достаточно передать системе структуру termois со сброшенным флагом ECHO. Программа passwdmode (файл passwdmode.c) демонстрирует эту технику:

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <termios.h>
#define BUF_SIZE 15
int main (int argc, char ** argv)
{
  struct termios oldsettings, newsettings;
  char password[BUF_SIZE+1];
  int len;
  sigset_t newsigset, oldsigset;
  sigemptyset(&newsigset);
  sigaddset(&newsigset, SIGINT);
  sigaddset(&newsigset, SIGTSTP);
  sigprocmask(SIG_BLOCK, &newsigset, &oldsigset);
  tcgetattr(fileno(stdin), &oldsettings);
  newsettings = oldsettings;
  newsettings.c_lflag &= ~ECHO;
  tcsetattr(fileno(stdin), TCSAFLUSH, &newsettings);
  printf("Enter password and press [Enter]\n");
  len = read(fileno(stdin), password, BUF_SIZE);
  password[len] = 0;
  tcsetattr(fileno(stdin), TCSANOW, &oldsettings);
  sigprocmask(SIG_SETMASK, &oldsigset, NULL);
  printf("Your password is %s\n", password);
  return EXIT_SUCCESS;
}
```

В начале программы мы блокируем сигналы SIGINT и SIGTSTP (зачем это нужно, рассмотрим чуть ниже). Затем, с помощью функции tcgetattr() мы заполняем переменную oldsettings типа struct termios текущими значениями параметров терминала. Далее мы копируем содержимое oldsettings в переменную newsettings. Строка newsettings.c_lflag &= ~ECHO;

Сбрасывает флаг ECHO в структуре newsettings. Остальные параметры терминала остаются без изменений. Далее, с помощью функции tcsetattr() мы устанавливаем новые параметры. Теперь терминал не будет выводить на экран символы, вводимые пользователем, и мы можем вызвать функцию, считывающую значение пароля. После этого программа восстанавливает прежнее состояние терминала. Теперь мы можем разблокировать заблокированные сигналы. Мы распечатываем строку с введенным «паролем» (не вздумайте вводить в программе какой-нибудь настоящий пароль, иначе злоумышленник, прячущийся за вашей спиной, обязательно его увидит).

Зачем мы блокировали сигналы вона время ввода пароля? Представьте себе, что в то время, когда программа ожидает ввода пароля, пользователь передумал и захотел завершить ее с помощью Ctrl-C. Если программа завершится в этот момент, состояние терминала не будет восстановлено, и символы, вводимые пользователем, по-прежнему не будут отображаться. Это не смертельно, но неудобно. Вот почему программы, ожидающие ввода пароля, временно блокируют некоторые сигналы.

Рассмотрим теперь другой случай. Представьте себе, что программа выполняет некую длительную операцию, и вы хотите, чтобы у пользователя была возможность прервать ее, не прибегая к таким средствам, как сигналы. Для этого программа может периодически проверять, не нажал ли пользователь клавишу выхода, например [q]. Однако если терминал находится в каноническом режиме, пользователю придется нажать еще и «Ввод», чтобы программа могла считать символ, и это не говоря о том, что в каноническом режиме функции чтения данных по умолчанию блокируют выполнение программы до появления данных. Нам нужно, чтобы функция ввода, например getchar(), проверяла наличие в потоке ввода символа q, причем без всякого дополнительного символа ввода, но возвращала управление немедленно, независимо от того, есть символ в потоке ввода или нет. Все это очень просто сделать, переведя терминал в неканонический режим, что демонстрирует программа pressq (файл pressq.c):

```
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
int main (int argc, char ** argv)
{
  struct termios oldsettings, newsettings;
  tcgetattr(fileno(stdin), &oldsettings);
  newsettings = oldsettings;
  newsettings.c_lflag &= ~(ECHO|ICANON|ISIG);
  newsettings.c_cc[VMIN] = 0;
  newsettings.c_cc[VTIME] = 0;
  tcsetattr(fileno(stdin), TCSANOW, &newsettings);
  while(getchar() != 'q') {
     sleep(1);
     printf("press [q] to quit\n");
  }
  tcsetattr(fileno(stdin), TCSANOW, &oldsettings);
  return EXIT_SUCCESS;
}
```

Мы получаем структуру с текущими параметрами терминала, и сохраняем ее, как и в предыдущем случае. Затем мы сбрасываем сразу три флага: ECHO, ICANON и ISIG. Что дает сброс флага ECHO, вы уже знаете. Мы не хотим, чтобы введенный пользователем символ команды отображался на экране.

Сброс флага ICANON переводит монитор в неканонический режим, а сброс флага ISIG приводит к тому, что ввод специальных символов Ctrl-C и Ctrl-Z не порождает соответствующих сигналов. Это еще один способ защитить программу от досрочного завершения в тот момент, когда параметры терминала изменены. Затем мы устанавливаем значения параметров newsettings.c_cc[VMIN] и newsettings.c_cc[VTIME]. Обоим параметрам присваивается значение 0. В результате функция getchar() всегда будет возвращать управление немедленно. Далее программа переходит в бесконечный цикл, из которого ее может вывести только появление символа q в потоке ввода. По выходе из цикла мы восстанавливаем параметры терминала и завершаем программу.

Необходимо подчеркнуть разницу между блокированием сигналов в программе psswdmode и сбросом флага ISIG в программе pressq. В первом случае обработка сигналов откладывается. Если во время ввода пароля пользователь нажмет Ctrl-C, программа получит соответствующий сигнал после того, как пользователь нажмет ввод (и после того, как программа вернет терминал в нормальный режим). Если же вы сбрасываете флаг ISIG,

специальные сочетания клавиш не будут инициировать сигналы и программа вообще их не получит.