14 Расширенные операции ввода/вывода

14.1. Введение

В этой главе мы обсудим большое количество тем и функций, которые объединяются под общим термином *расширенные операции ввода/вывода*: неблокирующий ввод/вывод, блокировка записей, мультиплексирование операций ввода/вывода (функции `select` и `poll`), асинхронный ввод/вывод, функции `readv` и `writev` и ввод/вывод для файлов, отображаемых в память (`mmap`). Нам необходимо рассмотреть эти темы, прежде чем мы перейдем к обсуждению межпроцессных взаимодействий в главах 15 и 17 и в многочисленных примерах в последующих главах.

14.2. Неблокирующий ввод/вывод

В разделе 10.5 мы говорили, что системные вызовы подразделяются на две категории: «медленные» и все остальные. Медленными называют такие системные вызовы, которые могут заблокировать процесс «навечно». В эту категорию входят:

- Операция чтения, которая может «навечно» заблокировать вызывающий процесс, если в файлах определенных типов (каналы, терминальные устройства, сетевые устройства) отсутствуют данные, доступные для чтения.
- Операция записи также может «навечно» заблокировать вызывающий процесс, если данные не могут быть немедленно записаны в файлы тех же типов (отсутствует место в канале, переполнено сетевое соединение и т. п.).
- Операция открытия может заблокировать вызывающий процесс, пока не будут соблюдены некоторые условия для файлов определенных типов (например, открытие терминального устройства не может быть произведено, пока не будет установлено соединение между модемами, или открытие именованного канала FIFO только на запись будет заблокировано, пока не появится другой процесс, который откроет этот канал на чтение).
- Операции чтения и записи над файлами, для которых установлена принудительная блокировка записей.
- Некоторые операции `ioctl`.
- Некоторые функции, относящиеся к механизму межпроцессных взаимодействий (глава 15).

Мы также говорили, что системные вызовы, связанные с дисковыми операциями ввода/вывода, не относятся к категории медленных, хотя операция чтения с диска или записи на диск может на какое-то время заблокировать вызывающий процесс.

Неблокирующий режим ввода/вывода позволяет запускать такие операции, как `open`, `read` или `write`, не опасаясь, что они заблокируют процесс. Если запрошенная операция не может быть выполнена немедленно, системный вызов тут же вернет управление вызывающему процессу с признаком ошибки, сообщающим, что операция может быть заблокирована.

Существует два способа указать, что для заданного дескриптора файла должны использоваться неблокирующие операции ввода/вывода.

1. Если для получения дескриптора вызывается функция `open`, можно указать флаг `O_NONBLOCK` (раздел 3.3).
2. Чтобы включить флаг `O_NONBLOCK` для уже открытого дескриптора, следует воспользоваться функцией `fcntl` (раздел 3.14). В листинге 3.5 приводится функция, с помощью которой можно установить любой флаг дескриптора файла.

В ранних версиях System V для выбора неблокирующего режима операций ввода/вывода использовался флаг `O_NDELAY`. В этих версиях при отсутствии доступных для чтения данных функция `read` возвращала значение 0. Поскольку это противоречит принятому в UNIX соглашению, в соответствии с которым функция `read` возвращает 0 по достижении конца файла, стандарт POSIX.1 определил флаг неблокирующего режима с другим именем и с другой семантикой. В старых версиях System V, когда функция `read` возвращала значение 0, нельзя было определить, то ли это системный вызов вернул управление, потому что мог быть заблокирован, то ли действительно был достигнут конец файла. Стандарт POSIX.1 требует, чтобы в неблокирующем режиме при отсутствии доступных для чтения данных функция `read` возвращала признак ошибки −1 и код ошибки `EAGAIN` в переменной `errno`. Некоторые версии UNIX, происходящие от System V, поддерживают оба флага — и устаревший `O_NDELAY`, и определяемый стандартом POSIX.1 `O_NONBLOCK`, но в данной книге мы будем использовать только ту функциональность, которая определяется стандартом POSIX.1. Флаг `O_NDELAY` поддерживается лишь для сохранения обратной совместимости и не должен использоваться в новых приложениях.

В 4.3BSD появился флаг `FNDELAY` функции `fcntl` с несколько иной семантикой. Он воздействовал не только на флаги состояния файла в его дескрипторе — изменялись также флаги терминального устройства или сокета, что оказывало влияние на всех пользователей терминала или сокета, а не только на пользователей, совместно использующих одну и ту же запись в таблице файлов (в 4.3BSD неблокирующий режим ввода/вывода мог назначаться только терминальным устройствам или сокетам). Кроме того, в 4.3BSD в вызывающую программу возвращалось значение `EWOULDBLOCK`, если операция с дескриптором в неблокирующем режиме не могла быть завершена. Современные BSD-системы поддерживают флаг `O_NONBLOCK` и определяют константу `EWOULDBLOCK` с тем же значением, что и `EAGAIN`. Эти системы предоставляют семантику неблокирующего режима, совместимую со стандартом POSIX.1: изменения флагов состояния файла оказывают влияние на всех пользователей одной и той же записи в таблице файлов, но не затрагивают режимы работы с одним и тем же устройством, если для доступа к нему используются различные записи в таблице файлов (рис. 3.1 и 3.3).

Пример

Рассмотрим пример ввода/вывода в неблокирующем режиме. Программа в листинге 14.1 читает 500 000 байт со стандартного ввода и пытается вывести их в стандартный вывод. Стандартное устройство вывода предварительно переводится в неблокирующий режим. Вывод результатов каждой операции записи производится в стандартное устройство вывода сообщений об ошибках. Функция `clr_fl` очень похожа на функцию `set_fl` из листинга 3.5. Она просто сбрасывает один или более флагов.

Листинг 14.1. Вывод большого количества данных в неблокирующем режиме

```
#include "apue.h"
#include <errno.h>
#include <fcntl.h>

char     buf[500000];

int
main(void)
{
    int     ntowrite, nwrite;
    char    *ptr;

    ntowrite = read(STDIN_FILENO, buf, sizeof(buf));
    fprintf(stderr, "прочитано %d байт\n", ntowrite);

    set_fl(STDOUT_FILENO, O_NONBLOCK); /* установить неблокирующий режим */

    ptr = buf;
    while (ntowrite > 0) {
        errno = 0;
        nwrite = write(STDOUT_FILENO, ptr, ntowrite);
        fprintf(stderr, "nwrite = %d, errno = %d\n", nwrite, errno);

        if (nwrite > 0) {
            ptr += nwrite;
            ntowrite -= nwrite;
        }
    }

    clr_fl(STDOUT_FILENO, O_NONBLOCK); /* выход из неблокирующего режима */

    exit(0);
}
```

Мы предполагаем, что функция `write` отработает всего один раз, если стандартный вывод перенаправить в обычный файл:

```
$ ls -l /etc/services                  проверим размер файла
-rw-r--r--  1 root    677959 Jun 23  2009  /etc/services
$ ./a.out < /etc/services > temp.file  для начала попробуем с обычным файлом
прочитано 500000 байт
nwrite = 500000, errno = 0             единственный вызов write
$ ls -l temp.file                      проверим размер получившегося файла
-rw-rw-r--  1 sar     500000 Apr 1  13:03 temp.file
```

Но если в качестве устройства вывода будет использоваться терминал, мы предполагаем, что функция `write` будет иногда возвращать меньшее значение счетчика, а иногда — признак ошибки. Вот что мы получили в этом случае:

```
$ ./a.out < /etc/services 2>stderr.out вывод в терминал

$ cat stderr.out
прочитано 500000 байт
nwrite = 999, errno = 0
nwrite = -1, errno = 35
nwrite = -1, errno = 35
nwrite = -1, errno = 35
nwrite = -1, errno = 35
nwrite = 1001, errno = 0
nwrite = -1, errno = 35
nwrite = 1002, errno = 0
nwrite = 1004, errno = 0
nwrite = 1003, errno = 0
nwrite = 1003, errno = 0
nwrite = 1005, errno = 0
nwrite = -1, errno = 35                61 такая ошибка
. . .
nwrite = 1006, errno = 0
nwrite = 1004, errno = 0
nwrite = 1005, errno = 0
nwrite = 1006, errno = 0
nwrite = -1, errno = 35                108 таких ошибок
. . .
nwrite = 1006, errno = 0
nwrite = 1005, errno = 0
nwrite = 1005, errno = 0
nwrite = -1, errno = 35                681 такая ошибка
. . .
                                       и так далее...
nwrite = 347, errno = 0
```

В данной системе число 35 соответствует коду ошибки `EAGAIN`. Объем данных, принимаемых терминалом за одно обращение, варьируется от системы к системе. Результаты также зависят от того, как был произведен вход в систему — с консоли, с удаленного терминала или через сетевое соединение, которое использует псевдотерминал. Если на вашем терминале работает многооконная система, значит, вы также работаете через устройство псевдотерминала.

В этом примере программа произвела более 9000 вызовов функции `write`, хотя фактически вся работа была выполнена 500 вызовами. Остальные только возвращали признак ошибки. Такой тип цикла называется *опросом* (polling), и в многопользовательских системах он понапрасну расходует процессорное время. В разделе 14.4 мы рассмотрим более эффективный подход к работе с дескрипторами в неблокирующем режиме.

Иногда удается избежать применения неблокирующих операций ввода/вывода за счет использования потоков (глава 11). В этом случае можно позволить заблокировать один поток, если другие потоки смогут продолжать работу. Иногда такой подход упрощает архитектуру приложения, как будет показано в главе 21; однако в некоторых случаях проблемы, связанные с необходимостью синхронизации потоков, могут свести на нет все преимущества многопоточной модели.