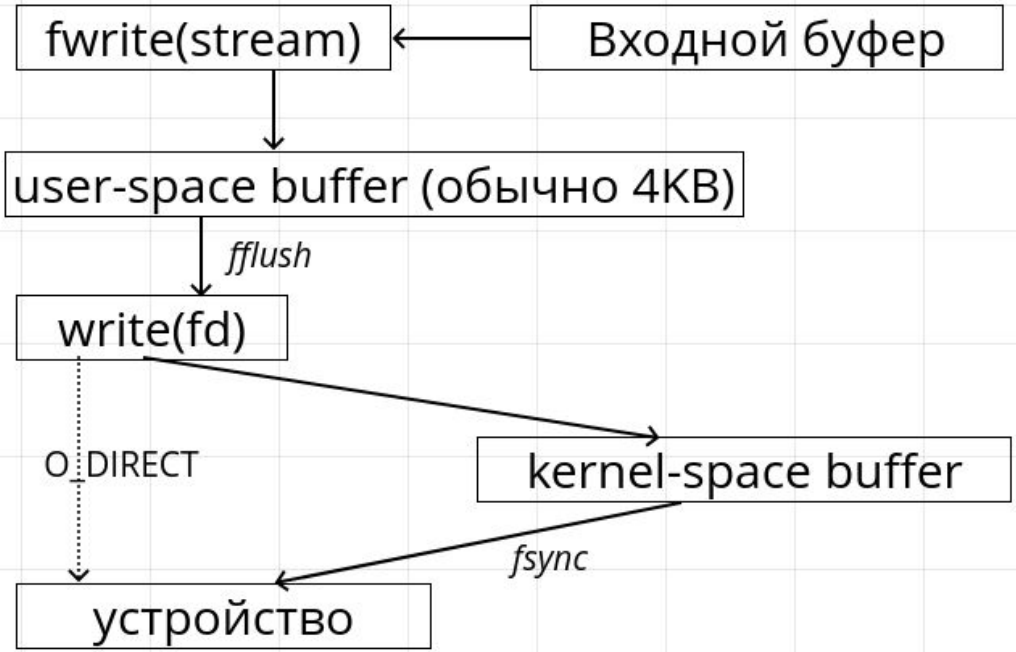
Программирование в Linux

Файлы. IO.

```c
#include <unistd.h>

ssize_t read(int fd, void* buf, size_t count);
ssize_t write(int fd, const void* buf, size_t count);
```



Конкурентный доступ к файлу

- Системные read/write атомарны
- Порядок определяет планировщик
- В каждом открытом файле есть мьютекс (см. flock)

Блокирующий ввод/вывод

Как прочитать N байт?

read(fd, buf, N)?

```
1  ssize_t read_all(int fd, char* buff, size_t len) {
2    ssize_t actual_read = 0;
3    while (len != 0) {
4      auto read_cnt = read(fd, buff, len);
5      if (read_cnt == 0) {
6        // файл кончился
7        break;
8      }
9      if (read_cnt < 0) {
10       if (errno == EINTR) {
11         // нас прервали, попробуй еще раз
12         continue;
13       }
14       // серьезная ошибка
15       return -1;
16     }
17     actual_read += read_cnt;
18     buff += read_cnt;
19     len -= read_cnt;
20   }
21   return actual_read;
22 }
```

Векторные чтение/запись

```c
ssize_t readv(int fd, const struct iovec *iov, int iovcnt);

ssize_t writev(int fd, const struct iovec *iov, int iovcnt);

struct iovec {
    void  *iov_base;    /* Starting address */
    size_t iov_len;     /* Number of bytes to transfer */
};
```

```c
char *str0 = "hello ";
char *str1 = "world\n";
struct iovec iov[2];
ssize_t nwritten;

iov[0].iov_base = str0;
iov[0].iov_len = strlen(str0);
iov[1].iov_base = str1;
iov[1].iov_len = strlen(str1);

nwritten = writev(STDOUT_FILENO, iov, 2);
```

Позиционирование

- Внутренний указатель связан с fd
- read/write двигают указатель
- pread/pwrite — не двигают

```
1  ssize_t pread(int fd, void *buf, size_t count, off_t offset);
2
3  ssize_t pwrite(int fd, const void *buf, size_t count, off_t offset);
```

- lseek — подвигать вручную
- lseek за конец + write = дыры в файле

Неблокирующий ввод/вывод

open с флагом O_NONBLOCK

```
start:
read_cnt = read(fd, buff, len);
if (read_cnt < 0) {
    if (errno == EINTR) {
        goto start;
    }
    if (errno == EAGAIN) {
        // читать рано -- надо ждать,
        // заимитесь чем-то другим
    } else {
       // что-то более серьезное
       return -1;
    }
}
```

Ожидание событий IO

- while (!ready) { ready = try_read(...); } // активное ожидание ест CPU
- while (!ready) { sleep(1ms); ready = try_read(...); } // частота опроса может быть недостаточной
- лучше попросить ОС об оповещении

```
 1 #include <sys/select.h>
 2
 3 int select(int nfds, fd_set *readfds, fd_set *writefds,
 4            fd_set *exceptfds, struct timeval *timeout);
 5
 6 void FD_CLR(int fd, fd_set *set);
 7 int  FD_ISSET(int fd, fd_set *set);
 8 void FD_SET(int fd, fd_set *set);
 9 void FD_ZERO(fd_set *set);
10
11
12 int pselect(int nfds, fd_set *readfds, fd_set *writefds,
13             fd_set *exceptfds, const struct timespec *timeout,
14             const sigset_t *sigmask);
```

```cpp
#include <sys/select.h>
#include <sys/unistd.h>
#include <sys/types.h>

#include <cstdlib>
#include <cstdio>

int main() {
    fd_set readfds; FD_ZERO(&readfds);
    FD_SET(STDIN_FILENO, &readfds);
    timeval timeout; timeout.tv_sec = 5;
                     timeout.tv_usec = 0;

    int ready_fd_cnt = select(STDIN_FILENO + 1, &readfds, nullptr, nullptr,
                              &timeout);
    if (ready_fd_cnt < 0) {
        perror("select");
        return EXIT_FAILURE;
    }
    if (ready_fd_cnt == 0) {
        printf("Timeout");
        return EXIT_SUCCESS;
    }
    if (FD_ISSET(STDIN_FILENO, &readfds)) {
        const int kBufSize = 1024;
        char buf[kBufSize + 1] = {0};
        auto read_cnt = read(STDIN_FILENO, buf, kBufSize);
        if (read_cnt < 0) {
            perror("read");
            return EXIT_FAILURE;
        }
        if (read_cnt) {
            printf("read: %s\n", buf);
        }
    }
}
```

Альтернатива select — poll

```c
#include <poll.h>
#include <signal.h>


int poll(struct pollfd *fds, nfds_t nfds, int timeout_ms);

int ppoll(struct pollfd *fds, nfds_t nfds,
          const struct timespec *tmo_p, const sigset_t *sigmask);


struct pollfd {
   int   fd;         /* file descriptor */
   short events;     /* requested events */
   short revents;    /* returned events */
};
```

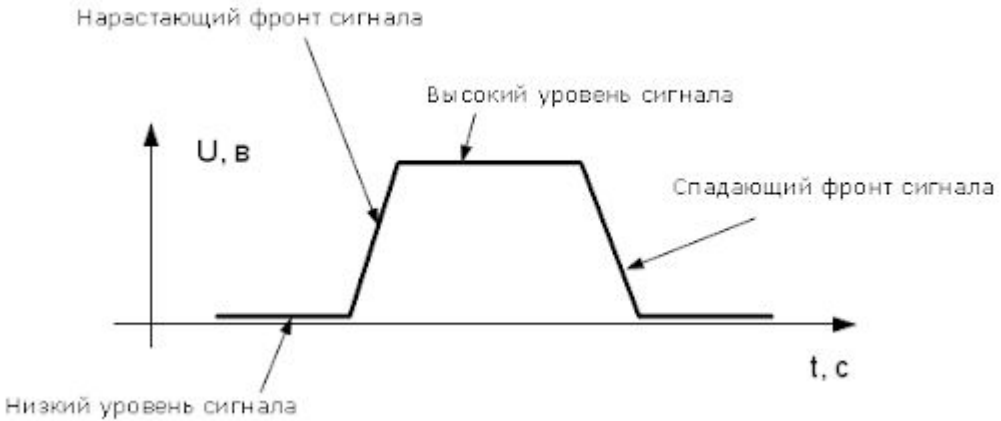
```c
#include <poll.h>
#include <unistd.h>

#include <cstdlib>
#include <cstdio>

int main() {

    pollfd request;
    request.fd = STDIN_FILENO;
    request.events = POLLIN;

    int ready_cnt = poll(&request, 1, 5000);

    if (ready_cnt < 0) {
        perror("poll");
        return EXIT_FAILURE;
    }
    if (ready_cnt == 0) {
        printf("Timeout");
        return EXIT_SUCCESS;
    }
    if (request.revents & POLLIN) {
        const int kBufSize = 1024;
        char buf[kBufSize + 1] = {0};
        auto read_cnt = read(STDIN_FILENO, buf, kBufSize);
        if (read_cnt < 0) {
           perror("read");
           return EXIT_FAILURE;
        }
        if (read_cnt) {
            printf("read: %s\n", buf);
        }
    }
}
```

epoll — linux specific

```cpp
int listen_sock; // каким-то образом полученный
                 // дескриптор серверного сокета,
                 // для которого ждем событий

// создаем ожидалку
const int kEpollFlags = 0;
int epollfd = epoll_create1(kEpollFlags);
if (epollfd == -1) {
    perror("epoll_create1");
    std::exit(EXIT_FAILURE);
}

{
    epoll_event ev;
    ev.events = EPOLLIN; // будем входящих данных
    ev.data.fd = listen_sock;
    // регистрируем событие для ожмдания
    if (epoll_ctl(epollfd, EPOLL_CTL_ADD, listen_sock, &ev) == -1) {
        perror("epoll_ctl: listen_sock");
        exit(EXIT_FAILURE);
    }
}
```

```cpp
#define MAX_EVENTS 10
epoll_event events_buffer[MAX_EVENTS];
for (;;) {
    int events_cnt = epoll_wait(epollfd,
                                events_buffer, MAX_EVENTS,
                                -1); // Timeout
    if (events_cnt == -1) {
        perror("epoll_wait");
        exit(EXIT_FAILURE);
    }
    // дождались событий
    for (int idx = 0; idx < events_cnt; ++idx) {
        if (events_buffer[idx].data.fd == listen_sock) {
            // для примера сервер подключает новых клиентов ->
            // новые файловые дескрипторы
            int conn_sock = accept(listen_sock, &addr, &addrlen);
            if (conn_sock == -1) {
                perror("accept");
                exit(EXIT_FAILURE);
            }
            // включаем неблокирующий режим
            fcntl(conn_sock,
                  F_SETFL,
                  fcntl(conn_sock, F_GETFL, 0) | O_NONBLOCK);
            // и будем ждать на них событий — добавляем к ожидалке
            epoll_event ev;
            ev.events = EPOLLIN | EPOLLET;
            ev.data.fd = conn_sock;
            if (epoll_ctl(epollfd,
                          EPOLL_CTL_ADD,
                          conn_sock, &ev) == -1) {
                perror("epoll_ctl: conn_sock");
                exit(EXIT_FAILURE);
            }
        } else {
            // дескриптор клиента готов к неблокирующим операцияи
            // events_buffer[idx].data.fd;
        }
    }
}

close(epollfd);
```

флаг EPOLLET — детект события по фронту сигнала, а не по уровню



Posix aio

```c
#include <aiocb.h>
#include <aio.h>

struct aiocb {
   /* The order of these fields is implementation-dependent */

   int             aio_fildes;     /* File descriptor */
   off_t           aio_offset;     /* File offset */
   volatile void  *aio_buf;        /* Location of buffer */
   size_t          aio_nbytes;     /* Length of transfer */
   int             aio_reqprio;    /* Request priority */
   struct sigevent aio_sigevent;   /* Notification method */
   int             aio_lio_opcode; /* Operation to be performed;
                                      lio_listio() only */

   /* Various implementation-internal fields not shown */
};

/* Operation codes for 'aio_lio_opcode': */

enum { LIO_READ, LIO_WRITE, LIO_NOP };

```

и пачка методов *aio_OPNAME: aio_read, aio_write, aio_cancel...*

Ссылки

1. Epoll https://habr.com/ru/post/416669/
2. https://idea.popcount.org/2017-01-06-select-is-fundamentally-broken/
3. https://habr.com/ru/company/badoo/blog/439972/