

Лекция 13

Взаимодействие процессов

Перенаправление ввода/вывода

- Для перенаправление ввода или вывода после `fork()` необходимо открыть нужный файл и потом скопировать его файловый дескриптор в стандартные файловые дескрипторы 0, 1 или 2.
- Копирование файлового дескриптора: `dup2`
 - `int dup2(int oldfd, int newfd);`
- Пример: `redir.c`

Соответствие с флагами open

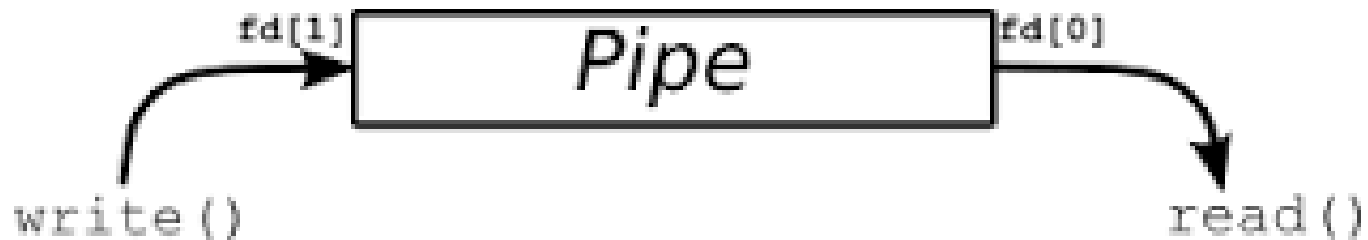
- [n]> FILE
 - fd = open(FILE, O_WRONLY | O_CREAT | O_TRUNC, 0666);
dup2(fd, n); close(fd);
- [n]>>FILE
 - fd = open(FILE, O_WRONLY | O_CREAT | O_APPEND, 0666);
dup2(fd, n); close(fd);
- [n]<FILE
 - fd = open(FILE, O_RDONLY, 0); dup2(fd, n); close(fd);
- [n]>&[m]
 - dup2(m, n);

Неименованные каналы

- Канал — механизм синхронной однонаправленной потоковой передачи данных между процессами
 - Синхронной — процесс-писатель и процесс-читатель могут синхронизировать работу по write/read
 - Потоковой — при передаче в канале не сохраняются границы записанных блоков, последовательность данных, записанных несколькими write, может быть прочитана за один read

Передача данных

- Посылка данных — `write`
- Прием данных — `read`
- `Read` и `write` могут быть не синхронизированы по времени — в ядре ОС находится буфер для временного хранения данных
- Буфер имеет **ограниченный** и **фиксированный** размер
- Например, Linux — 64 KiB. Пример: `pipesize.c`



Создание канала

```
int pipe(int fds[2]);
```

- Создаются два связанных файловых дескриптора, которые возвращаются в массиве `fds`
- `fds[0]` — файловый дескриптор для чтения из канала
- `fds[1]` — файловый дескриптор для записи в канал

Запись в канал: write

- Если все ф. д. чтения из канала закрыты, при попытке записи процессу посылается SIGPIPE и write возвращает код ошибки EPIPE
- Если размер данных $\leq \text{PIPE_BUF}$
 - Если в канале достаточно места, данные записываются в канал атомарно
 - Если в канале места недостаточно, записывающий процесс блокируется либо до освобождения места, либо до закрытия всех ф. д. чтения
- Если размер данных $> \text{PIPE_BUF}$, атомарность записи не гарантируется

Чтение из канала: read

- Если в канале нет данных и все ф. д. записи в канал закрыты, read возвращает 0 — признак конца файла
- Если в канале есть данные, то read завершается немедленно и возвращает минимум из запрошенного и имеющегося в канале размера
- Если в канале нет данных, процесс блокируется до наступления одного из первых двух условий

Взаимная блокировка, тупик (deadlock)

- Проблема синхронизации процессов, когда ни один из взаимодействующих процессов не может продолжить выполнение, так как ожидает выполнения действия другим процессом

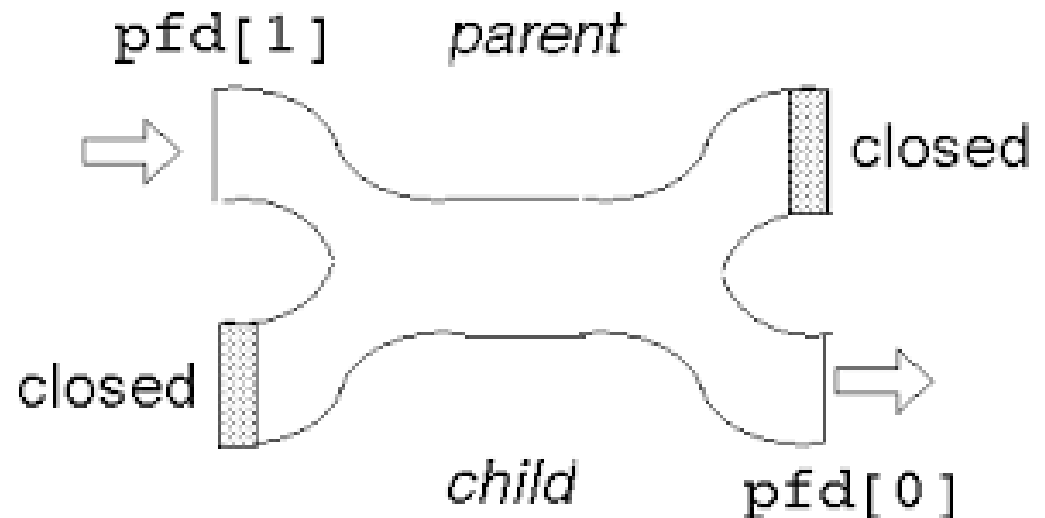
```
int fds1[2], fds2[2];  
pipe(fds1);  
pipe(fds2);
```

```
read(fds1[0], b, sizeof(b));  
write(fds2[1], b, sizeof(b));
```

```
read(fds2[0], b, sizeof(b));  
write(fds1[1], b, sizeof(b));
```

Использование каналов для конвейера (pipeline.c)

```
int main(void)
{
    pipe(pfd);
    if (!(pid1 = fork())) {
        dup2(pfd[1], 1); close(pfd[1]);
        execlp("/bin/ls", "/bin/ls", "-l", NULL);
    }
    if (!(pid1 = fork())) {
        dup2(pfd[0], 0);
        close(pfd[0]);
        execlp("/usr/bin/wc",
              "/usr/bin/wc", "-l", NU
    }
    wait(0); wait(0);
    return 0;
}
```



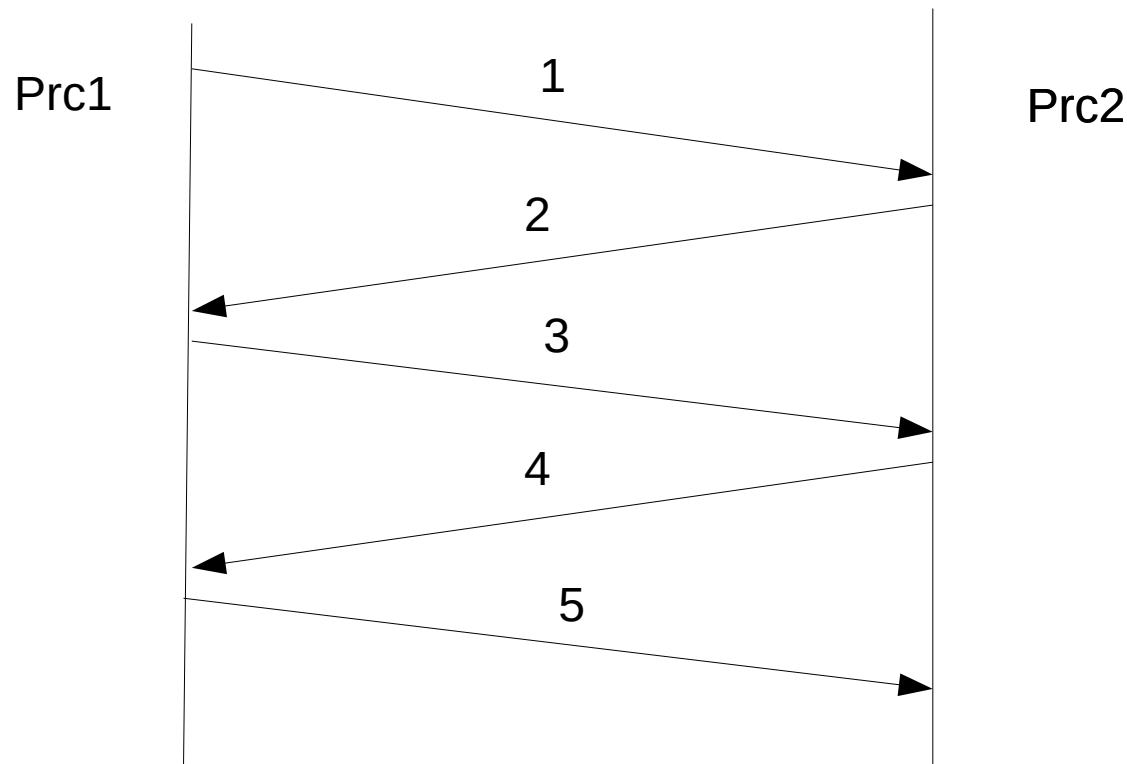
Использование каналов для конвейера

```
int main(void)
{
    pipe(pfd);
    if (!(pid1 = fork())) {
        dup2(pfd[1], 1); close(pfd[1]); close(pfd[0]);
        execlp("/bin/ls", "/bin/ls", "-l", NULL);
    }
    if (!(pid1 = fork())) {
        dup2(pfd[0], 0); close(pfd[0]); close(pfd[1]);
        execlp("/usr/bin/wc", "/usr/bin/wc", "-l", NULL);
    }
    close(pfd[0]); close(pfd[1]);
    wait(0); wait(0);
    return 0;
}
```

Все ненужные файловые дескрипторы должны быть закрыты!

Задача ping-pong

- Модельная задача взаимодействия двух процессов
- Поочередный обмен сообщениями между процессами



Ping-pong

- Чтобы данные процессов не перемешивались, два варианта
 - Использовать два pipe:
 - (1) от первого процесса ко второму,
 - (2) от второго процесса первому
 - Использовать один pipe, но арбитраживать доступ к pipe другими средствами
- Пример: pingpong.c

Низкоуровневый и высокоуровневый ввод-вывод (C)

- `fdopen` — создает структуру `FILE` по файловому дескриптору

`FILE *fdopen(int fd, const char *mode);`

- `fileno` — получить файловый дескриптор по структуре `FILE`

`int fileno(FILE *f);`

- Если из `pipe` создать `FILE *`, то он по умолчанию будет **полностью буферизован**
- Пример: `fdopen.c`

File descriptors & streams (C++)

- Расширение стандарта
`#include <ext/stdio_filebuf.h>`

`__gnu_cxx::stdio_filebuf<char>(int fd,
std::ios::mode);
std::ostream(filebuf *);`
- При работе с каналами поток будет полностью буферизован
- Пример: `fdstream.cpp`