Многопоточное программирование на C/C++

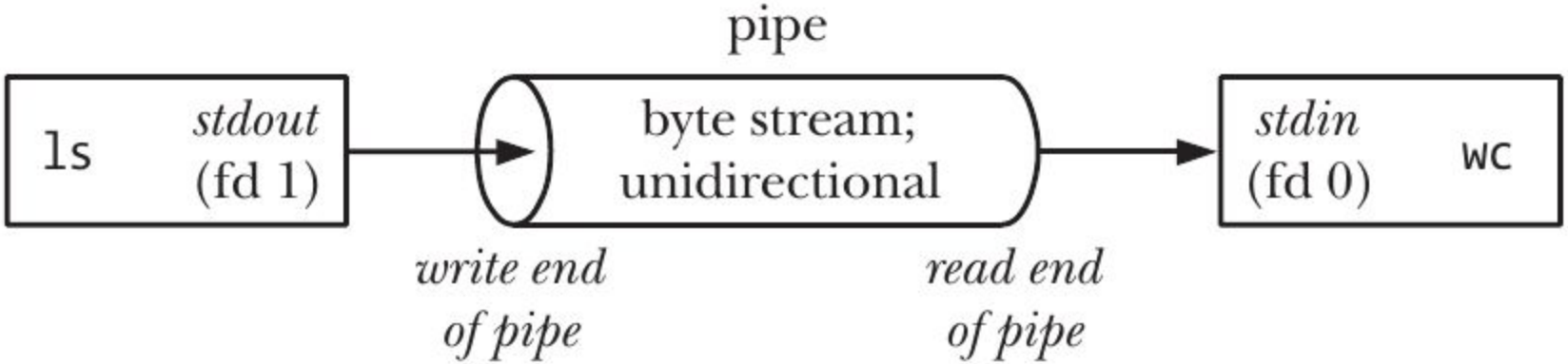
Pipes and FIFOs

Pipe

```
$ ls | wc -l
```

Шелл создает два процесса, запуская ls и wc (с помощью *fork()* и *exec()*)

Pipe


pipe
ls
stdout
(fd 1)
byte stream;
unidirectional
stdin
(fd 0)
WC
write end
of pipe
read end
of pipe

Pipe

Pipe - поток байт, процесс может читать из него столько байт, сколько хочет, независимо от того, какими кусками другой процесс пишет туда.

Невозможно свободно позиционироваться (использовать *lseek()*).

Pipe

Если запись прекращается, то читатель получит EOF (*read()* вернет 0), когда прочитает все данные.

Если несколько процессов пишут в один pipe, то гарантируется, что данные не будут смешаны, если процессы пишут не более чем PIPE_BUF байт за раз.

Создание и использование Pipe

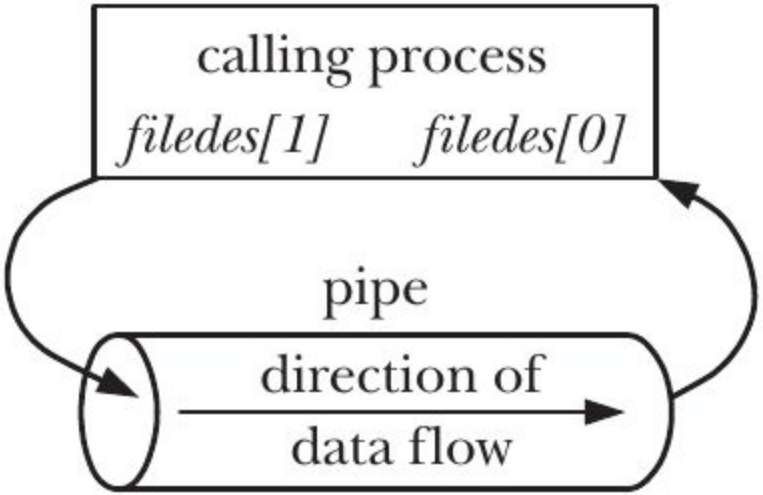
```
#include <unistd.h>

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

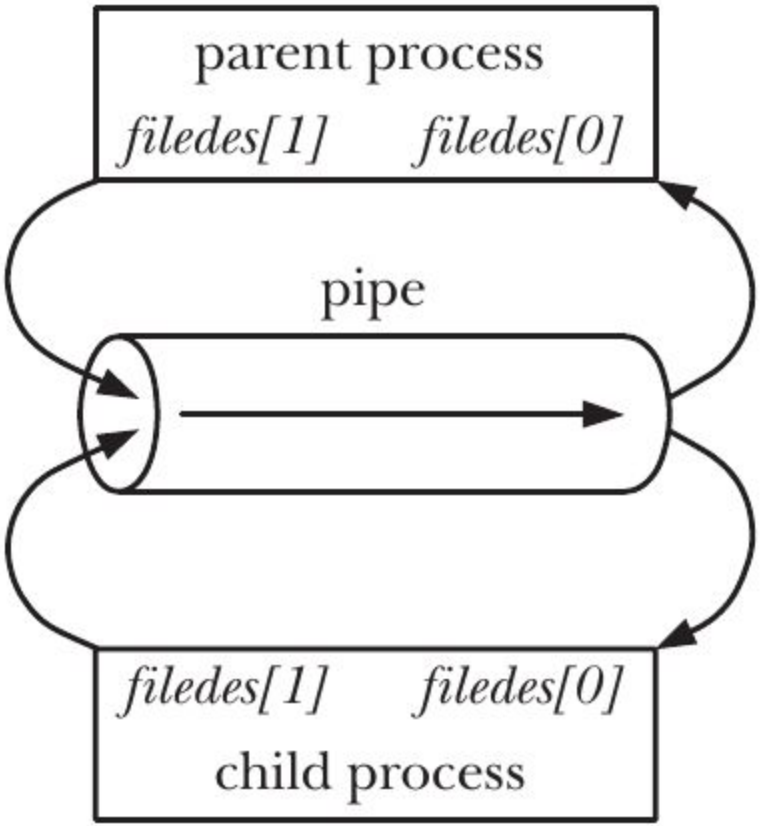
Возвращает 0 при успешном завершении, -1 при ошибке.

При успешном завершении в массиве filedes будут содержаться два открытых файловых дескриптора, один для чтения, другой для записи.

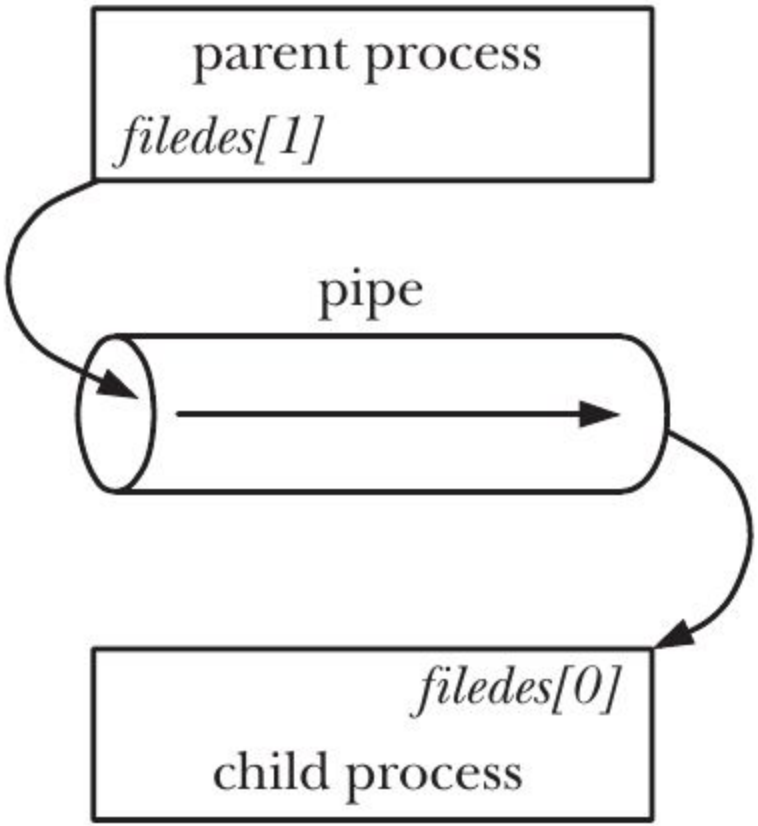
Pipe


calling process
filedes[1]
filedes[0]
pipe
direction of
data flow

Pipe


parent process
filedes[1]
filedes[0]
pipe
filedes[1]
filedes[0]
child process

Pipe



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

switch (fork()) {
case 0: /* Child */
     close(filedes[1])

     /* Child now reads from pipe */
     break;

default: /* Parent */
     close(filedes[0]);

     /* Parent now writes to pipe */
     break;
}
```

Зачем закрывать ненужные дескрипторы?

Если читающий процесс не закроет пишущий дескриптор - то после того как пишущий процесс прекратит запись, читающему не придет EOF.

Пишущий процесс при попытке записать в закрытый пайп получает SIGPIPE, либо же *wirte()* возвращает EPIPE.

Пример: simple_pipe.c

dup

```
int pfd[2];

/* Allocates (say) file descriptors 3 and 4 for pipe */
pipe(pfd);

/* Other steps here, e.g., fork() */

/* Free file descriptor 1 */
close(STDOUT_FILENO);
/* Duplication uses lowest free file
descriptor, i.e., fd 1 */
dup(pfd[1]);
```

dup2

```
/* Close descriptor 1, and reopen bound to
write end of pipe */

dup2(pfd[1], STDOUT_FILENO);
```

dup2

```
close(pfd[1]);
```

Пример: pipe_ls_wc.c

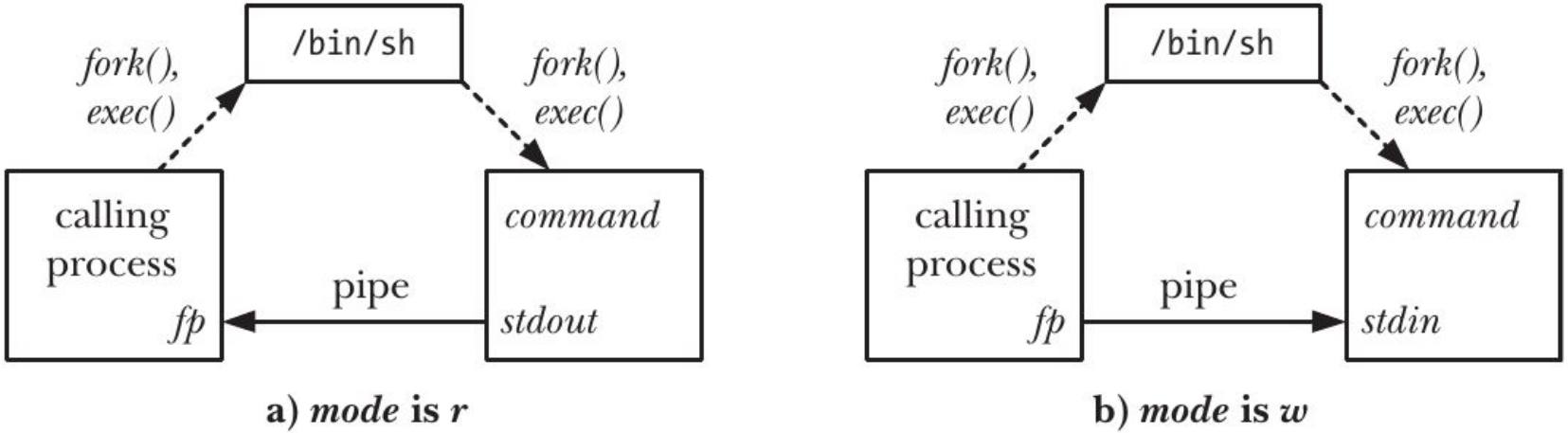
Взаимодействие с командой шелла

```
#include <stdio.h>

FILE *popen(const char *command, const char *mode);

int pclose(FILE *stream);
```

Popen


/bin/sh
fork(),
exec()
fork(),
exec()
calling
process
fp
pipe
command
stdout
a) mode is r
/bin/sh
fork(),
exec()
fork(),
exec()
calling
process
fp
pipe
command
stdin
b) mode is w

FIFO

Тоже самое, что и pipe, только дополнительно имеет имя в файловой системе.

$ mkfifo [-m mode] pathname

mkfifo

```
#include <sys/stat.h>

int mkfifo(const char *pathname, mode_t mode);
```