IPC-shell

Виды

IPC (взаимодействие между процессами):

- Сигналы (служебное взаимодействие)
- Файлы (read/write/flock/...)
- Именованные/неименованные каналы
- Семафоры/разделяемая память/очереди сообщений:
  - Sys V
  - POSIX
- Сетевое взаимодействие

Файловые дескрипторы

Получение: open(char *path, int flags, ...)

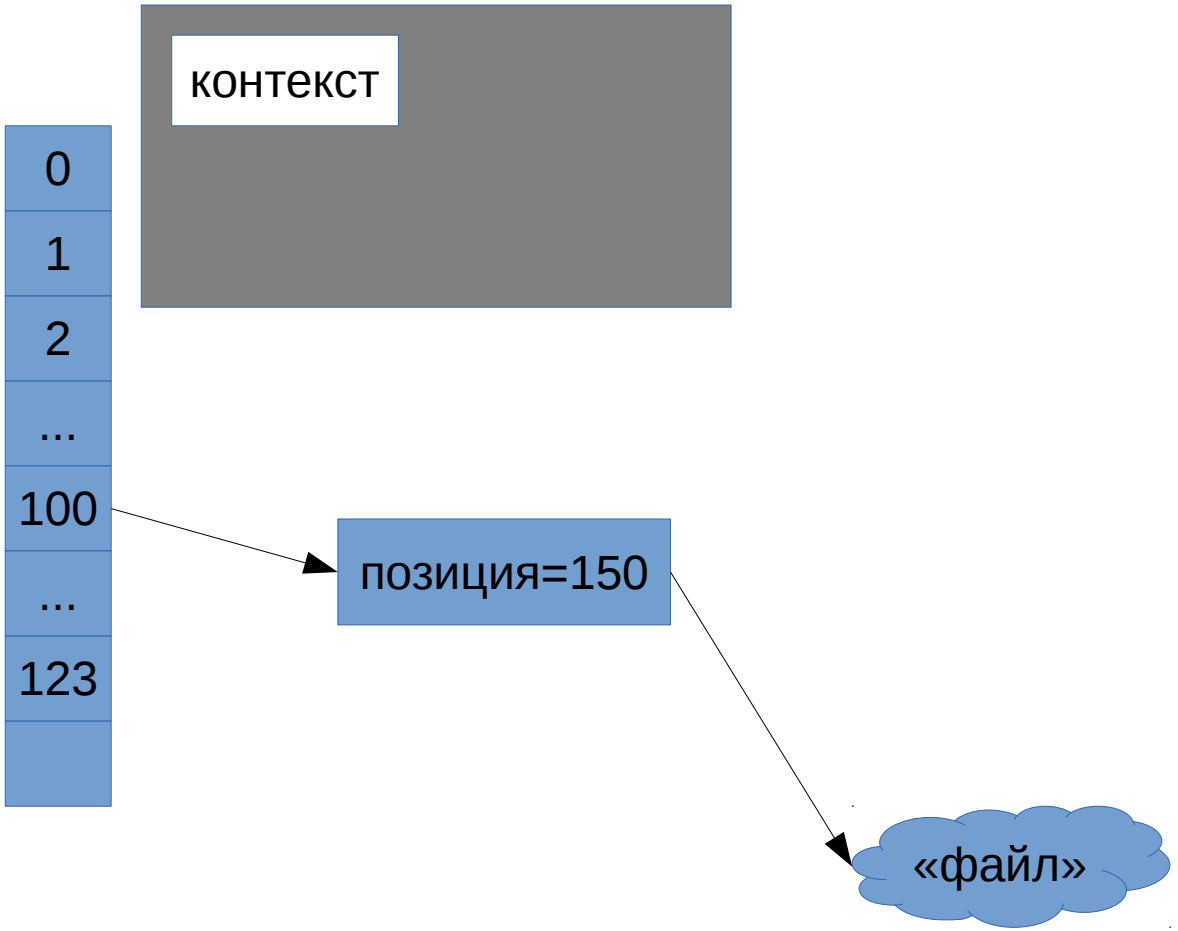
- Работа:
  - Чтение: read(int d, void *buf, size_t nbytes)
  - Запись: write(int d, void *buf, size_t nbytes)
  - Перемещение: lseek(int d, offset, int whence)
- Завершение работы: close(int d)

man 2 <имя>

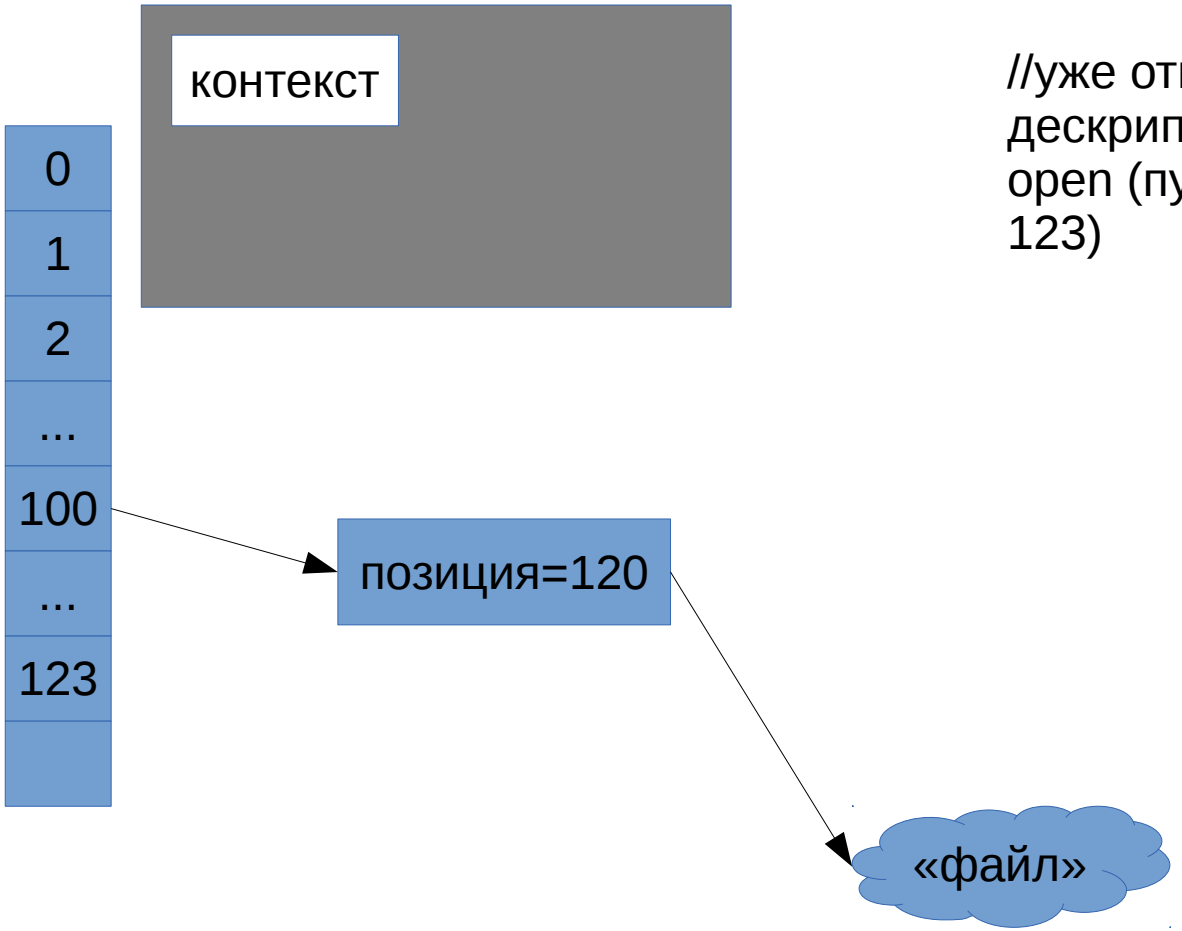
Дескрипторы

Таблица дескрипторов — это просто массив «указателей», от которого мы знаем только индексы

Процесс и дескрипторы

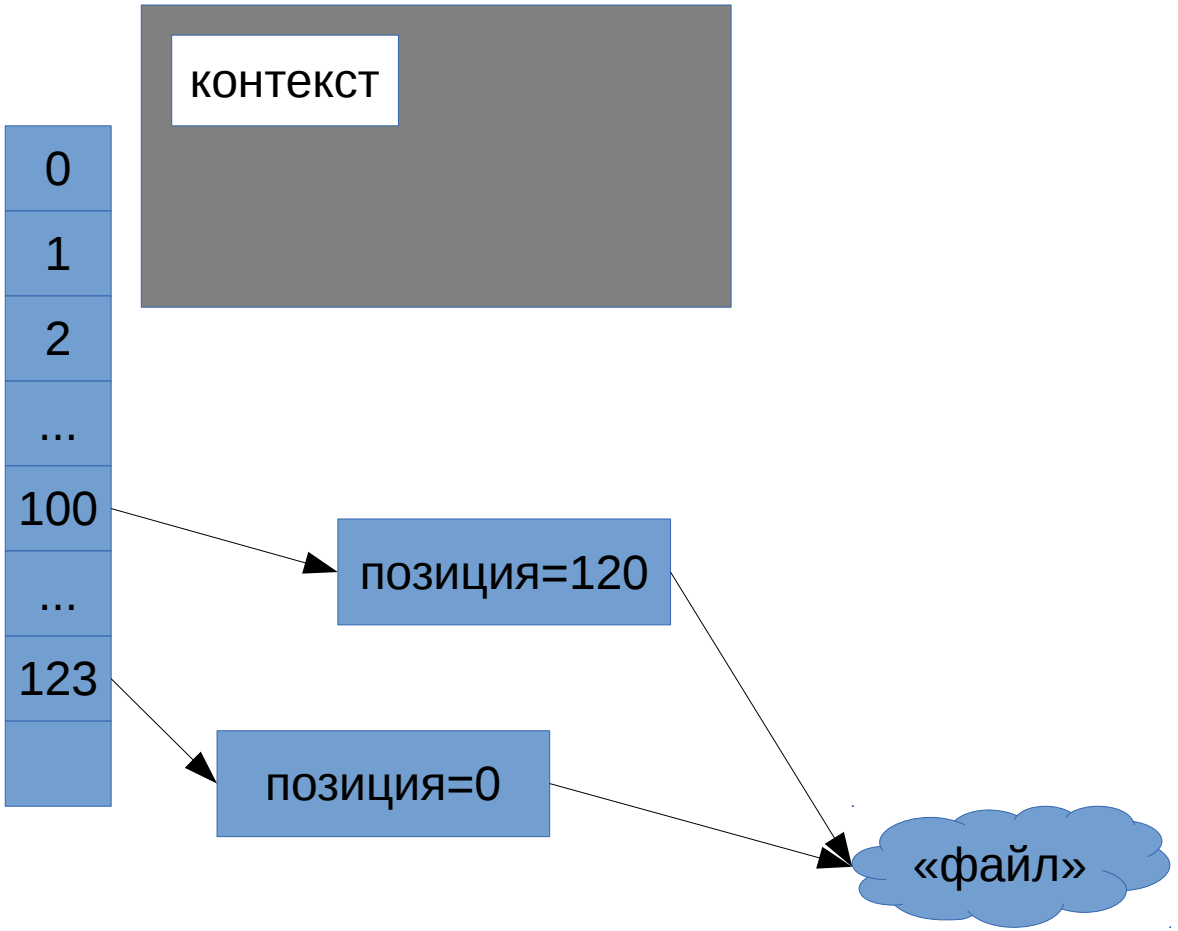


open + open



//уже открыт файл (пусть дескриптор 100), делаем новый open (пусть вернется дескриптор 123)

open + open



Дескрипторы и нити

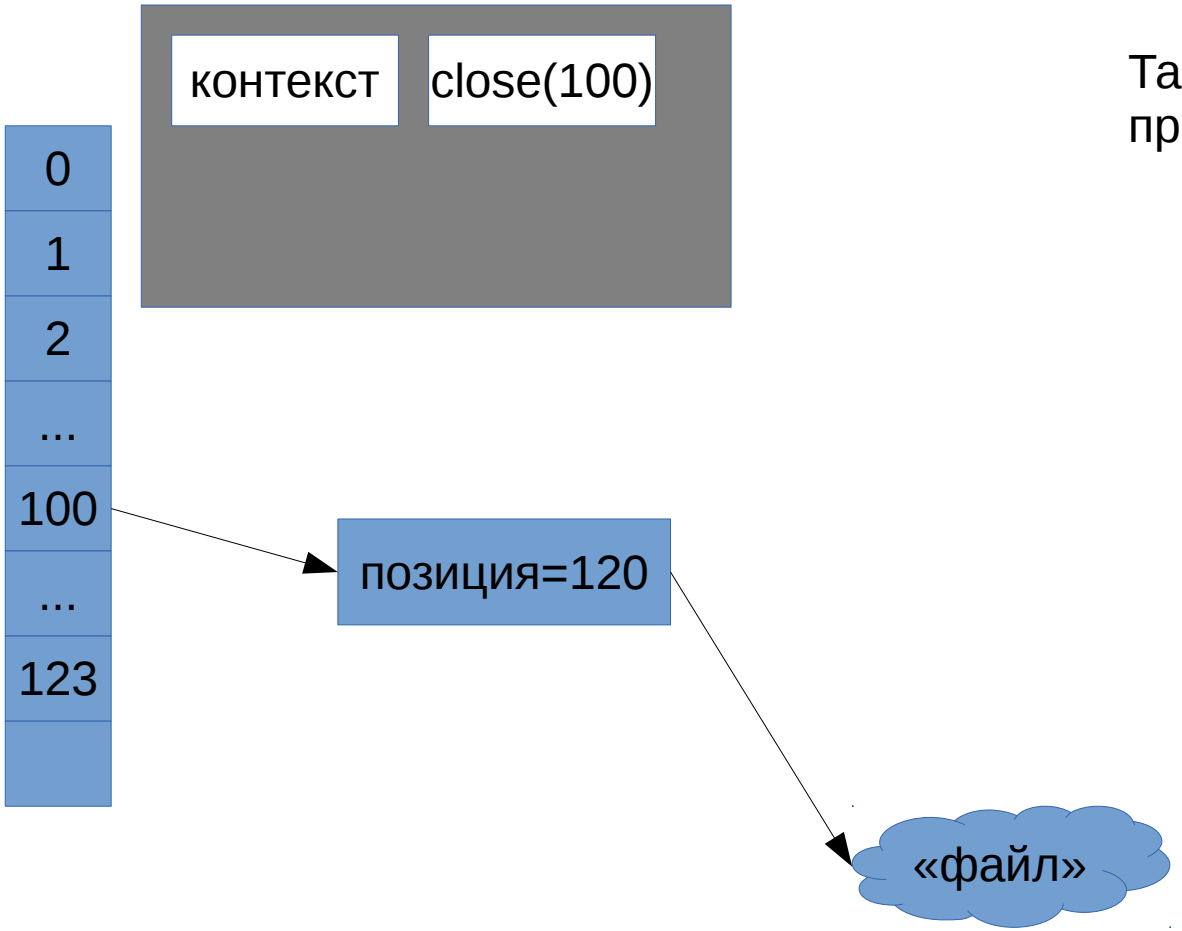


Таблица одна на весь процесс

Дескрипторы и нити

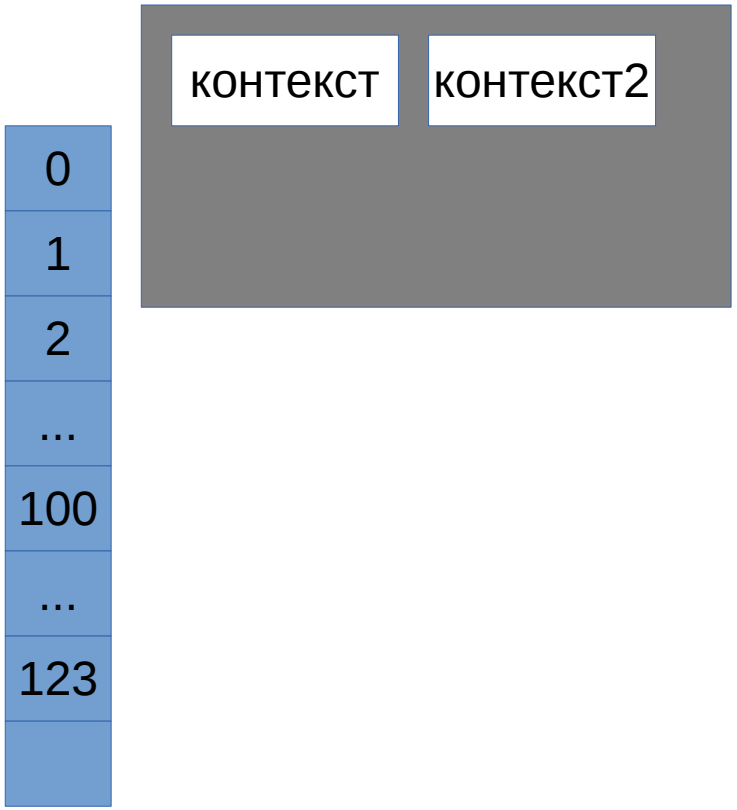
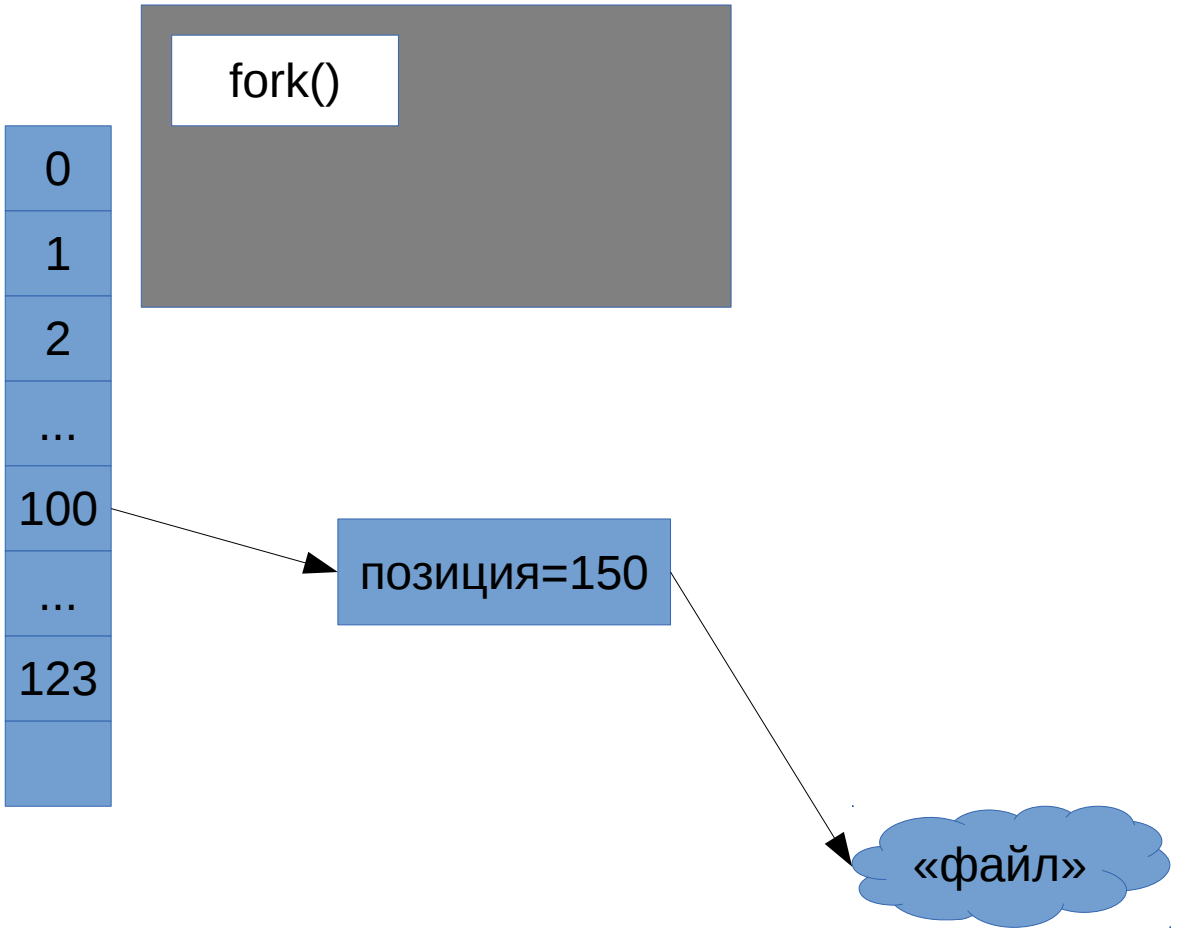
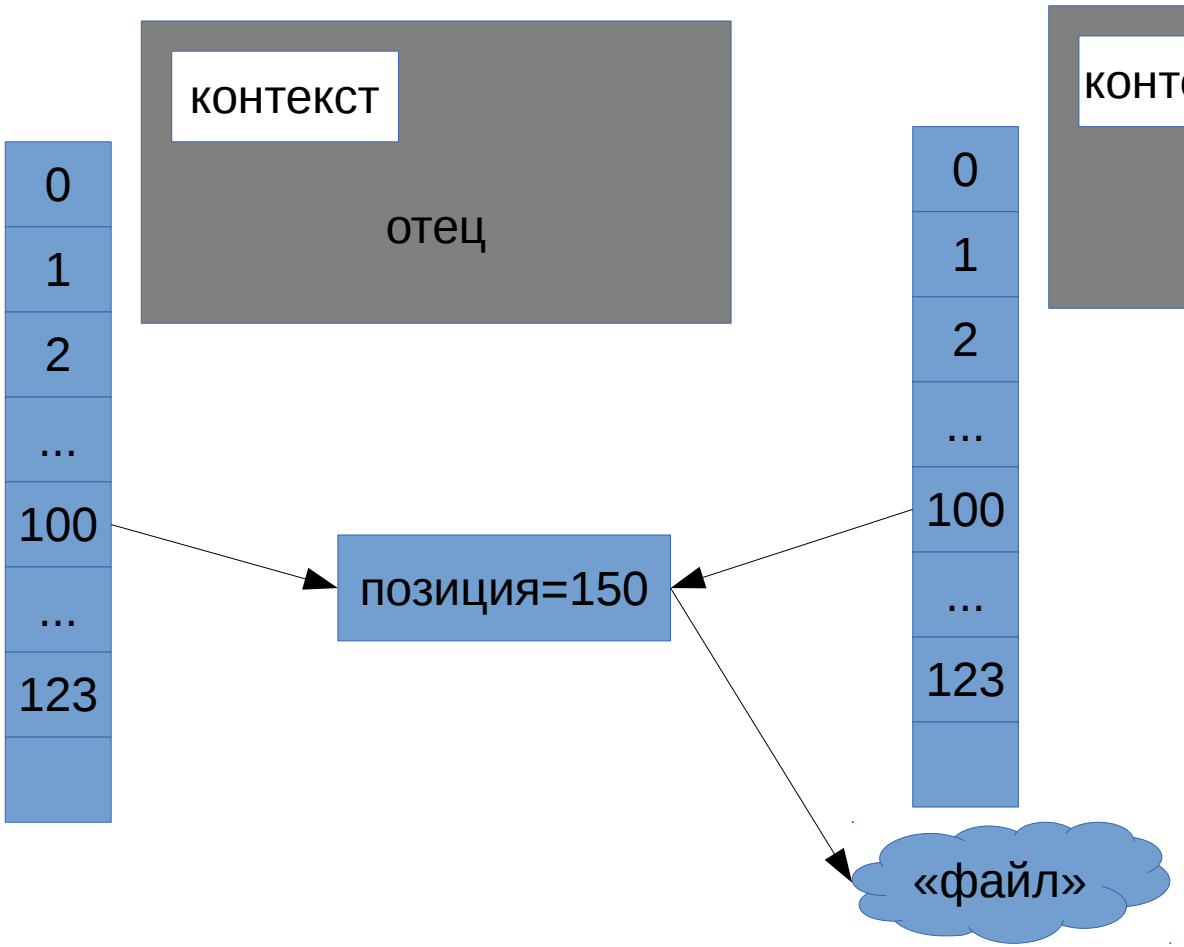


Таблица одна на весь процесс

Дескрипторы и fork

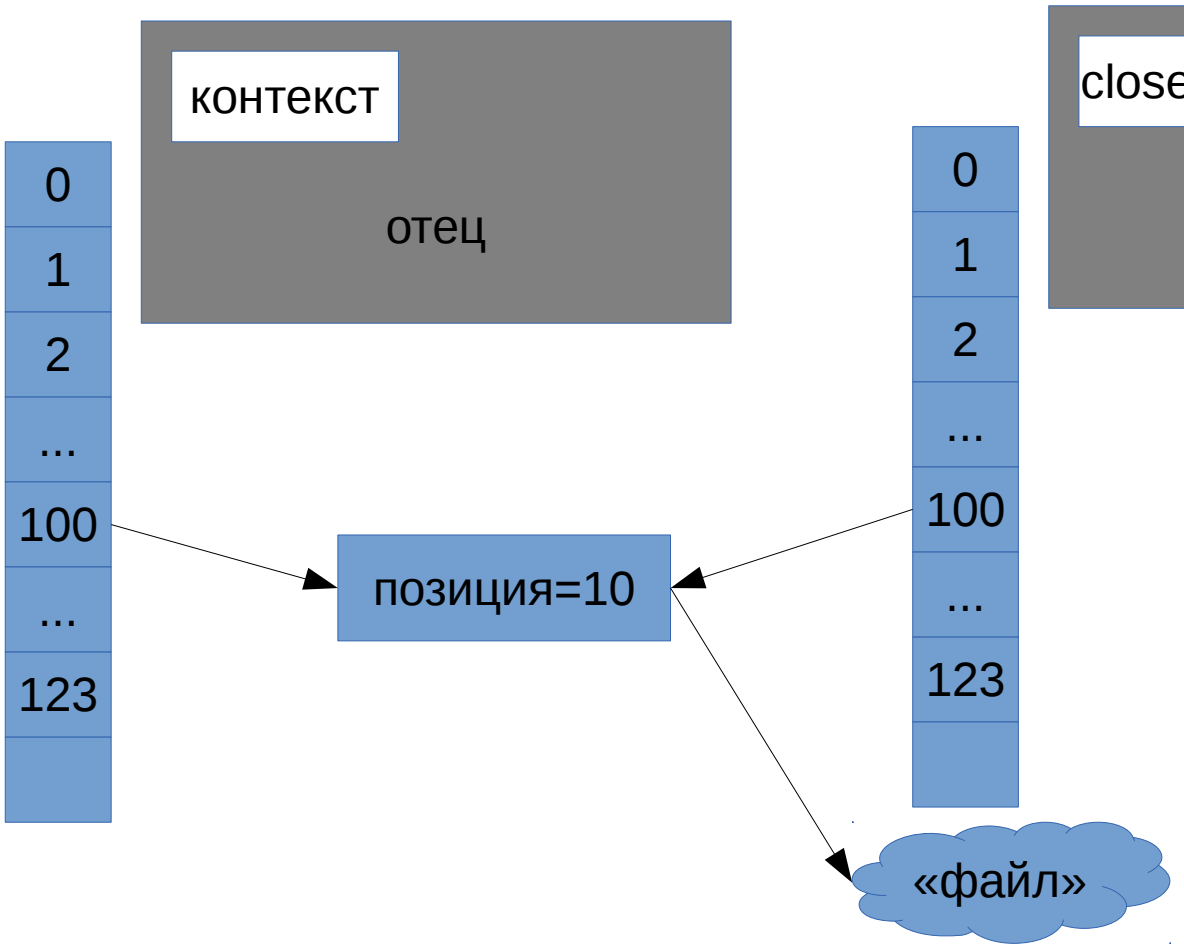


Тяжелые процессы и дескрипторы



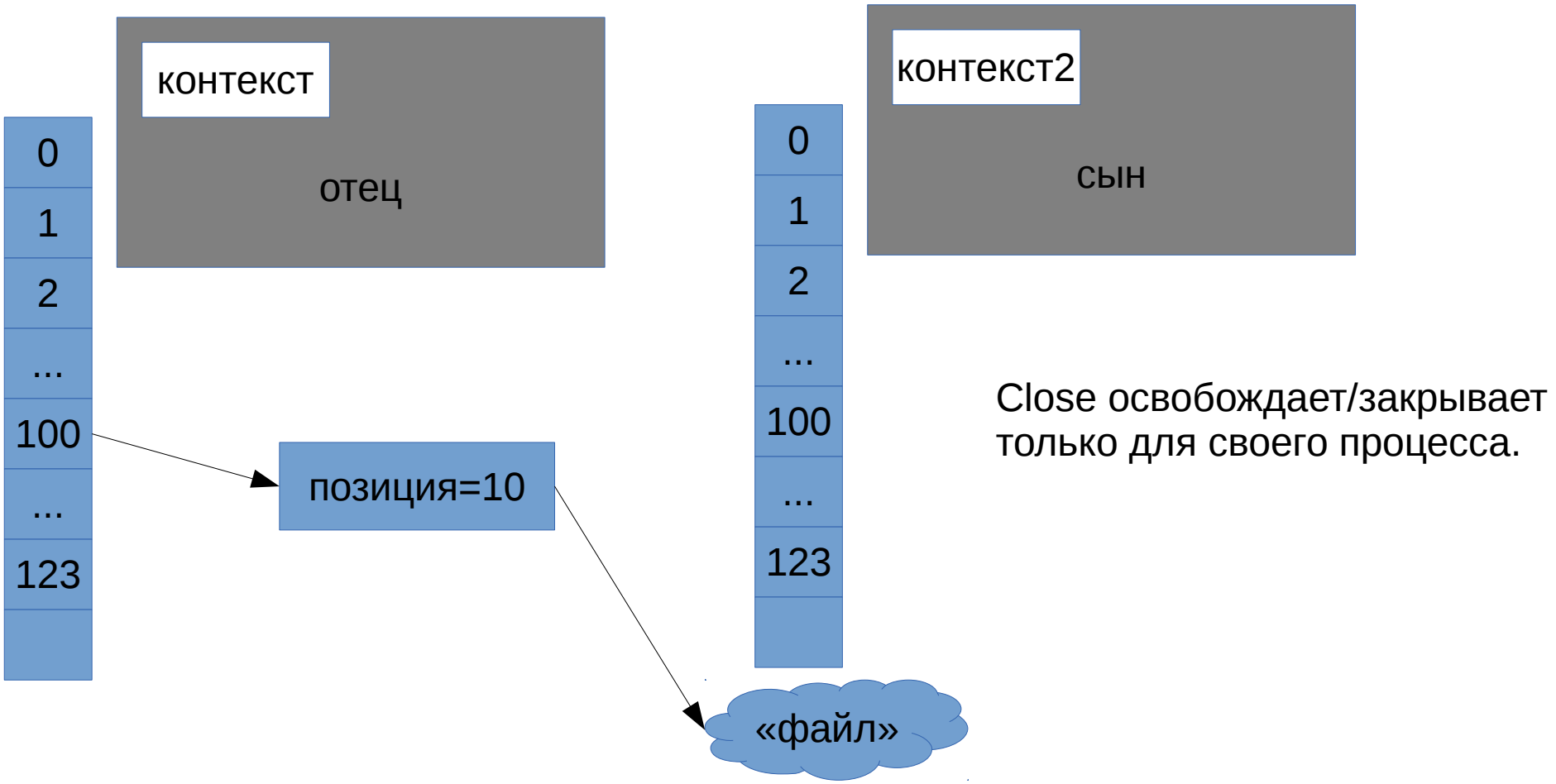
Изменение (lseek/read/write)
в одном процессе приведет
к изменению позиции и в другом

Тяжелые процессы и дескрипторы



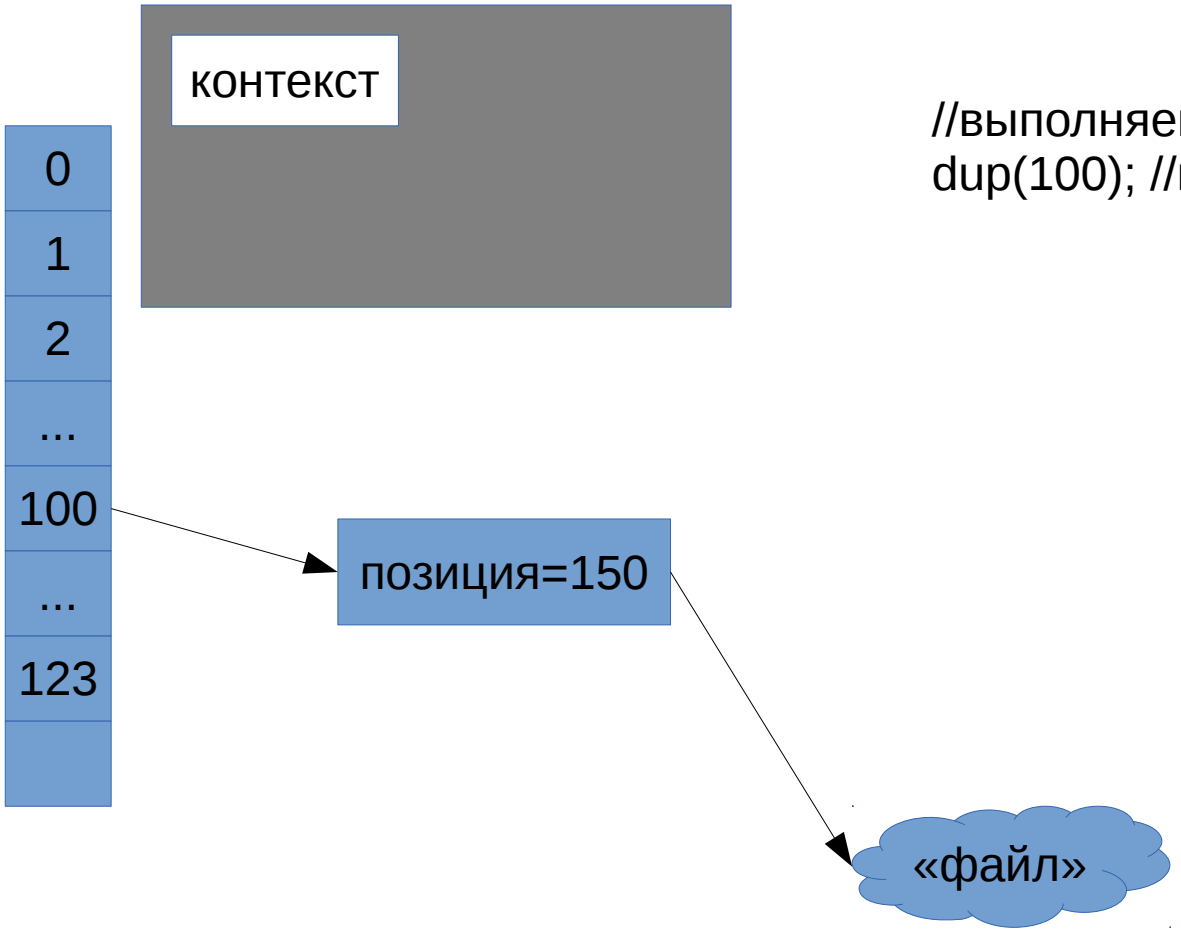
Изменение (lseek/read/write)
в одном процессе приведет
к изменению позиции и в другом

Тяжелые процессы и дескрипторы



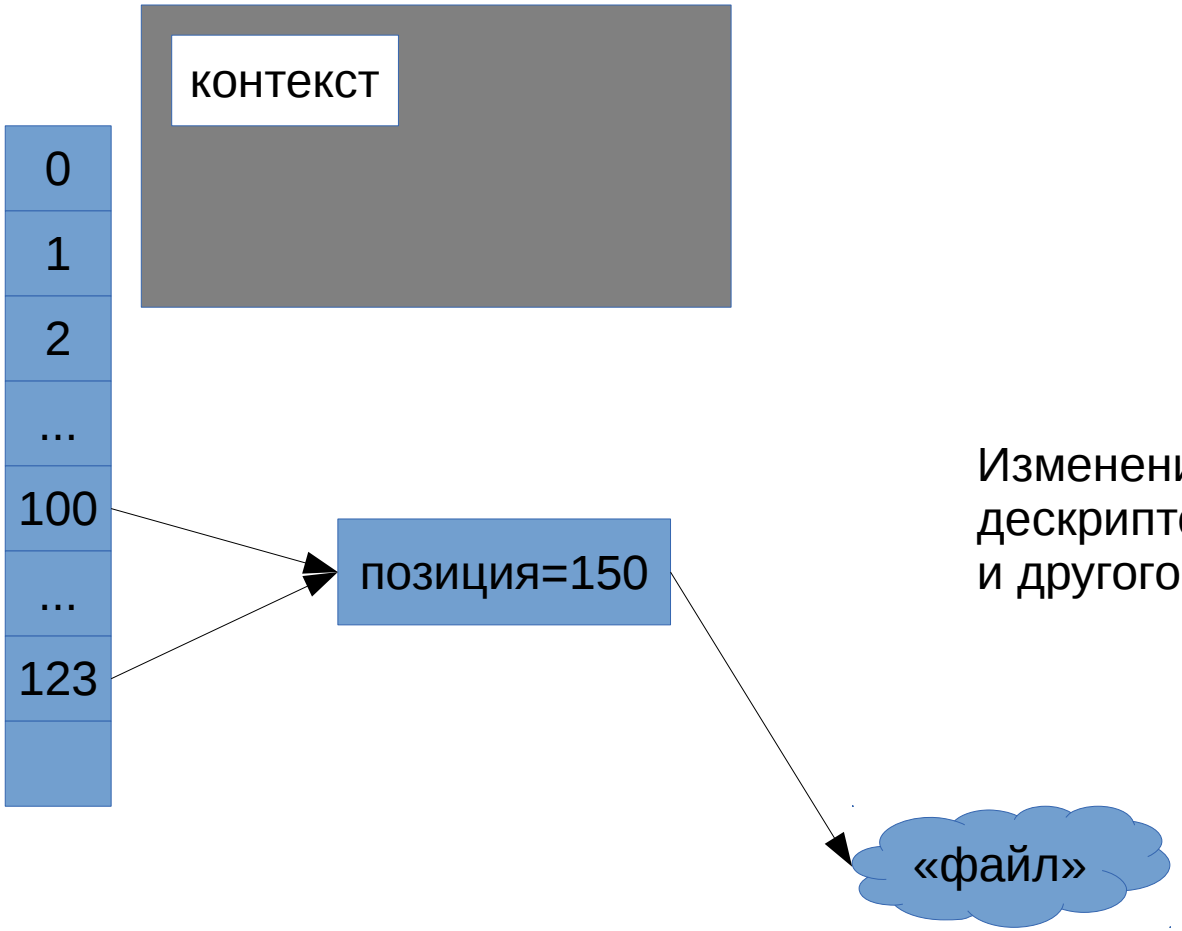
Close освобождает/закрывает только для своего процесса.

Дескрипторы и dup



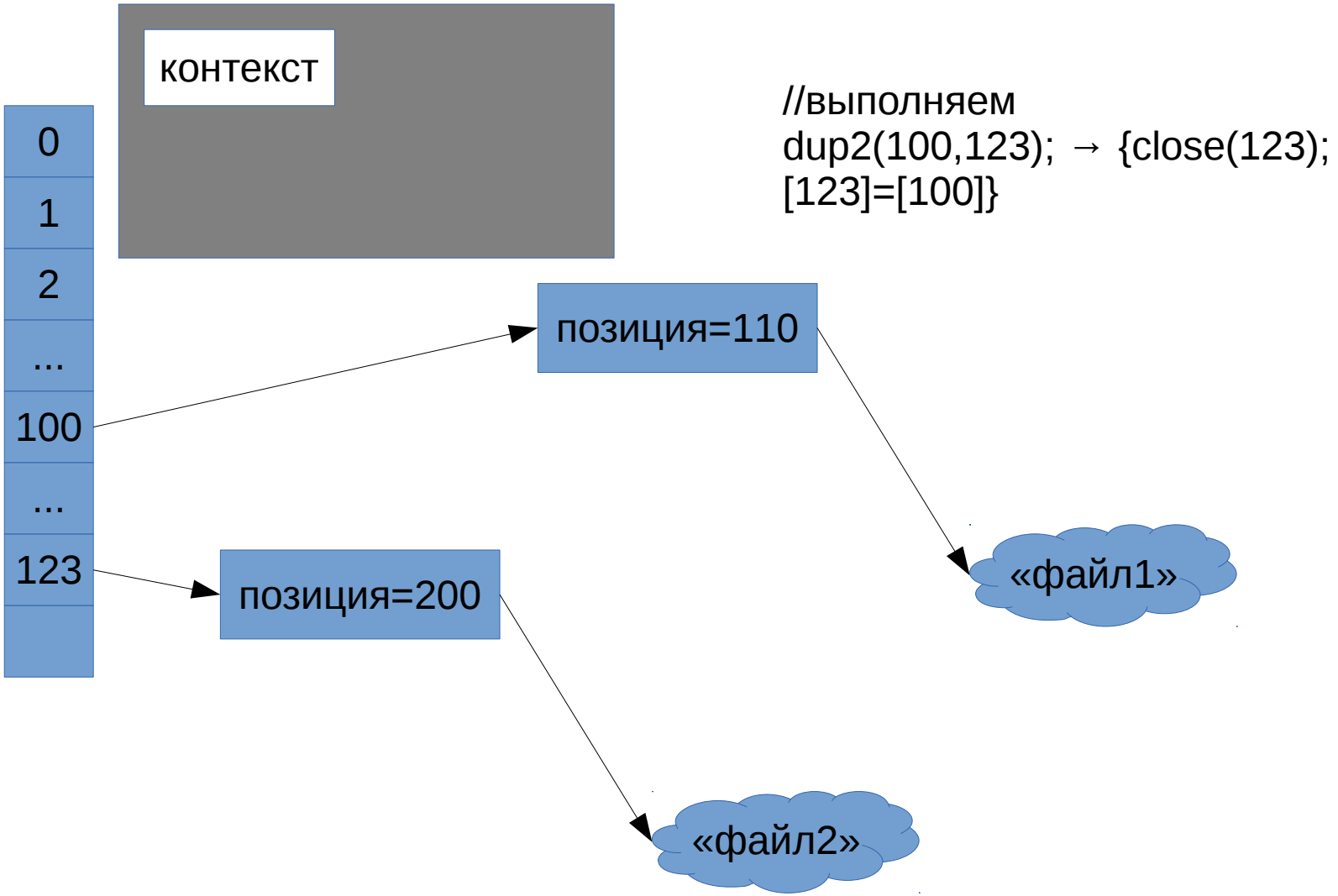
//выполняем
dup(100); //пусть вызов вернёт 123

Выполнили dup



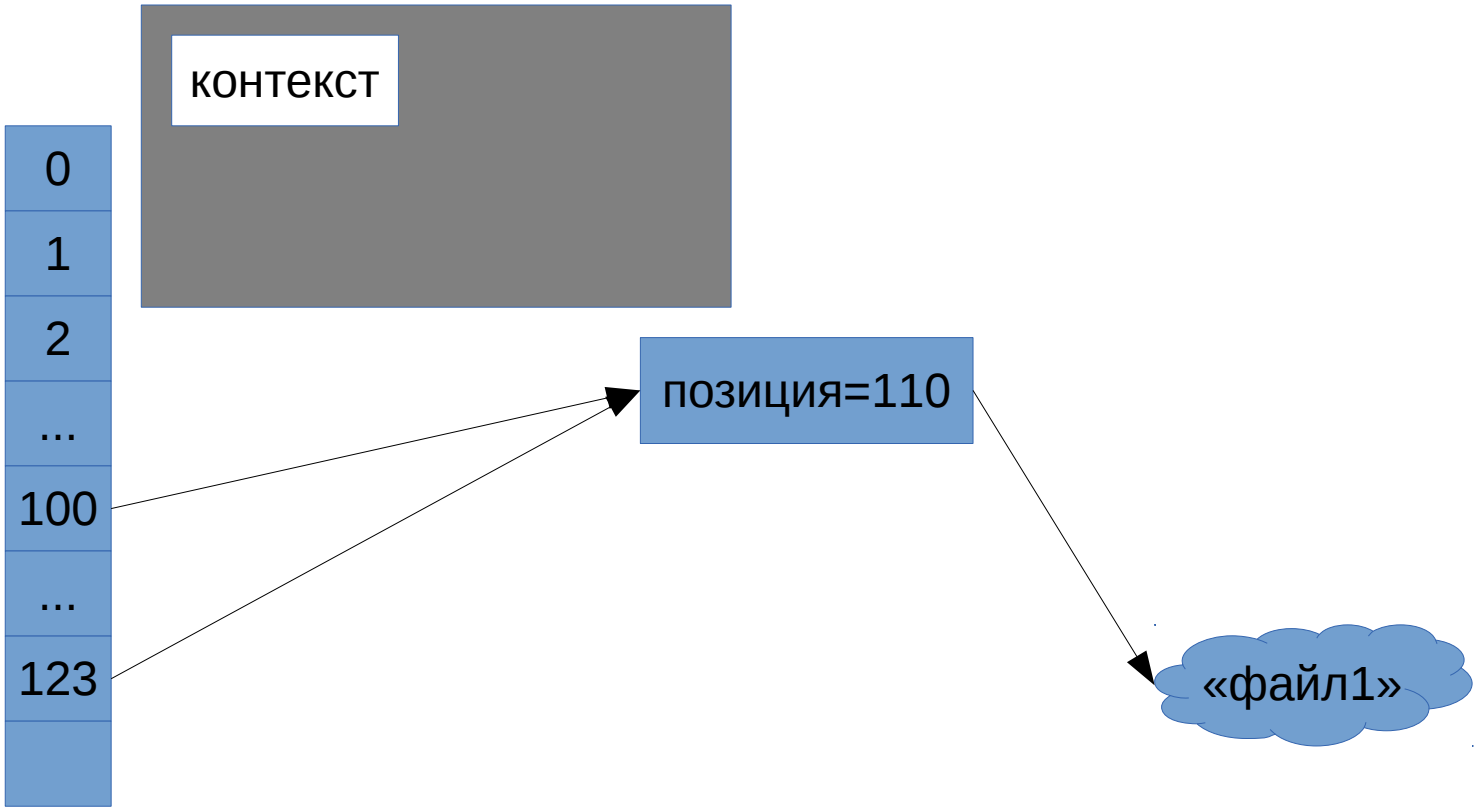
Изменение позиции по одному из дескрипторов приведет к изменению и другого

Дескрипторы и dup2

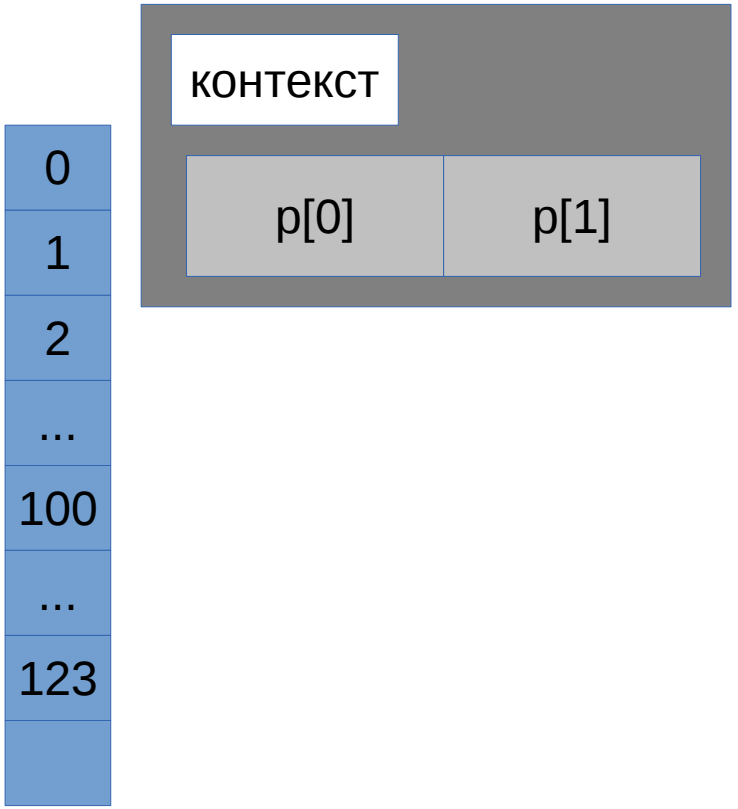


//выполняем
dup2(100,123); → {close(123);
[123]=[100]}

Выполнили dup2

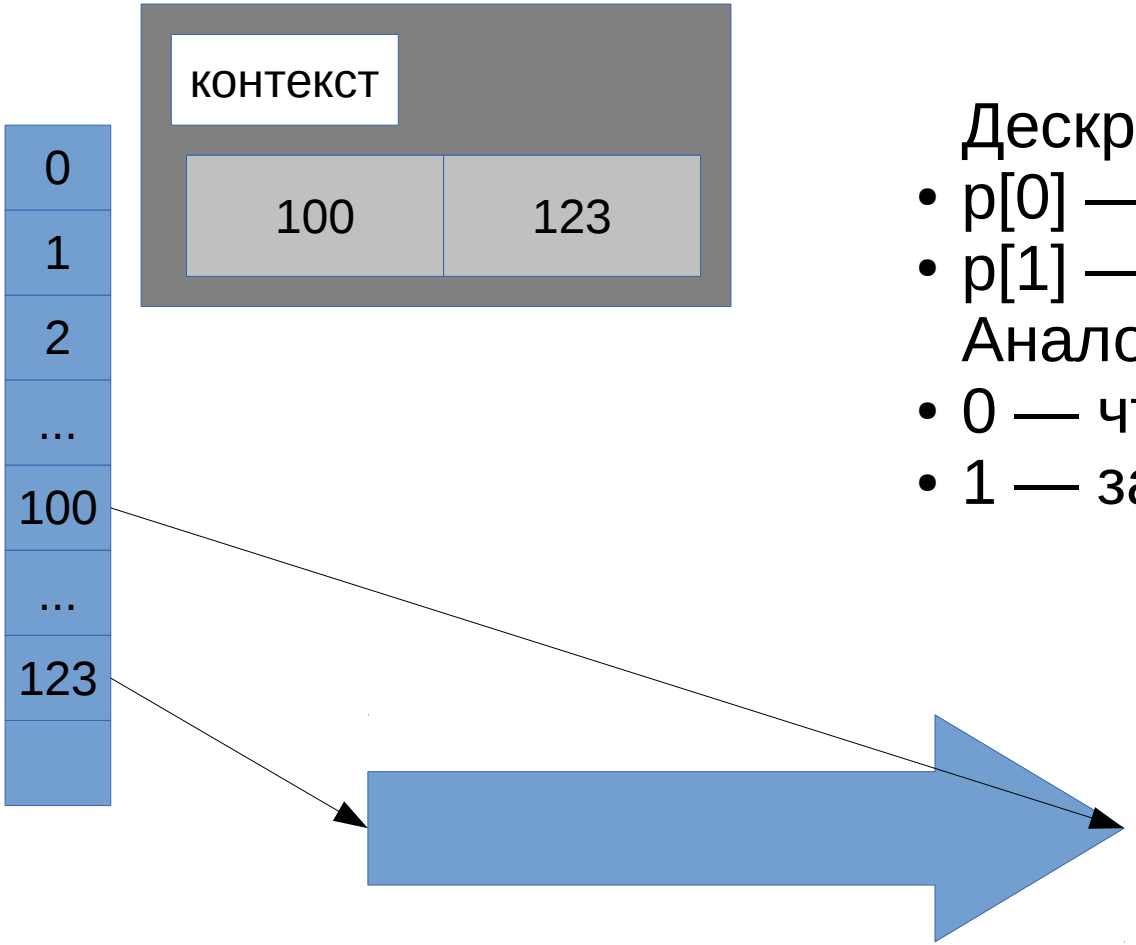


pipe (man 2 pipe)




```
int p[2]; //int *p
pipe(p);//p[0]=100,p[1]=123
```

pipe



Дескрипторы:

- p[0] — чтение
- p[1] — запись

Аналогично:

- 0 — чтение
- 1 — запись

Непрерывный поток: записали 123 и 456, считать можем 1, 234, 5, ...

Именованные каналы

- int mkfifo(const char *pathname, mode_t mode);
- mkfifo [OPTION]... NAME...

Сигналы (варианты, bsd-name)

SIGFPE /* floating point exception */

SIGILL /* illegal instr. (not reset when caught) */

SIGBUS /* bus error */

SIGSEGV /* segmentation violation */

SIGSYS /* non-existent system call invoked */

SIGPIPE /* write on a pipe with no one to read it */

SIGALRM /* alarm clock */

SIGCHLD /* to parent on child stop or exit */

SIGUSR1 /* user defined signal 1 */

SIGUSR2 /* user defined signal 2 */

SIGHUP /* hangup */

SIGINT /* interrupt */

SIGKILL /* kill (cannot be caught or ignored) */

SIGQUIT /* quit */

SIGCONT /* continue a stopped process */

SIGTERM /* software termination signal from kill */

SIGSTOP /* sendable stop signal not from tty */

SIGTSTP /* stop signal from tty */

Сигналы (вызовы)

- int kill(pid_t pid, int sig);
- void (*signal(int sig, void (*func)(int)))(int);

SIG_DFL
SIG_IGN
SIG_ERR (служебный для signal)

Через typedef

- typedef void (*sig_t) (int);
- sig_t signal(int sig, sig_t func);

SIG_DFL
SIG_IGN
SIG_ERR (служебный для signal)

Проблемы

- Как должна вести себя система, если во время обработки сигнала придет другой сигнал.
- Предыдущий обработчик ставится после первого вызова пользовательского обработчика или нет.
- Какие сигналы обрабатываются, а какие нет
- Чтобы сигнал не прервал процесс, надо установить обработчик на каждый сигнал
- Один сигнал может прийти по разным причинам, как установить причину
- Как дождаться сигнала (while(1);?)
  (есть pause)

Новые интерфейсы

- int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
- int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
- int sigpending(sigset_t *set);
- int sigsuspend(const sigset_t *mask);