Разработка программного обеспечения ОС UNIX

Передача информации между процессами

Неименованные каналы

```
#include <unistd.h>

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

fildes[0] – для чтения;
fildes[1] – для записи.
В SVR4 – двунаправленные.

Определение направления канала – закрытие ненужного дескриптора.

lseek использовать нельзя.

Размер канала PIP_BUF байт.

Чтобы читающая сторона увидела EOF, пишущая закрывает канал.

Пример использования канала

```
int fildes[2];
const int BSIZE = 100;   char buf[BSIZE];   ssize_t nbytes;

int status = pipe(fildes);
if (status == -1 ) {    /* an error occurred */  }

switch (fork()) {
    case -1:          /* Процесс не создан */    break;
    case 0:           /* Дочерний процесс читает из канала */
       close(fildes[1]);
       nbytes = read(fildes[0], buf, BSIZE);
       close(fildes[0]);
       exit(EXIT_SUCCESS);

    default:          /* Родительский процесс передает данные дочернему */
       close(fildes[0]);
       write(fildes[1], "Hello child\n", 12 );
       close(fildes[1]);                    /* Чтобы показать, что данные все.*/
       exit(EXIT_SUCCESS);
    }
```

Каналы и конвейеры

```
int  fifo[2];            /*ПРОВЕРКА НА ОШИБКИ УСЛОВНО НЕ ПОКАЗАНА! */
pid_t pid1, pid2;
pipe( fifo );
switch( pid1 = fork() ) {
    case 0:
            dup2( fifo[1], STDOUT_FILENO );
            close( fifo[0]); close( fifo[1] );
            execlp( программа1 );
}
switch( pid2 = fork() ) {
    case 0:
            dup2( fifo[0], STDIN_FILENO );
            close( fifo[0]); close( fifo[1] );
            execlp( программа2 );
}
close( fifo[0]); close( fifo[1] );
waitpid( pid1, status1, 0 ); waitpid( pid2, status2, 0 );
```

Именованные каналы

Связь неродственных процессов

Имеют имя в файловой системе (тип p в ls)

```
#include <sys/stat.h>
int mkfifo(const char *path, mode_t mode);
```

Действует как open с флагами O_CREAT|O_EXCL

Если канал уже существует – EEXIST.

Затем открыть или не чтение, или на запись.

Пример. ----- ВОЗМОЖНЫЕ ОШИБКИ ИГНОРИРУЮТСЯ! -----

```
mkfifo(“/tmp/fifo1”, S_IWUSR | S_IRUSR |S_IRGRP | S_IROTH);
rfd =open(“/tmp/fifo1”, O_RDONLY|O_NONDLOCK);
while( read( rfd, buf, sizeof(buf) ) == -1 && errno==EAGAIN ) sleep(1);
while( read( rfd, buf, sizeof(buf) ) > 0 ) {          …          }
close( rfd );
unlink(“/tmp/fifo1” );
```

Неблокируемый дескриптор

а) флаг O_NONBLOCK в open;

б) установка после открытия:

```
flags = fcntl( fd, F_GETFL, 0 );
flags |= O_NONBLOCK;
fcntl( fd, F_SETFL, flags );
```

Особенности чтения:

1. Возвращается имеющийся объем данных, даже если запрашивается больше.
2. Если записывается не больше чем PIPE_BUF, гарантируеися атомарность записи.
3. Если больше – сколько помещается и возвращается число реально записанных. Нет места – EAGAIN.
4. При записи в неоткрытый для чтения канал – SIGPIPE.
5. Если процесс игнорирует или перехватывает SIGPIPE – errno == EPIPE.

Сообщения SVR4

```
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(key_t key, int msgflg);     // Возвращает дескриптор очереди
```

Key: 1. IPC_PRIVATE
2. key_t ftok(const char *path, int id);

msgflg: 1. IPC_CREAT | IPC_EXCL | права
2. 0.

Время жизни – ядро.

Посмотреть –

ipcs [-qms] [-a|-bcopt]

Удалить –

ipcrm [-q msgid|-Q msgkey|-s semid|-S semkey|-m shmid|-M shmkey]....

Передача сообщений

Отправка сообщения

```
int msgsnd( int msqid, const void *msgp, size_t msgsz, int msgflg );   // 0  |  -1
struct mymsg {
    long   mtype;       /* Тип сообщения  - должен быть больше 0  */
    char   mtext[1];    /* Текст сообщения  [MSGMAX] */
}
```

Приём сообщения

```
ssize_t msgrcv( int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg );
```

Возвращает размер полученного сообщения.

Тип сообщения:

0	первое сообщение в очереди
>0	первое сообщение указанного типа
<0	первое сообщение наименьшего типа, меньше или равного заданному модулем msgtyp.

msgflg: 0 или IPC_NOWAIT MSG_NOERROR-отбрасывать лишние байты

Управление очередью

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

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

Команды:

IPC_RMID – удалить

IPC_STAT – получить информацию, buf → msqid_ds

IPC_SET – установить атрибуты, buf → msqid_ds (msg_perm, msg_qnum)

```
struct msqid_ds {
    struct ipc_perm   msg_perm;     // поля uid, gid, mode – права доступа
    msgqnum_t     msg_qnum;         // число сообщений в очереди
    msglen_t      msg_qbytes;       // Макс. число байт в очереди
    pid_t         msg_lspid;        // Процесс, последним отправивший
    pid_t         msg_lrpid;        // Процесс, последним получивший
    time_t        msg_stime;        // Время последнего отправления
    time_t        msg_rtime;        // Время последнего получения
    time_t        msg_ctime;        // Время последнего изменения
}
```

Сообщения стандарта POSIX

```
#include <mqueue.h>

mqd_t mq_open(const char *name, int oflag, ...);
```

Имя “/…..” – должно начинаться с /

oflag – флаги как у open: O_RDONLY O_WRONLY O_RDWR
O_CREAT O_EXCL O_NONBLOCK

Если флаг O_CREAT, дополнительные аргументы:

mode_t mode – права доступа
mq_attr attr – атрибуты очереди.

```
struct mq_attr {
    long    mq_flags;       // флаги
    long    mq_maxmsg;      // макс. число сообщений
    long    mq_msgsize;     // макс. размер сообщения.
    long    mq_curmsgs;     // число сообщений в очереди.
}
```

Время жизни – ядро.

Отправка сообщения

```
#include <mqueue.h>
#include <time.h>

int mq_send( mqd_t mqdes, const char *msg_ptr, size_t msg_len,
     unsigned msg_prio);

int mq_timedsend( mqd_t mqdes, const char *msg_ptr, size_t msg_len,
      unsigned msg_prio, const struct timespec *abstime);
```

Возвращают 0/-1.

Таймаут отсчитывается с использованием CLOCK_REALTIME.

Приоритет сообщения:

меньше чем MQ_PRIO_MAX.

Чем больше msg_prio, тем приоритет выше (раньше в очереди).

С равным приоритетом – в порядке поступления.

Получение сообщения

```
#include <mqueue.h>
ssize_t mq_receive( mqd_t mqdes, char *msg_ptr, size_t msg_len,
      unsigned *msg_prio);

#include <mqueue.h>
#include <time.h>
ssize_t mq_timedreceive( mqd_t mqdes, char *msg_ptr, size_t msg_len,
     unsigned *msg_prio,  const struct timespec *abstime);
```

Возвращают размер сообщения, не более SSIZE_MAX.

Получают самое старое сообщение наибольшего приоритета.

Таймаут отсчитывается с использованием CLOCK_REALTIME

Оповещение о сообщении:

```
int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

Сообщения POSIX: служебные функции

```
#include <mqueue.h>

int mq_getattr( mqd_t mqdes, struct mq_attr *mqstat);    //  0/-1

int mq_setattr( mqd_t mqdes, const struct mq_attr *mqstat,
     struct mq_attr *omqstat );

int mq_close(mqd_t mqdes);           // отсоединяет от дескриптора

int mq_unlink(const char *name);     // удаляет очередь из памяти
```

Разделяемая память System V R4. ч.1

#include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg);

key – получить с использованием ftok или IPC_PRIVATE
shmflg – флаги IPC_CREAT | IPC_EXCL | права

Присоединение к разделяемой памяти
void *shmat(int shmid, const void *shmaddr, int shmflg);

Возвращает указатель на разделяемую ппмять

Обычно shmaddr=NULL или подсказка, где отобразить
shmflg SHM_RND – округление до страницы
SHM_RDONLY – только для чтения

int shmdt(const void *shmaddr); - отсоединение от разделяемой памяти

Разделяемая память SVR4. ч.2

Управление памятью:

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
                          cmd:   IPC_STAT   IPC_SET   IPC_RMID

struct  shmid_ds {
    struct ipc_perm shm_perm;       // Права доступа
    size_t         shm_segsz;       // Размер в байтах
    pid_t          shm_lpid;        // Последний работавший с памятью
    pid_t          shm_cpid;        // Процесс-создатель
    shmatt_t       shm_nattch;      // Число подсоединившихся процессов
    time_t         shm_atime;       // Время последнего присоединения
    time_t         shm_dtime;       // Время последнего отсоединения
    time_t         shm_ctime;       // Время последнего изменения
    атрибутов
};
```

Время жизни – ядро.

Отображение файлов в память

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

void *mmap( void *addr,     // NULL или точный адрес (кратный размеру стр.)
            size_t len,     // размер отображения
            int prot,       // PROT_READ, PROT_WRITE, PROT_EXEC
            int flags,      // MAP_FIXED, MAP_SHARED, MAP_PRIVATE
            int fildes,     // описатель файла
            off_t off);     // позиция в файле
```

Возвращает указатель на отображение

Отключить отображение `int munmap(void *addr, size_t len);` // 0/-1

Анонимное отображение (BSD) MAP_ANON, fildes=-1

(AT&T) открыть /dev/zero

Синхронизация отображения

```
int msync(void *addr, size_t len, int flags);    // 0/-1
                    MS_ASYNC,  MS_SYNC,  MS_INVALIDATE
```

Разделяемая память POSIX

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

```
int shm_open( const char *name, int oflag, mode_t mode ); // -1 - ошибка
```

Имя – “/…”.

Флаги O_RDONLY, O_RDWR, O_CREAT, O_EXCL, O_TRUNC

mode – права доступа.

Время жизни – ядро.

Удалить: `int shm_unlink(const char *name);`

Схема применения:

shm_open	создать
ftruncate	установить размер
mmap	отобразить в память
	используем разделяемую память
munmap	отменить отображение
shm_unlink	удалить память

Разделяемая память POSIX. Пример

```
#include <unistd.h>
#include <sys/mman.h>
#define MAX_LEN 10000
struct region { /* Структурируем память */
      int len;
      char buf[MAX_LEN];
};
struct region *rptr;    int fd;

fd = shm_open("/myregion", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
if ( fd == -1 ) /* Не получилось создать память */;

if ( ftruncate(fd, sizeof(struct region)) == -1 )  /* размер не задали */

rptr = mmap(NULL, sizeof(struct region), PROT_READ | PROT_WRITE, MAP_SHARED,
      fd, 0);
if (rptr == MAP_FAILED) /* Не отобразилось */;
rptr->len = sprintf( rptr->buf, “%s”, “Hello world!”);
```

Семафоры SVR4 (XSI)

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

int semget( key_t key,      IPC_PRIVATE, ftok()
            int nsems,      число семафоров в наборе
            int semflg);    IPC_CREAT|права
```

Возвращает идентификатор набора семафоров

```
int semop( int              semid,   идентификатор набора семафоров
           struct sembuf    *sops,   массив операций
           size_t           nsops);  число элементов массива

struct sembuf {
    short sem_num;      // Номер семафора в наборе
    short sem_op;       // Действие –N (уменьш.), +N (увел.), 0
    short sem_flg;      // IPC_NOWAIT,   SEM_UNDO
};
```

Время жизни – ядро.

Управление семафорами SVR4

```
int semctl( int semid, int semnum, int cmd, ...);
                        GETVAL  SETVAL   GETPID   GETNCNT   GETZCNT
                        GETALL   SETALL   IPC_STAT   IPC_SET   IPC_RMID
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
} arg;   // четвёртый аргумент
struct semid_ds {
      struct ipc_perm  sem_perm;
      unsigned short   sem_nsems;  // Число семафоров в наборе
      time_t               sem_otime;    // Время последней semop, иниц. 0
      time_t               sem_ctime;    // Время последней semctl
 };
struct {
      unsigned short  semval;    //   Значение семафора
      pid_t               sempid;    //   процесс, выполнивший последнюю операцию
      unsigned short  semncnt;  // Сколько процессов ждут увеличения семафора
      unsigned short  semzcnt;  //  Сколько процессов ждут обнуления семафора
};
```

Семафоры POSIX

```
#include <semaphore.h>
sem_t *sem_open(const char *name, int oflag, [mode_t mode, unsigned value]);
                          "/…"              O_CREAT, O_EXCL
```

Возвращает адрес семафора или SEM_FAILED.

Альтернатива:

```
int sem_init( sem_t *sem, int pshared, unsigned value);

int sem_wait( sem_t *sem );
int sem_trywait( sem_t *sem );
int sem_timedwait( sem_t *sem, const struct timespec *abstime);

int sem_post( sem_t *sem );

int sem_getvalue( sem_t *sem, int *sval );

int sem_close(sem_t *sem);          // Отсоединяет от процесса
int sem_unlink(const char *name);   // Для именованных семафоров
int sem_destroy(sem_t *sem);        // Для неименованных семафоров
```