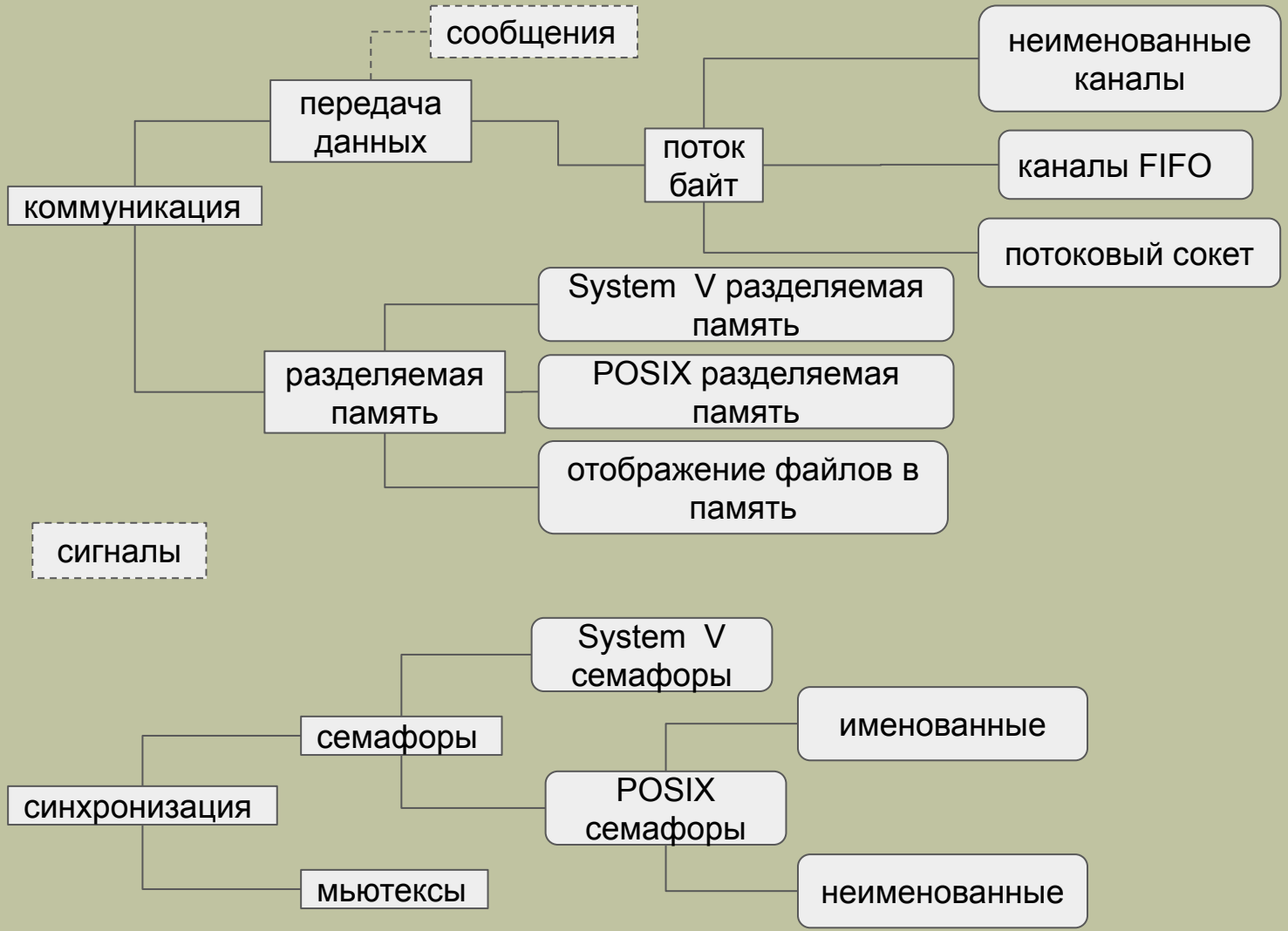
Лекция 12

- Межпроцессное взаимодействие (*IPC*).
- Разделяемая память и семафоры *SysV*.


коммуникация
передача данных
сообщения
поток байт
неименованные каналы
каналы FIFO
потоковый сокет
разделяемая память
System V разделяемая память
POSIX разделяемая память
отображение файлов в память
сигналы
синхронизация
семафоры
System V семафоры
POSIX семафоры
именованные
неименованные
мьютексы

Программные интерфейсы для System V объектов

Интерфейс	**Семафоры**	**Разделяемая память**
Заголовочный файл Структура данных Создание/Открытие Закрытие объекта Операции управления Выполнение IPC	*<sys/sem.h>* *semid_ds* *semget()* - *semctl()* *semop()* semaphore	<sys/shm.h> Shmid_ds shmget()+shmat() shmdt() shmctl() Доступ к памяти в разделяемой области

lab12d.h

```
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
typedef enum { FALSE, TRUE } Boolean;
#define SHM_KEY 0x1234
#define SEM_KEY 0x5678
#define OBJ_PERMS (S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP)
#define WRITE_SEM 0
#define READ_SEM   1
#define BUF_SIZE 1024
extern Boolean bsUseSemUndo;
extern Boolean bsRetryOnEintr;
```

```
struct shmseg {
int cnt;
char buf[BUF_SIZE];
};

union semun { //используется в вызовах semctl(...)
  int val;
  struct semid_ds *buf;
  unsigned short *array;
  struct seminfo * __buf;
};
```

```
int reserveSem(int semId, int semNum);
int releaseSem(int semId, int semNum);
int initSemAvailable(int semId, int semNum);
int initSemInUse(int semId, int semNum);
```

lab12d3.c

```
#include "lab12d.h"
Boolean bsUseSemUndo = FALSE;
Boolean bsRetryOnEintr = TRUE;

int initSemAvailable(int semId, int semNum){
  union semun arg;
  arg.val = 1;
  return semctl(semId, semNum, SETVAL, arg);
}

int initSemInUse(int semId, int semNum){
  union semun arg;
  arg.val = 0;
  return semctl(semId, semNum, SETVAL, arg);
}
```

```
int reserveSem(int semId, int semNum){
  struct sembuf sops;

  sops.sem_num = semNum;
  sops.sem_op = -1;
  sops.sem_flg = bsUseSemUndo ? SEM_UNDO : 0;

  while (semop(semId, &sops, 1) == -1)
    if (errno != EINTR || !bsRetryOnEintr)
      return -1;

  return 0;
}
```

```
int releaseSem(int semId, int semNum){
  struct sembuf sops;

  sops.sem_num = semNum;
  sops.sem_op = 1;
  sops.sem_flg = bsUseSemUndo ? SEM_UNDO : 0;

  return semop(semId, &sops, 1);
}
```

lab12d1.c

```
#include "lab12d.h"
int main(int argc, char *argv[]){
  int semid, shmid, bytes, xfrs;
  struct shmseg *shmp;
  union semun dummy;

  semid = semget(SEM_KEY, 2, IPC_CREAT | OBJ_PERMS);
  if (semid == -1)
     fprintf(stderr,"semget");

  if(initSemAvailable(semid, WRITE_SEM) == -1)
    fprintf(stderr,"initSemAvailable");

  if (initSemInUse(semid, READ_SEM) == -1)
    fprintf(stderr,"initSemInUse");
```

```
shmid = shmget(SHM_KEY, sizeof(struct shmseg),
                        IPC_CREAT|OBJ_PERMS);

if (shmid == -1)
   fprintf(stderr,"shmget");

 shmp = shmat(shmid, NULL, 0);
 if (shmp == (void *) -1)
       fprintf(stderr,"shmat");
```

```
for (xfrs = 0, bytes = 0; ; xfrs++, bytes += shmp->cnt) {

  if (reserveSem(semid, WRITE_SEM) == -1)
    fprintf(stderr,"reserveSem");

  shmp->cnt = read(STDIN_FILENO, shmp->buf, BUF_SIZE);
  if (shmp->cnt == -1)
    fprintf(stderr,"read");

  if (releaseSem(semid, READ_SEM) == -1)
    fprintf(stderr,"releaseSem");  /* Give reader a turn */

  if (shmp->cnt == 0)
    break;
}
```

```
 if (reserveSem(semid, WRITE_SEM) == -1)
      fprintf(stderr,"reserveSem");

  if (semctl(semid, 0, IPC_RMID, dummy) == -1)
    fprintf(stderr,"semctl");

  if (shmdt(shmp) == -1)
    fprintf(stderr,"shmdt");

  if (shmctl(shmid, IPC_RMID, 0) == -1)
    fprintf(stderr,"shmctl");

  fprintf(stderr, "Sent %d bytes (%d xfrs)\n", bytes, xfrs);
  exit(EXIT_SUCCESS);
}
```

lab12d2.c

```
#include "lab12d.h"
int main(int argc, char *argv[]){
  int semid, shmid, xfrs, bytes;
  struct shmseg *shmp;

  semid = semget(SEM_KEY, 0, 0);
  if (semid == -1)
     fprintf(stderr,"semget");

  shmid = shmget(SHM_KEY, 0, 0);
  if (shmid == -1)
     fprintf(stderr,"shmget");

  shmp = shmat(shmid, NULL, SHM_RDONLY);
  if (shmp == (void *) -1)
     fprintf(stderr,"shmat");
```

```
for (xfrs = 0, bytes = 0; ; xfrs++) {
   if (reserveSem(semid, READ_SEM) == -1)
    fprintf(stderr,"reserveSem");

   if (shmp->cnt == 0)
     Break;

   bytes += shmp->cnt;

   if (write(STDOUT_FILENO, shmp->buf, shmp->cnt) != shmp->cnt)
    fprintf(stderr,"partial/failed write");

   if (releaseSem(semid, WRITE_SEM) == -1)   /* Give writer a turn */
    fprintf(stderr,"releaseSem");
 }
```

```
 if (shmdt(shmp) == -1)
    fprintf(stderr,"shmdt");

  if (releaseSem(semid, WRITE_SEM) == -1)
    fprintf(stderr,"releaseSem");

  fprintf(stderr, "Received %d bytes (%d xfrs)\n", bytes, xfrs);

  exit(EXIT_SUCCESS);
}
```

```
~/Лекция12/shm_sysvd> ls  |  ./lab12d1
```

```
~/Лекция12/shm_sysvd> ./lab12d2
lab12d1
lab12d1.c
lab12d2
lab12d2.c
lab12d3.c
lab12d.h
Received 55 bytes (1 xfrs)
```

```
00400000-00401000 r-xp 00000000 08:13 33023725     ~/lab12d1
00601000-00602000 r--p 00001000 08:13 33023725     ~/lab12d1
00602000-00603000 rw-p 00002000 08:13 33023725     ~/lab12d1
7f852de6b000-7f852e01c000 r-xp 00000000 00:2d 18863
/lib64/libc-2.26.so
……
7f852e41b000-7f852e41d000 rw-p 00000000 00:00 0
7f852e44a000-7f852e44b000 rw-s 00000000 00:01 163853
/SYSV0000 1234 (deleted)
7f852e44c000-7f852e44d000 rw-p 00026000 00:2d 18855
/lib64/ld-2.26.so
7ffdf17c8000-7ffdf17ea000 rw-p 00000000 00:00 0   [stack]
```

```
~/Лекция12/shm_sysvd> ipcs -a

------ Сегменты совм. исп. памяти --------
ключ       shmid    владелец    права   байты nattch
                                                состояние
0x00001234 163863     malkov     660         1028       1
```