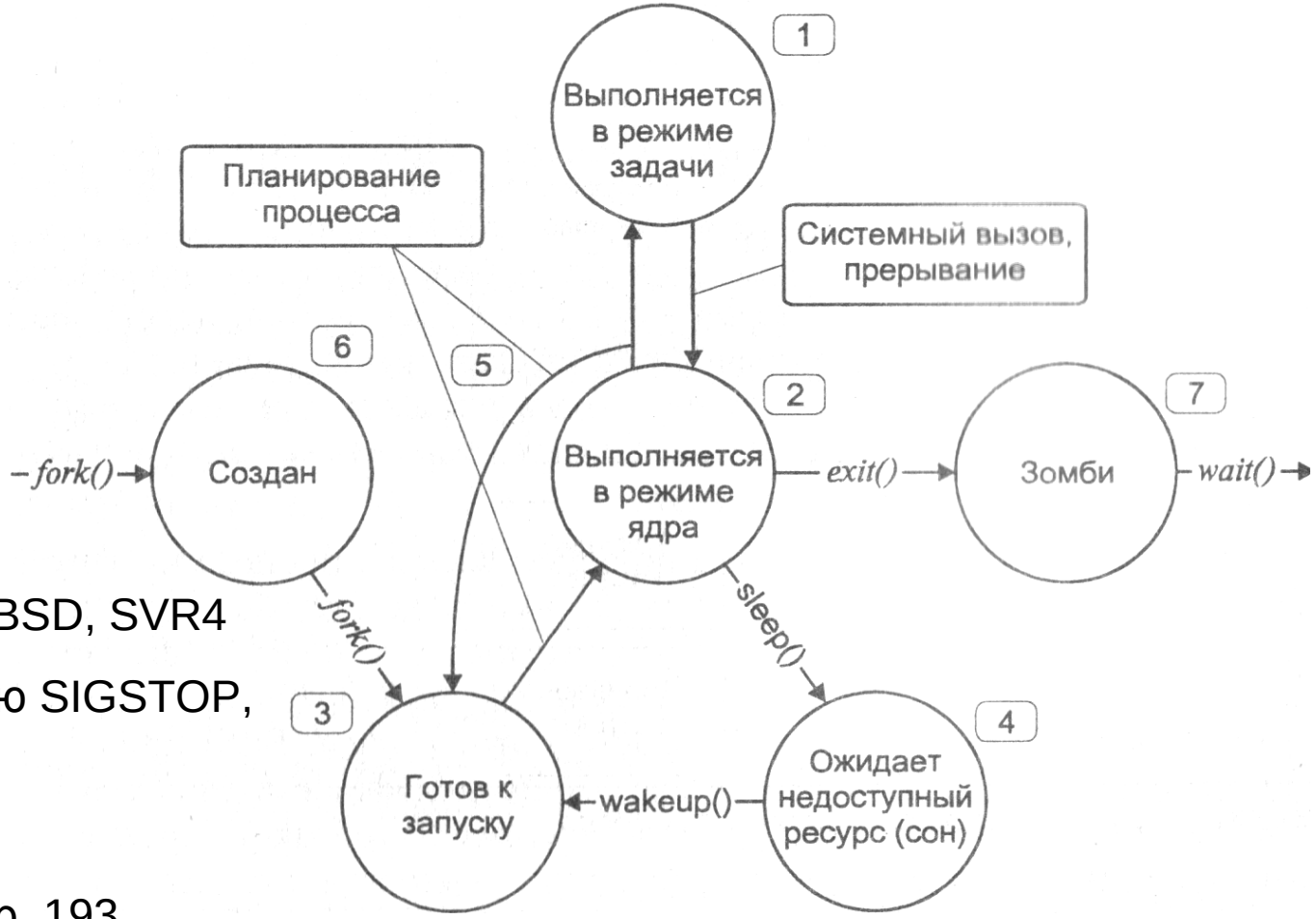
Системные вызовы, задачи для лабораторных

- Работа с файловой системой
  - Чтение и запись
  - Атрибуты (метаданные)
- Программирование процессов
  - Состояния процессов в ОС UNIX
    - Создание процесса, запуск программы, завершение выполнения процесса
    - Управление: квоты
  - Взаимодействие процессов:
    - Сигналы (надежные, ненадежные)
    - каналы (pipes)
    - именованные каналы (FIFO, named pipes)
    - SystemV IPC: семафоры, очереди сообщений, разделяемая память
- Сетевое взаимодействие процессов
    - TCP/IP, сокеты (sockets)
    - потоки (streams), TLI
    - Удаленный вызов процедур RPC
    - X-Window

Основные системные функции работы с файлами

Системная функция	Описание
open	Служит для получения доступа на чтение и/или запись к указанному файлу. Если файл существует, он открывается, и процессу возвращается файловый дескриптор, адресующий дальнейшие операции с файлом. Если файл не существует, он может быть создан
creat	Служит для создания файла
close	Закрывает файловый дескриптор, связанный с предварительно открытым файлом
dup	Возвращает дубликат файлового дескриптора
dup2	Возвращает дубликат файлового дескриптора, но позволяет явно указать его значение
lseek	Устанавливает файловый указатель на определенное место файла. Дальнейшие операции чтения/записи будут производиться, начиная с этого смещения
read	Производит чтение указанного количества байтов из файла
readv	Производит несколько операций чтения указанного количества байтов из файла
write	Производит запись указанного количества байтов в файл
writev	Производит несколько операций записи указанного количества байтов в файл
pipe	Создает коммуникационный канал, возвращая два файловых дескриптора
fcntl	Обеспечивает интерфейс управления открытым файлом

Состояния процесса



Доп. состояния 4.xBSD, SVR4

Переход с помощью SIGSTOP, SIGTTIN, SIGTTOU

Выход - SIGCONT

А. Робачевский, стр. 193

Создание процессов в ОС UNIX

```
main ()
{
int pid;

pid = fork();

if (pid == 0) printf("I am child with PID=$d\n“,pid);
else printf("I am parent with PID=%d\n“,pid);
}
```

в код добавлены небольшие преднамеренные ошибки

Взаимодействие процессов

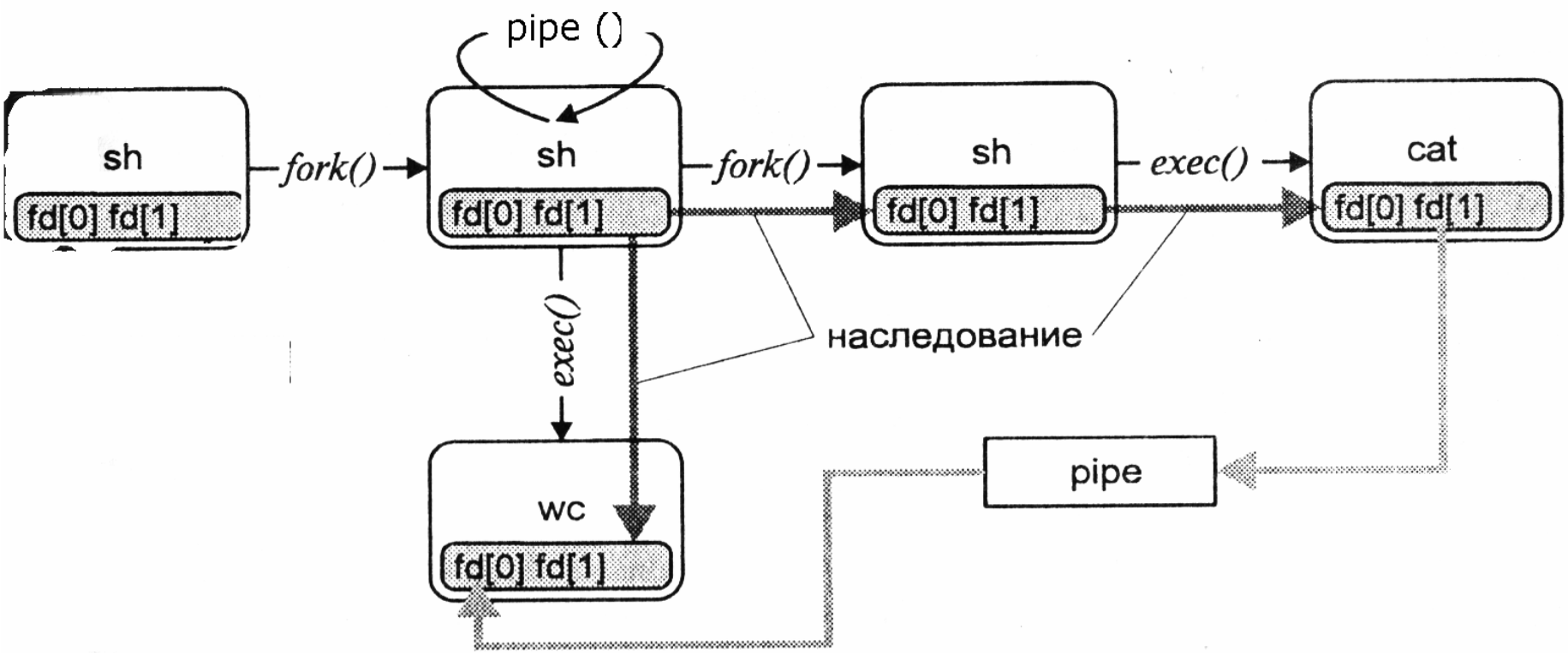
- каналы (pipes)
- именованные каналы (FIFO, named pipes)
- сигналы
- SystemV IPC: семафоры, очереди сообщений, разделяемая память
- сокеты (BSD sockets)

Каналы. Пример: cat filename | wc

Каналы (неименованные каналы) основаны на pipe (int *filedes)

filedes[0] – запись в канал; filedes[1] - чтение из канала

Область применения – родственные процессы



Именованные каналы (fifo,сервер)

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#define FIFO "fifo.1"
#define MAXBUFF 80

main()
{
        int readfd, n;

        char buff[MAXBUFF];
        mknod(FIFO,S_IFIFO|0600,0);
        readfd=open(FIFO,O_RDONLY);
        while((n=read(readfd,buff,MAXBUFF))>0) write(1,buff,n);
        close(readfd);
        exit(0);
}
```

используйте mkfifo()

Именованные каналы (fifo, клиент)

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#define FIFO "fifo.1"

main()
{
        int writefd;

        writefd=open(FIFO,O_WRONLY);
        write(writefd, "Hi!\n", 4);
        close(writefd);
        unlink(FIFO);
        exit(0);
}
```

Ограничение ресурсов процесса (getrlimit(), setrlimit())

```
void disp(int resource, char *rname)
{
        struct rlimit rlim;
        printf("%s", rname);
        getrlimit(resource,&rlim);
        if (rlim.rlim_cur==RLIM_INFINITY)
                printf("\tinfinite\t");
        else
                printf("\t%d\t",rlim.rlim_cur);
        if (rlim.rlim_max==RLIM_INF)
                printf("infinity\n");
        else
                printf("%d\n",rlim.rlim_max);
}
```

```
#include<sys/resource.h>
#include<sys/types.h>
#include<unistd.h>


main()
{
        disp(RLIMIT_CPU,"CPU");
        disp(RLIMIT_CORE,"CORE");
        disp(RLIMIT_NOFILE,"FILES");
        disp(RLIMIT_NPROC,"NPROC");
}
```

Утилиты:

ulimit

в код добавлены небольшие преднамеренные ошибки … может быть.

system()

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

main()
{
        int fd;
        fd=creat("my_file",S_IRUSR|S_IWUSR|S_IXUSR);
        system("ls -l my_file");
        fchmod(fd,S_IRWXU|S_ISUID);
        system("ls -l my_file");
}
```

```
[test@nix] ./system
-rwx------    1 test     test            0 Apr  9 12:45 my_file
-rws------    1 test     test            0 Apr  9 12:45 my_file
```

Сигналы

```
#include <signal.h>

static void sig_hndlr(int signo)
{
signal(SIGINT, sig_hndlr);
printf ("Got Signal\n");
}

main() {

signal(SIGINT, sig_hndlr);
signal(SIGUSR1, SIG_DFL);
signal (SIGUSR2, SIG_IGN);

while(1)
pause() ;
}
```

Утилиты:

man **kill**

Надежные (reliable) сигналы

```
static void sig_hndlr(int signo)
{
        printf("Got signal!\n");
}

void(*mysignal(int signo,void(*hndlr)(int)))(int)
{
        struct sigaction act,oact;
        act.sa_handler=hndlr;
        sigemptyset(&act.sa_mask);
        act.sa_flags=0;
        if (signo!=SIGALRM)
                act.sa_flags|=SA_RESTART;
        sigaction(signo,&act,&oact);
}

main()
{
        mysignal(SIGINT,sig_hndlr);
        mysignal(SIGUSR1,SIG_DFL);
        mysignal(SIGUSR2,SIG_IGN);
        while(1) pause();
}
```

Для управления набором сигналов sigset_t применяют:
sigemptyset(), sigfillset(), sigaddset(), sigdelset().
Для управления диспозицией:
sigaction()

Структура sigaction:

```
struct sigaction {
  void (*sa_handler)(int);
  void (*sa_sigaction)(int, siginfo_t *, void *);
  sigset_t sa_mask;
  int sa_flags;
  void (*sa_restorer)(void);
}
```

Интерфейс сигналов POSIX.1, основанный на 4.2BSD

IPC SystemV, операции семафоров. Файл - shmem.h.

```
#define MAXBUFF 80
#define PERM 0600

typedef struct mem_msg
{
        int segment;
        char buff[MAXBUFF];
} Message;
```

Элементы списка операций:
- номер семафора;
- код операции;
- флаги.

- В данной примере:
- 0 – разблокировано
- 1 - заблокировано

```
static struct sembuf proc_wait[1]={1,-1,0};
static struct sembuf proc_start[1]={1,1,0};
static struct sembuf mem_lock[2]={0,0,0,0,1,0};
static struct sembuf mem_unlock[1]={0,-1,0};
```

См. подробнее (IPC SystemV):
http://www.citforum.ru/operating_systems/bach/glava_98.shtml

Коды операций и значения семафоров

- Ядро меняет значение семафора в зависимости от кода операции, который может быть равен 0, больше 0 или меньше 0
- Если код операции имеет положительное значение, ядро увеличивает значение семафора и выводит из состояния приостанова все процессы, ожидающие наступления этого события.
- Если код операции равен 0, процесс ожидает обнуления семафора
  - ядро проверяет значение семафора: если оно равно 0, ядро переходит к выполнению других операций;
  - в противном случае ядро увеличивает число приостановленных процессов, ожидающих, когда значение семафора станет нулевым, и процесс "засыпает".
- Если код операции имеет отрицательное значение и
  - если его абсолютное значение не превышает значение семафора, ядро прибавляет код операции (отрицательное число) к значению семафора. Если результат равен 0, ядро выводит из состояния приостанова все процессы, ожидающие обнуления значения семафора.
  - Если результат меньше абсолютного значения кода операции, ядро приостанавливает процесс до тех пор, пока значение семафора не увеличится. Если процесс приостанавливается посреди операции, он имеет приоритет, допускающий прерывания; следовательно, получив сигнал, он выходит из этого состояния.
- semop (дескриптор, структура с операциями, число операций)

IPC SystemV, сервер

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<sys/shm.h>
#include"shmem.h"

main()
{
        Message *msgptr;
        int semid,shmid;
        key_t key;

        key=ftok("srv",'A');
        shmid=shmget(key,sizeof(Message),PERM|IPC_CREAT);
        msgptr=(Message *)shmat(shmid,0,0);
        semid=semget(key,2,PERM|IPC_CREAT);
        semop(semid,&proc_wait[0],1);
        semop(semid,&mem_lock[0],2);
        printf("%s",msgptr->buff);
        semop(semid,&mem_unlock[0],1);
        shmdt(msgptr);
        exit(0);
}
```

Утилиты:

ipcs

Ipcrm

semtool

в код добавлены небольшие преднамеренные ошибки … может быть

IPC SystemV, клиент

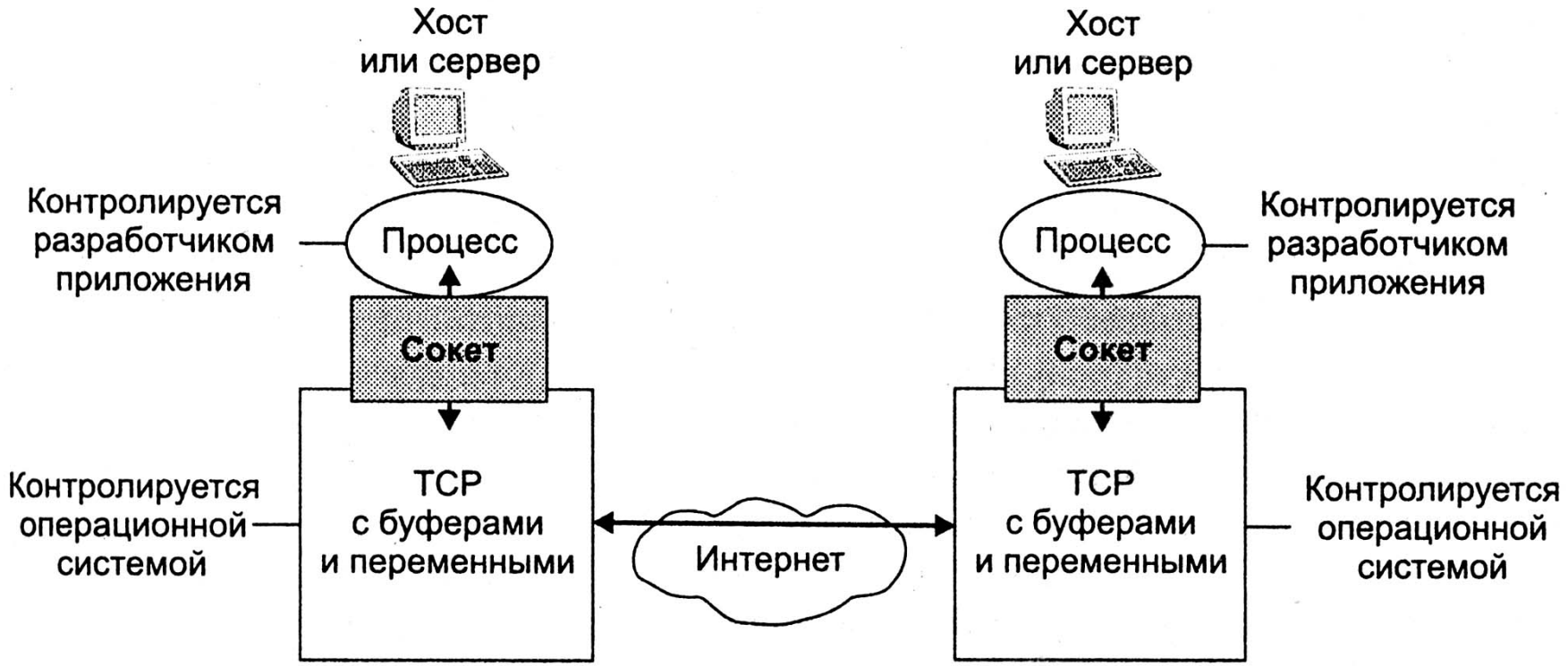
```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<sys/shm.h>
#include"shmem.h"

main()
{
        Message *msgptr;
        key_t key;
        int semid,shmid;

        key=ftok("server",'A');
        shmid=shmget(key,sizeof(Message),0);
        msgptr=(Message *)shmat(shmid,0,0);
        semid=semget(key,2,PERM);
        semop(semid,&mem_lock[0],2);
        semop(semid,&proc_start[0],1);
        sprintf(msgptr->buff,"Hi!\n");
        semop(semid,&mem_unlock[0],1);
        semop(semid,&mem_lock[0],2);
        shmdt(msgptr);
        shmctl(shmid,IPC_RMID,0);
        semctl(semid,0,IPC_RMID);
```

в код добавлены небольшие преднамеренные ошибки … может быть.

Процессы приложения, сокеты и протокол транспортного уровня

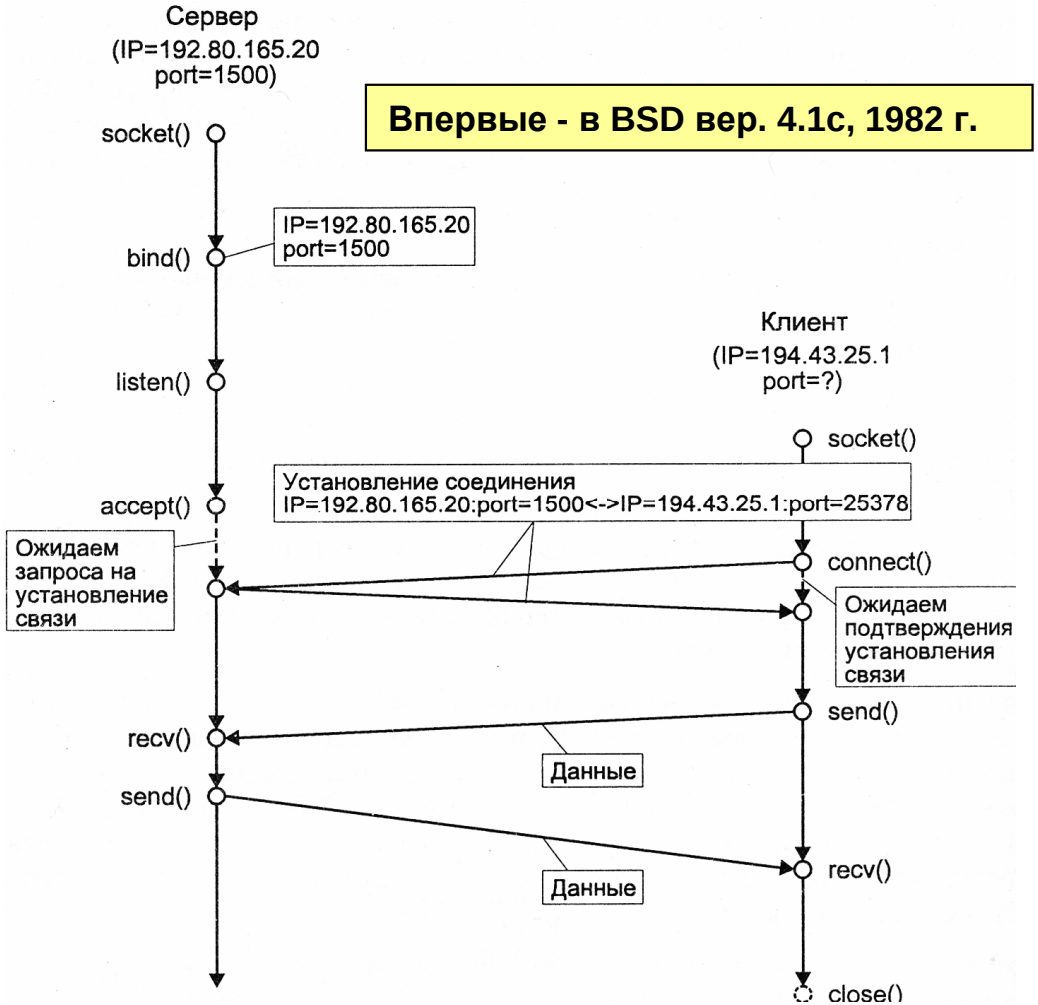


Сокет - конечная точка сетевых коммуникаций

-имеет тип (raw, dg, stream, packet) и ассоциированный с ним процесс.

- существуют внутри коммуникационных доменов: *UNIX, Internet*

Программирование с использованием BSD сокетов




```
main()
{
      int s,ns,pid,nport;
      struct sockaddr_in serv_addr,clnt_addr;
      struct hostent *hp;
      char buff[80],hname[80];

      nport=1500;
      nport=htons((u_short)nport);
      s=socket(AF_INET,SOCK_STREAM,0);
      bzero(&serv_addr,sizeof(serv_addr));
      serv_addr.sin_family=AF_INET;
      serv_addr.sin_addr.s_addr=INADDR_ANY;
      serv_addr.sin_port=nport;
      bind(s,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
      printf("Server is OK!\n");
      listen(s,5);
while(1)
      {
            int addrlen;
            addrlen=sizeof(clnt_addr);
            bzero(&clnt_addr,sizeof(clnt_addr));
            ns=accept(s,(struct sockaddr *)&clnt_addr,&addrlen);
            pid=fork();
            if (pid==0)
            {
                  int nbytes,font;
                   while (nbytes=(recv(ns,buff,sizeof(buff),0))!=0)
                   {
                         send(ns,buff,nbytes,0);
                   }
                   close(ns);
                   exit(0); }
            close(ns);
} }
```

Сервер BSD Sockets

```
#include<sys/socket.h>
#include<sys/types.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<stdio.h>
#include<fcntl.h>
#include<netdb.h>

#define PORT 1500

main()
{
        int s,ns,pid,nport;
        struct sockaddr_in serv_addr,clnt_addr;
        struct hostent *hp;
        char buff[80],hname[80];

        nport=PORT;
        nport=htons((u_short)nport);
        s=socket(AF_INET,SOCK_STREAM,0);
        bzero(&serv_addr,sizeof(serv_addr));
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr=INADDR_ANY;
        serv_addr.sin_port=nport;
        bind(s,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
        printf("Server is OK!\n");
        listen(s,5);
        while(1)
        {
                int addrlen;
                addrlen=sizeof(clnt_addr);
                bzero(&clnt_addr,sizeof(clnt_addr));
                ns=accept(s,(struct sockaddr *)&clnt_addr,&addrlen);
                pid=fork();
                if (pid==0)
                {
                        int nbytes,font;
                        while (nbytes=(recv(ns,buff,sizeof(buff),0))!=0)
                        {
                                send(ns,buff,nbytes,0);
                        }
                        close(ns);
                        exit(0);
                }
                close(ns);
        }
}
```

Утилиты:

netstat

telnet hostname port

в код добавлены небольшие преднамеренные ошибки … может быть

Удаленный вызов процедур

- RPC (Remote Procedure Call) — удаленный вызов процедур.
- Если сокеты – своеобразное продолжение файлового интерфейса для сети, то RPC – продолжение локальных процедур для сети.
- Основная идея RPC - скрыть распределенный характер вызова процедур.
- Клиент и сервер являются независимыми от сетевой реализации, а вызовы процедур имеют стандартный интерфейс.
- Впервые RPC был реализован компанией Sun Microsystems в 1984 году в рамках ее продукта NFS (Network File System - сетевая файловая система.

Этапы RPC

1. Программа-клиент производит локальный вызов процедуры, называемой заглушкой (stub). Задача заглушки - принять аргументы, предназначаемые удаленной процедуре, преобразовать их в формат XDR и сформировать сетевой запрос. Упаковка аргументов и создание сетевого запроса называется ***сборкой*** (***marshalling***).

2. Сетевой запрос ***пересылается*** по сети на удаленную систему. При этом могут быть использованы различные транспортные протоколы, например, UDP или TCP , в IP-сетях.

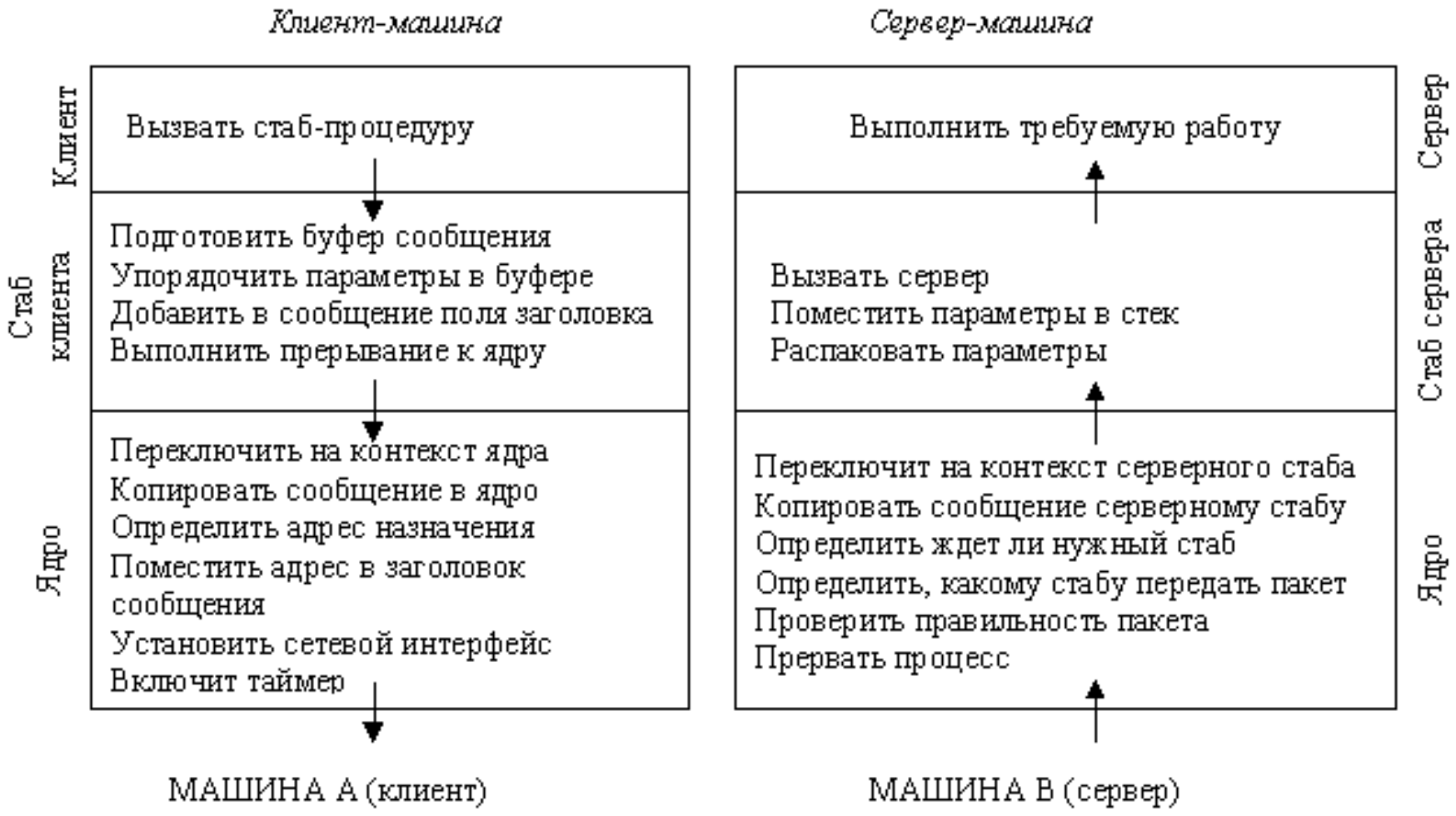
3. На удаленном хосте заглушка сервера ожидает запрос и при получении извлекает параметры - аргументы вызова процедуры. ***Извлечение*** (***unmarshalling***) проводит необходимые преобразования (например, изменения порядка расположения байтов).

4. Заглушка сервера выполняет вызов процедуры-сервера, которой адресован запрос клиента, передавая ей полученные по сети аргументы.

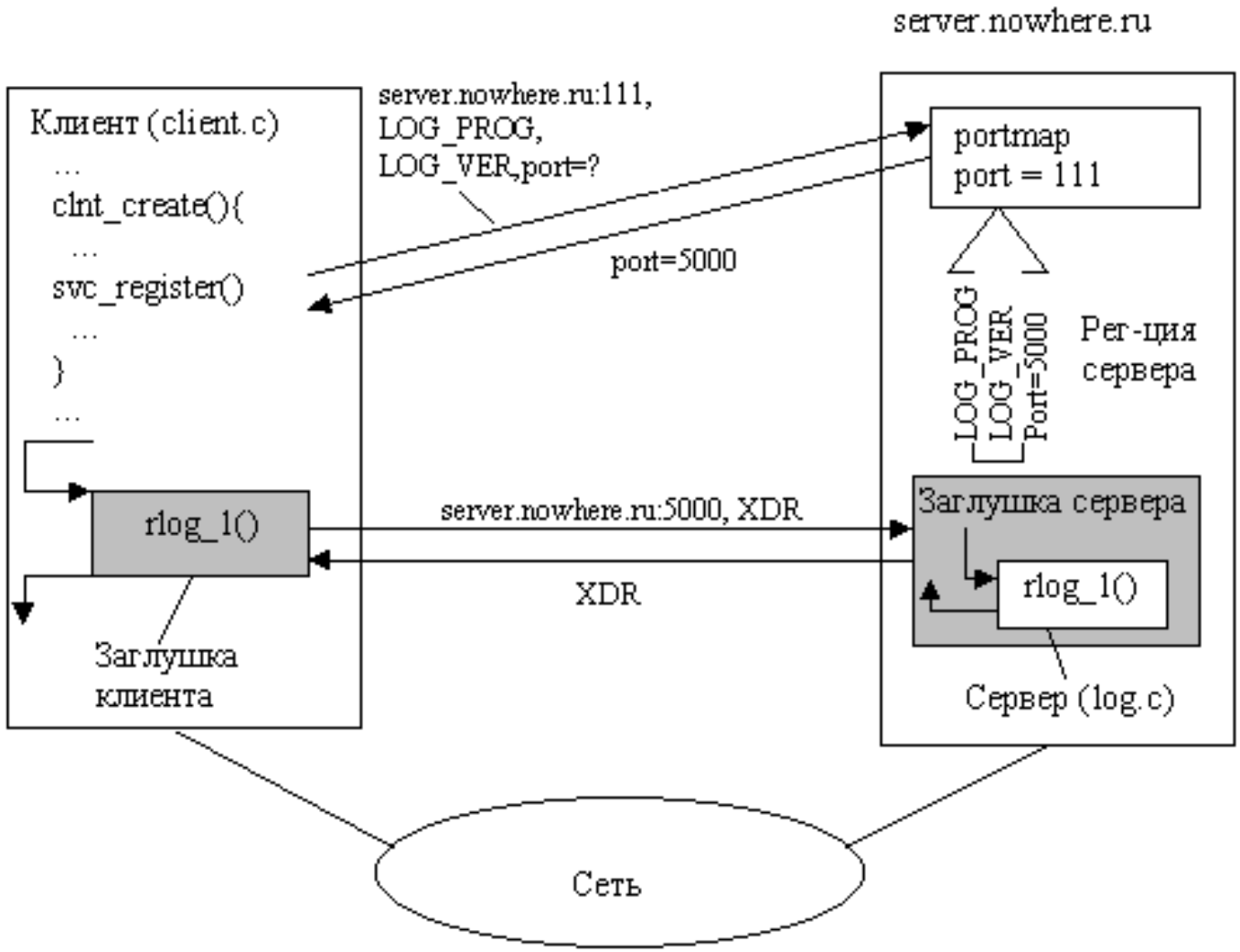
5. После выполнения процедуры управление возвращается в заглушку сервера, передавая ей требуемые параметры. Как и заглушка клиента; заглушка сервера преобразует возвращенные процедурой значения, формируя сетевое сообщение-отклик, который передается по сети системе, от которой пришел запрос.

6. Операционная система передает полученное сообщение заглушке клиента, которая передает значения отклика клиенту, воспринимающему это как обычный возврат из процедуры.

Порядок вызова процедур в RPC



Взаимодействие через UDP/IP



Remote Procedure Call

```
program LOG_PROG{
        version LOG_VER{
                int
RLOG(string)=1;
        }=1;
}=0x12345678;
```

```
main(int argc, char *argv[])
{
        CLIENT *cl;
        char *mystring,*clnttime;
        char *server;
        time_t bintime;

        if (argc!=2)
        {       fprintf(stderr,"Args!\n");
    exit(1);    }
        server=argv[0];

    cl=clnt_create(server,LOG_PROG,LOG_VER,"
    udp");
        mystring=(char*)malloc(100);
        bintime=time((time_t*)NULL);
        clnttime=ctime(&bintime);
        sprintf(mystring,"%s\n",clnttime);
        rlog_1(&mystring,cl);
        clnt_destroy(cl);
        exit(0);
}
```

```
int *rlog_1_svc(char**arg, struct svc_req * req)
{
      static int result;
      int fd,len;
      result=1;
      fd=open("Server.log",O_CREAT|O_RDWR|O_APPEND);
      len=strlen(*arg);
      if
              (write(fd,*arg,strlen(*arg))!=len)
              result=1;
      else
              result=0;
      close(fd);
      return(&result);
}
```

Утилиты:

rpcgen

rpcinfo

в код добавлены небольшие преднамеренные ошибки … может быть.

Реализация RPC

- Реализация:
  - Демон portmapper – сервисный брокер
  - Сами серверы RPC
- Службы, основанные на RPC: NFS, NIS

```
#
# /etc/rpc - miscellaneous RPC-based services
#
portmapper      100000  portmap sunrpc
rstatd          100001  rstat rstat_svc rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
yppasswdd       100009  yppasswd
ypupdated       100028  ypupdate
```

Network Information System (YP, NIS, NIS+, NYS)

- Разработка Sun Microsystems, начальное название Yellow Pages, YP
- Обеспечивает сетевой доступ к базе данных административной информации о пользователе, узле и т.п.
- Состав: сервер, библиотека клиента, утилиты администрирования.

Данные NIS

- Данные хранятся в таблицах, т.н. «maps»:

Master File	Map(s)
/etc/hosts	hosts.byname, hosts.byaddr
/etc/networks	networks.byname, networks.byaddr
/etc/passwd	passwd.byname, passwd.byuid
/etc/group	group.byname, group.bygid
/etc/services	services.byname, services.bynumber
/etc/rpc	rpc.byname, rpc.bynumber
/etc/protocols	protocols.byname, protocols.bynumber
/usr/lib/aliases	mail.aliases

Network File System (NFS)

- Наиболее известная служба, основанная на RPC
- Реализация
  - на стороне сервера – демон(ы) rpc.nfsd (NFS Daemon)
  - таблица разделяемых каталогов - /etc/exports
  - на стороне клиента – демон biod (Block I/O Daemon)

```
# exports file for nix.cs.vsu.ru
/home              nix1(rw) nix2(rw)
/usr/docs          nix3(ro)
/var/spool/mail    nix*.cs.vsu.ru (rw)
```

à mount -t nfs nix.cs.vsu.ru:/home /home/users
Программа mount соединится с демоном rpc.mountd на узле nix.cs.vsu.ru
Передача данных будет происходить с помощью системных вызовов, посылаемых демону rpc.nfsd