СКРИПТЫ

1. Скрипт представляет собой текстовый файл с программой на языке команд *shell*, с атрибутами исполняемого файла, и интерпретируемый оболочкой (т.е. *Shell*).

2. Примеры задания атрибутов исполняемого файла (например, файлу prog1):
$ chmod u+x g+x prog1
$ chmod a+x prog1
$ chmod 755 prog1 – *что аналогично* $chmod u=rwx g=rx o=rx prog1

$ls –il prog1
1083771 -rw-r--r-- 1 user1 users 37 Sep 12 11:02 prog1 – было до *chmod 755 prog1*
1083771 -rwxr-xr-x 1 user1 users 37 Sep 12 11:02 prog1 – стало после *chmod 755 prog1*

3. Вызов файла на исполнение

$ ./prog1 <параметр 1> <параметр 2> <параметр 3> ……

Имя исполняемого файла и путь к нему, представляют собой нулевой параметр командной строки внутри скрипта, который вызывается как $0. Параметры 1 -9 вызываются как переменные $1, $2, … $9. Доступ к остальным параметрам можно получить в цикле for или сдвигом командой shift (см. примеры ниже).

4. Чтобы перед командами или исполняемыми файлами не указывать их местоположение, путь к ним можно вписать в переменную окружения PATH.

Пусть изначально: PATH= /usr/local/bin:/usr/bin:/bin:/usr/local/games
Добавляем путь: $ PATH=$PATH:/home/user1/dir1
Проверяем: $echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/home/usr1/dir1

5. Внутри скрипта можно объявлять и использовать переменные. Также можно использовать переменные, объявленные и заданные вне скрипта, но тогда их надо предварительно экспортировать. Пример:
$vasia=’student of group A-7-08’
$export vasia
$./prog1 ……

СКРИПТ 1
```
echo –n Hello!
echo  My name is $0
echo Count of arguments: $#
echo First argument: $1
```

вызов и результат исполнения
$./prog1 dima vasia
Hello! My name is ./prog1
Count of arguments: 2
First argument: dima

СКРИПТ 2 *(выводит содержимое параметров командной строки)*

Вариант 1
```
for x
do
echo x=$x
done
```

Вариант 2
```
for x
do
echo x=$1
shift
done
```

вызов и результат исполнения
$prog2 dima vasia petia
x=dima
x=vasia
x=petia

СКРИПТ 3 *(присвоение и перебор значений переменной x)*

Вариант 1
```
for x in dima vasia petia
do
echo x=$x
done
```

Вариант 2
```
set dima vasia petia
for x
do
echo x=$x
done
```

СКРИПТ 4 *(вывод содержимого каталога)*

```
N=1
for x in *
do
 echo $N. $x
 N=` expr $N + 1 `
done
```

результат исполнения

1. file1
2. file2.txt

…..

СКРИПТ 5 *(проверка, является ли файл каталогом?)*

```
for x
do
 if test –d $x        # - аналогично if [ -d $x ]
 then echo $x is directory
 else echo $x is not directory
 fi
done
```

вызов скрипта

$./prog5 file[1,2,3]*

СКРИПТ 6 *(проверка, какие файлы текущего каталога являются регулярными, а какие каталогами?)*

```
for x in *
do
 if [ -d $x ]
 then echo $x is directory
 elif [ -f $x ]
  then echo $x is regular
 else echo $x is other
 fi
fi
done
```

СКРИПТ 7 *(search: поиск подстроки в строках из некоторого набора, поступающего на стандартный ввод?)*

примеры вызовов

1. **$ search <искомая подстрока>**
2. **$ search <искомая подстрока> < <имя файла>**
3. **$ ls –l | search <искомая подстрока>**

результат исполнения

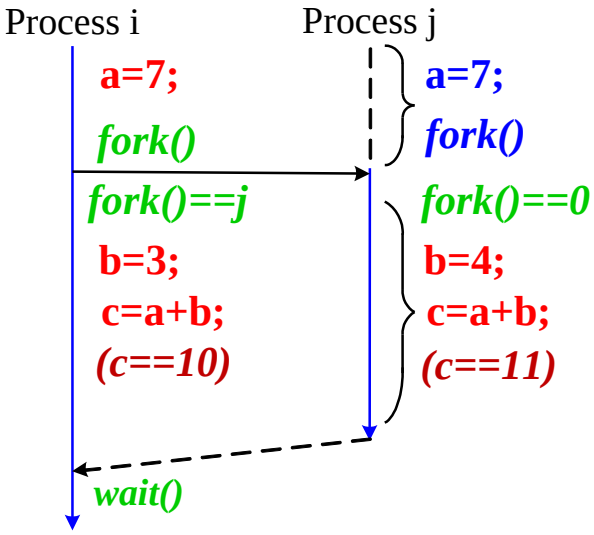
Пусть <искомая подстрока>=open, которая ищется в некотором файле My.txt

```
$ search open < MY.txt
   [3] openCL language
   [14] close and open the window
   Search 2 of 23 strings
```

текст скрипта *search*

```
ln=0   # - число строк где подстрока не была найдена
s=0    # - число строк с найденной подстрокой
while read a
do
 case $a in
  *$1*) s=` expr  $s + 1 `; echo [` expr  $s  +  $ln `] $a ;;
  *) ln=` expr  $ln  + 1 ` ;;
 esac
done
echo Search $s of ` expr $s  +  $ln  `
```

Вариант без замены контекста

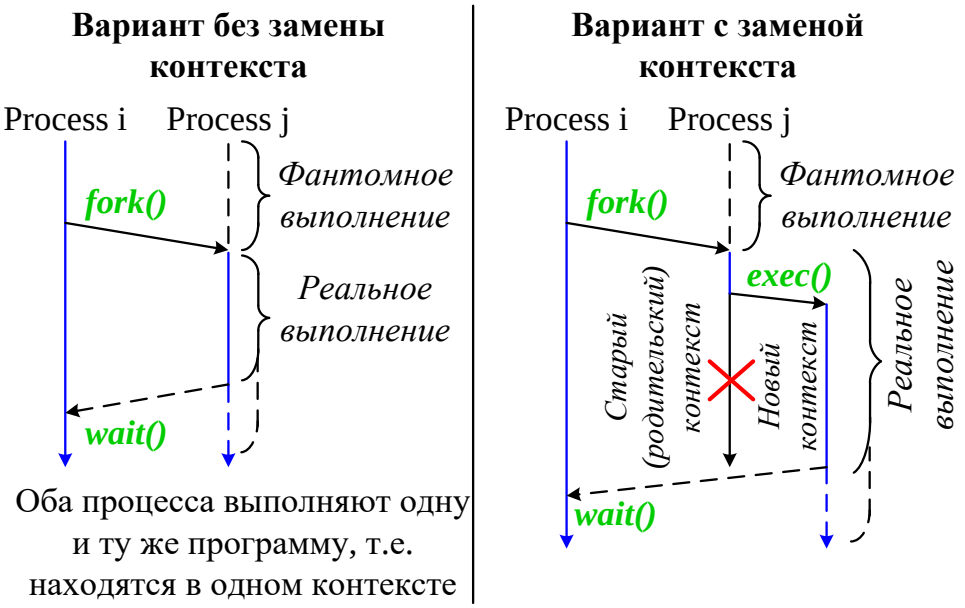


Программа **Multiprocess 1**

```
#include<stdio.h>

int main(int argc, char **argv)
{
int pid=-10,ppid=-10,spid=-10; // main_pid, parent_pid, son_pid
int i,A[4];

pid=getpid(); ppid=getppid();
printf("1. pid=%d, ppid=%d, spid=%d\n",pid,ppid,spid);
for (i=0;i<4; i++) A[i]=i;

spid=fork(); // Вилка – порождение процесса потомка

printf("2. pid=%d, ppid=%d, spid=%d\n",pid,ppid,spid);

pid=getpid(); ppid=getppid();
printf("3. pid=%d, ppid=%d, spid=%d\n",pid,ppid,spid);

A[2]=pid;
printf("4. A[0]=%d, A[1]=%d, A[2]=%d,  A[3]=%d\n",A[0],A[1],A[2],A[3]);
return 0;
}
```

Вывод:

1. pid=4202, ppid=4159, spid=-10
2. pid=4202, ppid=4159, spid=4203
3. pid=4202, ppid=4159, spid=4203
4. A[0]=0, A[1]=1, A[2]=4202, A[3]=3
2. pid=4202, ppid=4159, spid=0
3. pid=4203, ppid=4202, spid=0
4. A[0]=0, A[1]=1, A[2]=4203, A[3]=3



Программа **Multiprocess 2**

```
#include<stdio.h>
//#include<signal.h>
//#define N=4;

int main(int argc, char **argv)
{
int pid=-10,ppid=-10,spid=-10; // main_pid, parent_pid, son_pid

pid=getpid(); ppid=getppid();
printf("1. pid=%d, ppid=%d, spid=%d\n",pid,ppid,spid);


if (spid=fork())   // Если условие выполнилось то это процесс предок
{
pid=getpid(); ppid=getppid();
printf("2. Hello! I'm PARENT, My pid=%d, my parent pid=%d, my son
pid=%d\n",pid,ppid,spid);
}
else // Если условие не выполнилось то это процесс потомок
{
pid=getpid(); ppid=getppid();
printf("2. Hello! I'm SON, My pid=%d, my parent pid=%d, my son
pid=%d\n",pid,ppid,spid);
}

return 0;
}
```

Вывод:

1. pid=4242, ppid=4159, spid=-10
2. Hello! I'm PARENT, My pid=4242, my parent pid=4159, my son pid=4243
2. Hello! I'm SON, My pid=4243, my parent pid=4242, my son pid=0

Программа **Semaphor**

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/types.h>
#include<sys/sem.h>

int main(int argc, char **argv)
{
int sem_id,i;
FILE *f;
float L;
struct sembuf SB;

sem_id=semget(100 | IPC_PRIVATE,1,IPC_CREAT|600);
if (sem_id==-1) {printf("No sem\n"); return 1;}

if (fork())
{
SB.sem_num=0; SB.sem_op=-1; SB.sem_flg=0;
semop(sem_id,&SB,1);

f=fopen("regfile1","w"); L=18.56;
fprintf(f,"%f",L);
fclose(f);

SB.sem_num=0; SB.sem_op=-1; SB.sem_flg=0;
semop(sem_id,&SB,1);

} else
{
SB.sem_num=0; SB.sem_op=2; SB.sem_flg=0;
semop(sem_id,&SB,1);
printf("Ready: ");
SB.sem_num=0; SB.sem_op=0; SB.sem_flg=0;
semop(sem_id,&SB,1);

f=fopen("regfile1","r");
```

```
fscanf(f,"%f",&L);
fclose(f);
printf("Out: %f",L);
}

return 0;
}
```

Программа **Shared memory**

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>

int main()
{
  int shm_id,i,pid,*A;

//  Получение доступа к экземпляру механизма типа Shmem
  shm_id=shmget(100|IPC_PRIVATE,16*sizeof(int),IPC_CREAT|0666);

  pid=fork(); // Порождение потомка

if (pid==0)
{ //Выполняется потомком
      A=shmat(shm_id,0,0); // Подключение памяти к процессу
      for (i=0; i<16; i++) A[i]=i;
      shmdt(A);
  } //Конец текста определенного только для ПОТОМКА
else
{ //Выполняется предком
      A=shmat(shm_id,0,0); // Подключение памяти к процессу
      scanf("%d",&i);
for (i=0; i<16; i++) printf("%d ",A[i]);
      shmdt(A);
  } //Конец текста определенного только для ПРЕДКА

return 0;
   }
```

Пример мини-интерпретатора (калькулятор) Скрипт **Myshell**

```
echo ‘MyShell is running  …’
var0= c
while [ $var0 != exit ]
do
   echo –n ‘>’
   read var0 var1 var2
   case $var0 in
      add) ./fun1.out $var1 $var2 ;;
      sub) ./fun2.out $var1 $var2 ;;
      mul) ./fun3.out $var1 $var2 ;;
      div)  ./fun4.out $var1 $var2 ;;
      exit) echo Good by…… ;;
      *) echo Command not found ;;
   esac
done
```

Пример запуска

```
$ ./Myshell
 MyShell is running …
 > mul 4 3
 12
 > sub 5 2
 3
 > qwerty 5 7
 Command not found
 > add 23 18
 41
 > exit
 Good by……
$
```

Комплекс скриптов **FUNCTION & SIGNAL (***упрощенный вариант***)**

scr1

```
fun()
{
read t <fifo1
echo $t
}

echo pid=$$
mknod fifo1 p
trap fun 15
…
for h in *
do   cat $h > outf   done
```

scr2

```
mknod fifo1 p
while read a; [ $? –eq 0 ]
do
echo $a > fifo1 &
kill -15 $1
done
```

Пример запуска

$ **./scr1 &** – Запуск *scr1* в фоновом режиме
pid=14521
$ **./scr2 14521** – Запуск *scr2* с подстановкой **pid** процесса выполняющего *scr1*
vasia – Ввод строки в *scr2*
vasia – Вывод строки скриптом *scr1*

Программа **PIPE**

```
#include<stdio.h>
int main(int argc, char **argv)
{
int p[2], A[3];
pipe(p);
if (fork()==0)
{
        close(p[1]);
        read(p[0],A,3*sizeof(int));
        printf("A[0]=%d, A[1]=%d, A[2]=%d\n",A[0],A[1],A[2]);  return 0;
}
close(p[0]);
A[0]=12; A[1]=5; A[2]=678;
write(p[1],A,3*sizeof(int));
return 0;
}
```

Программа **CONV**

$./conv <имя исп. Ф1> <имя исп. Ф2>

```
#include<stdio.h>
int main(int argc, char **argv)
{
        int pp[2];
if (argc<3) return(1);

pipe(p);
if (fork()==0)
{
        dup2(p[1],1); close(p[0]);
        execl(argv[1],0);   exit(1);
}
if (fork()==0)
{
        dup2(p[0],0); close(p[1]);
        execl(argv[2],0);   exit(1);
}
wait()
return 0;
}
```

Программа **BREAK_POINT**

```
#include<stdio.h>
#define Ns 1024
#define Ne 1024
int STEP,ELEM,pid,i;
float A[Ne];
FILE *f;
```

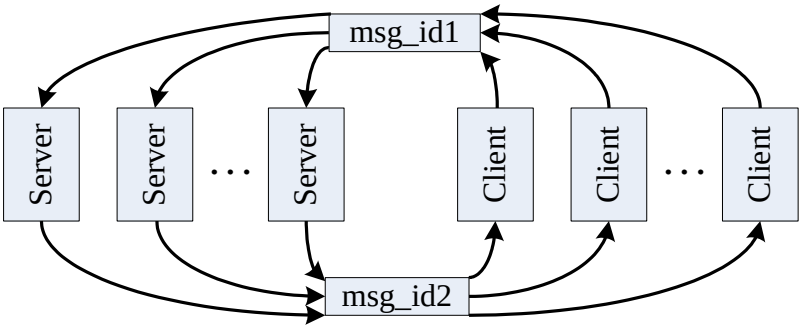
```
int funsig15() // Функция – обработчик 15-го сигнала
{
    fprintf(stderr,"Abort process %d ...",getpid());
    f=fopen("break.pnt","w");
    fprintf(f,"%d %d\n",STEP,ELEM);
    for (i=0;i<Ne;i++)fprintf(f,"%f ",A[i]);
    fclose(f); fprintf(stderr,"BREAK.PNT file save.\n");
    exit(1);
}
int main(int argc, char **argv)
{   int i;
    if (argc!=2) {fprintf(stderr,"Error of arguments\n"); return 2;}

    pid=fork(); // Порождение процесса потомка
    if (pid==0)
    { //Выполняется потомком
    if (!strcmp(argv[1],"new"))  //
    { f=fopen("data.dat","r"); STEP=0; ELEM=0;}
    else
        if (!strcmp(argv[1],"continue"))
        {
            f=fopen("break.pnt","r");
            fscanf(f,"%d %d",&STEP,&ELEM); STEP++;
              for (i=0;i<Ne;i++) fscanf(f,"%f ",A[i]);
            for (ELEM=ELEM;ELEM<Ne;ELEM++)
                 A[ELEM]=function(STEP-1,ELEM,A[ELEM]);
        }
    else {fprintf(stderr,"Corrupted argument \"%s\" \n",argv[1]); return 2;}

    if (STEP==0) for (i=0;i<Ne;i++) fscanf(f,"%f ",&A[i]);
    fclose(f);
    signal(15,funsig15); // Назначение функции обработки сигнала
    for (STEP=STEP;STEP<Ns;STEP++)
        for (ELEM=0;ELEM<Ne;ELEM++)
             A[ELEM]=function(STEP,ELEM,A[ELEM]);
    for (i=0;i<Ne;i++) printf("%f ",A[i]); // Вывод результатов
    } //Конец текста определенного только для ПОТОМКА

    else
    { //Выполняется предком
        …
        kill(pid,15); //Посылка сигнала процессу-потомку
        …
    } //Конец текста определенного только для ПРЕДКА

return 0;  }
```

Предача сообщений «по запросу» програма SEND-RECV

```
#include<stdio.h>
#include<sys/types.h>
#include<signal.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>

int i,j,k,Q_id,pid;
float Param1;
struct send{long mtype; int si,sj,sk; float sP;}snd;

void s_fun()
{
        snd.si=i; snd.sj=j; snd.sk=k; snd.sP=Param1; snd.mtype=1;
        msgsnd(Q_id,&snd,3*sizeof(int)+sizeof(float),0);
}

int main()
{
        Q_id=msgget(1|IPC_PRIVATE,IPC_CREAT|0600);

        if (pid=fork())
        {
                signal(SIGUSR1,s_fun);
                Param1=0.0;
                for (i=0;i<4096;i++)
                        for (j=0;j<4096;j++)
                                for (k=0;k<4096;k++)
                                        Param1= . . . ;
                printf("END Param=%f\n",Param1); waitpid(pid);
        } else
        {       . . .

                kill(getppid(),SIGUSR1);
                msgrcv(Q_id,&snd,3*sizeof(int)+sizeof(float),0,0);
                printf("RECV i=%d; j=%d; k=%d; P=%f\n",snd.si,snd.sj,snd.sk,snd.sP);

                . . .

                kill(getppid(),SIGUSR1);
                msgrcv(Q_id,&snd,3*sizeof(int)+sizeof(float),0,0);
                printf("RECV i=%d; j=%d; k=%d; P=%f\n",snd.si,snd.sj,snd.sk,snd.sP);

                . . .
        } return 0; }
```

Комплекс CLIENT-SERVER

Программа **SERVER**




```
#include<stdio.h>
#include<sys/ipc.h>
#include<fcntl.h>
#include<sys/msg.h>
#include<signal.h>

int main(int argc,char **argv)
{
int rank=0,reserv=0,pidP=0,pidN=0,msg_id1,msg_id2,f,i,SF;
struct ms1{long type; char file[128]; char client;}msgI;
struct ms2{long type; char data[256]; char flag; int size;}msgO;

msg_id1=msgget(100,IPC_CREAT|0600);
msg_id2=msgget(101,IPC_CREAT|0600);

if (argc>1) reserv=atoi(argv[1]); else exit(1);
if (reserv<=0) exit(1);

while (rank<reserv-1 && pidN==0)
{pidN=fork(); if (pidN==0) {rank++; pidP=getppid();}}

for (;;)
{
msgrcv(msg_id1,&msgI,129,1,0);

f=open(msgI.file,O_RDONLY,0);
SF=lseek(f,0,SEEK_END);
lseek(f,0,SEEK_SET);
msgO.type=msgI.client;
msgO.flag=0;

for (i=0;i<SF;i+=256)
{
msgO.size=read(f,msgO.data,256);
msgsnd(msg_id2,&msgO,261,0);
}
msgO.flag=1;
msgsnd(msg_id2,&msgO,261,0);
close(f);
}
return 0;
}
```

Программа **CLIENT**

```
#include<stdio.h>
#include<sys/shm.h>

int main(int argc, char **argv)
{
int SR=0,msg_id1,msg_id2,i;
struct ms1{long type; char file[128]; char client;}msgI;
struct ms2{long type; char data[256]; char flag; int size;}msgO;

for (i=0; i<256; i++) SN[i]=0;
msg_id1=msgget(100,IPC_CREAT |0600);
msg_id2=msgget(101,IPC_CREAT |0600);

msgI.type=1;  msgI.client=. . . . . .;

strcpy(msgI.file,argv[1]);
msgsnd(msg_id1,&msgI,129,0);

msgrcv(msg_id2,&msgO,261,msgI.client,0);
while (msgO.flag==0)
{
. . . . . .
msgrcv(msg_id2,&msgO,261,msgI.client,0);
}
return 0;
}
```

Программа **PIPELINE**

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/sem.h>
#define N 1800
#define M 1500

int main(int argc, char **argv)
{
        int shm_id,sem_id,i,j,k,*A;
        struct sembuf SB;

        shm_id=shmget(100|IPC_PRIVATE,N*sizeof(int),IPC_CREAT|0600);
        sem_id=semget(100|IPC_PRIVATE,1,IPC_CREAT|0600);

        if (shm_id<=0||sem_id<=0)  {      printf("Error=%d %d\n",shm_id,sem_id); return 1; }
```

```
        A=shmat(shm_id,0,0);

    if (fork()==0)
    {
        for (i=0; i<M; i++)
        {
            for (j=0; j<M; j++) A[i..k..j]=f(.....);
            SB.sem_num=0; SB.sem_op=1; SB.sem_flg=0;
            semop(sem_id,&SB,1);
        }
    } else
    {
        for (i=0; i<(int)M/5; i++)
        {
            SB.sem_num=0; SB.sem_op=-5; SB.sem_flg=0;
            semop(sem_id,&SB,1);
            for (k=0; k<5; k++)
                for (j=0; j<M; j++) A[(5*i+k)*16+j]*=2;
        }
    }
    shmdt(A);
}
```

Программа **FIFO - файл**

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<fcntl.h>
#include<signal.h>

int f,c_ms[100],i,myp;

void fun()   { . . .     read(f,&c_ms[i], sizeof(int));      . . . }

int main()
{
    f=open("/home/spec/server.pid",O_WRONLY | O_CREAT,0600);
    myp=getpid();  write(f,&myp, sizeof(int)); close(f);
    mkfifo("/home/spec/fifo.can",S_IRWXU);
    f=open("/home/spec/fifo.can",O_RDONLY,0);
    signal(SIGUSR1,fun);
    . . .
    pause();
    . . .
    close(f);
    return 0;
}
```

UNIX SOCKET **SERVER**

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
int main()
{
        int sock1,sock2,sock3,size,buf[3];
        struct sockaddrX{ ushort sa_family; char sa_data[6];} SA;
        sock1=socket(AF_UNIX, SOCK_STREAM,0);
        SA.sa_family=AF_UNIX;
        SA.sa_data[0]='S'; SA.sa_data[1]='o'; SA.sa_data[2]='c'; SA.sa_data[3]='k';
SA.sa_data[4]='X'; SA.sa_data[5]='X';
        bind(sock1,&SA,sizeof(SA));
        listen(sock1,5);
        size=sizeof(SA);
        sock2=accept(sock1,&SA,&size);
        sock3=accept(sock1,&SA,&size);
        recv(sock2,buf,sizeof(buf),0);
        . . .
        recv(sock3,buf,sizeof(buf),0);
        . . .
        send(sock2,buf,sizeof(buf),0);
        send(sock3,buf,sizeof(buf),0);
        shutdown(sock1,2);

return 0;
}
```

UNIX SOCKET **CLIENTs**

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
int main()
{
        int sock1,buf[3];
        struct sockaddrX{ ushort sa_family; char sa_data[6];} SA;
        sock1=socket(AF_UNIX,SOCK_STREAM,0);
        SA.sa_family=AF_UNIX;
        SA.sa_data[0]='S'; SA.sa_data[1]='o'; SA.sa_data[2]='c'; SA.sa_data[3]='k';
SA.sa_data[4]='X'; SA.sa_data[5]='X';
        connect(sock1,&SA,sizeof(SA));
        . . .
        send(sock1,buf,sizeof(buf),0);
        . . .
        recv(sock1,buf,sizeof(buf),0);
        . . .
        shutdown(sock1,2);      return 0; }
```

INET SOCKET **SERVER**

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main()
{
   int sock, listener;
   struct sockaddr_in addr;
   char buf[1024];
   int bytes_read;

   listener = socket(AF_INET, SOCK_STREAM, 0);
   if(listener < 0) exit(1);

   addr.sin_family = AF_INET;
   addr.sin_port = htons(3425);

        inet_aton("10.4.129.38", &addr.sin_addr);

   if(bind(listener, (struct sockaddr *)&addr, sizeof(addr)) < 0) exit(2);

   listen(listener, 1);

   while(1)
   {
      sock = accept(listener, NULL, NULL);   if(sock < 0)    exit(3);

      while(1)
      {
         bytes_read = recv(sock, buf, 1024, 0);
               .....
         if(bytes_read <= 0) break;
                     buf=……;
         send(sock, buf, bytes_read, 0);
      }

      close(sock);
   }

   return 0;
}
```

ПРИМЕР ФРАГМЕНТОВ ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ УДАЛЁННОГО ВЫПОЛНЕНИЯ В СЕТИ ПРОГРАММ ПОЛЬЗОВАТЕЛЕЙ.

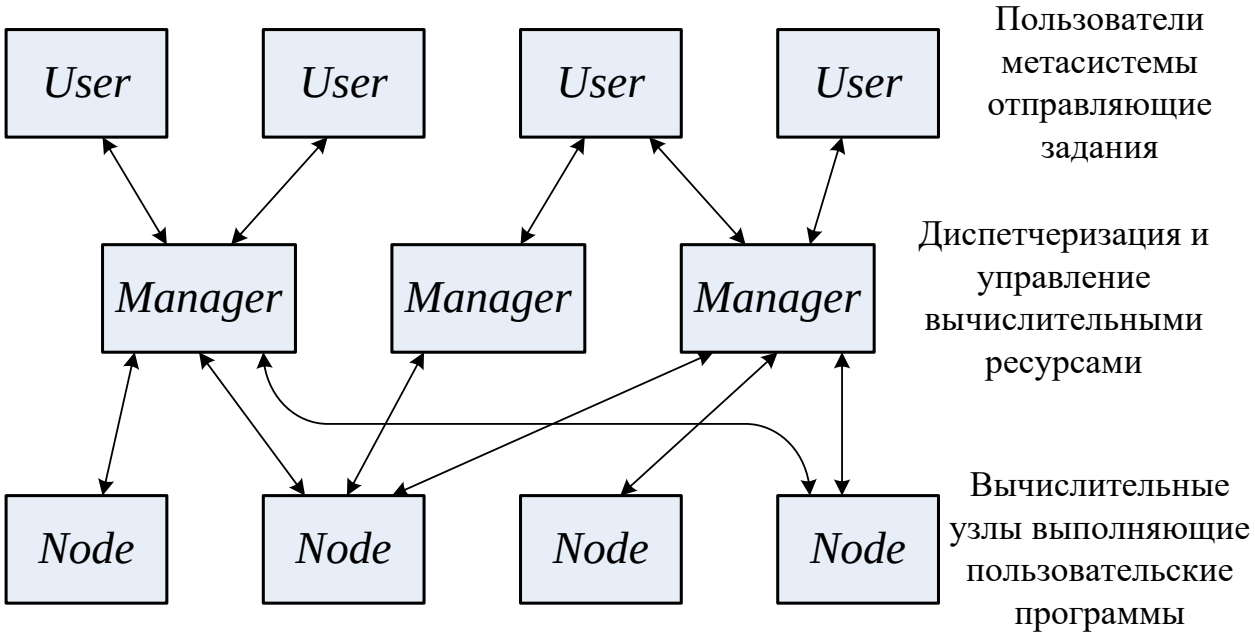


Рис.1. Общая (*без имеющихся упрощений*) схема программного комплекса

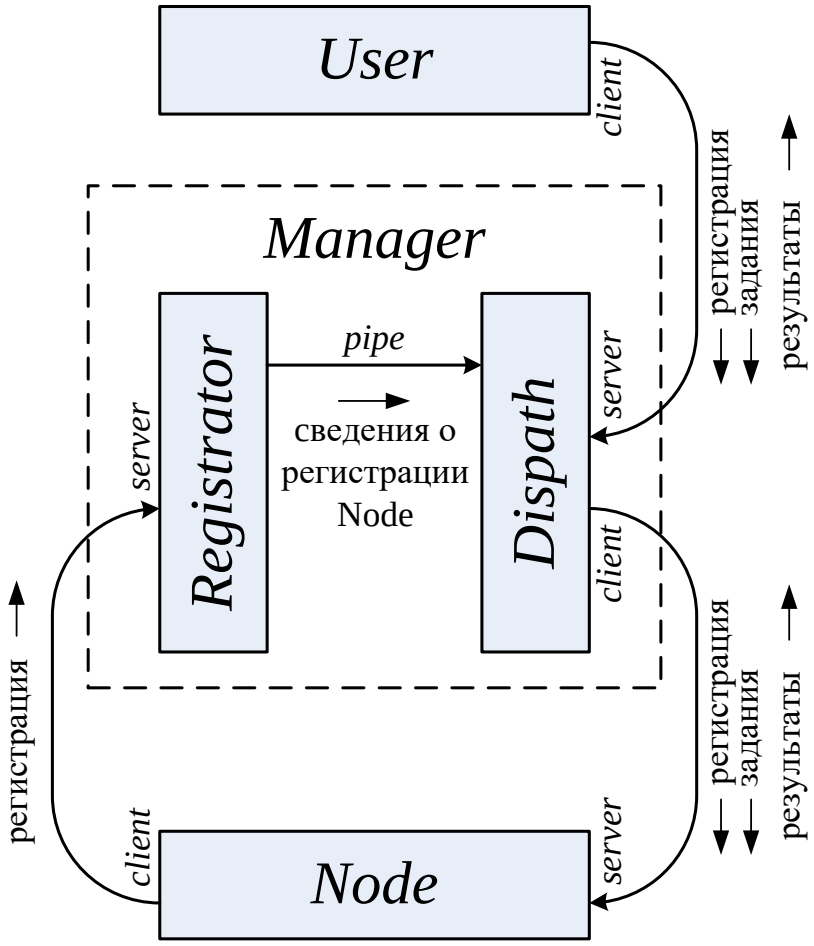


Рис.2. Схема взаимодействия процессов с помощью сетевых сокетов

Программа **Node**

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<fcntl.h>
#include<sys/msg.h>
#include<signal.h>
#include<sys/socket.h>

int main(int argc, char **argv)
{
int sockO,sockI,sockC[4],sSAI,sSAO,f,size,cldpid,i;
struct sockaddr_in SAO,SAI;
struct registr{char num,flag;}reg;
struct tasks{char flag,task_n[16],task_a[16],task_d[16];}task;
char data_byte[513]; //data_byte[0] - flag; data_byte[1-512] - datas;

SAI.sin_family = AF_INET;
SAI.sin_port = htons(3427);
inet_aton("192.168.1.15", &SAI.sin_addr);
sSAI=sizeof(SAI);
SAO.sin_family = AF_INET;
SAO.sin_port = htons(3425);
inet_aton("192.168.1.X", &SAO.sin_addr);
sSAO=sizeof(SAO);

sockO=socket(AF_INET,SOCK_STREAM,0);
sockI=socket(AF_INET,SOCK_STREAM,0);
bind(sockI,&SAI,sSAI);
listen(sockI,4);
reg.num=XXX; reg.flag=1;
if (connect(sockO,&SAO,sSAO)==-1) {printf("EXIT_1\n"); exit(1);}
send(sockO,&reg,sizeof(reg),0);
recv(sockO,&reg,sizeof(reg),0);
if (reg.flag!=2) exit(1);

sockC[0]=accept(sockI,&SAI,&sSAI);
if (sockC[0]==-1) {printf("EXIT_2\n"); exit(2);}

recv(sockC[0],&reg,sizeof(reg),0);
if (reg.flag==100){reg.flag=101; send(sockC[0],&reg,sizeof(reg),0);}
else exit (3);

for(;;)
{
```

```
    recv(sockC[0],&task,sizeof(task),0);

    if (task.flag==10)
    {
    f=open(task.task_n,O_WRONLY|O_CREAT,0700);

    size=recv(sockC[0],data_byte,513,0);
    write(f,&data_byte[1],size-1);
    while (data_byte[0]==11 && size==513)
    {
        size=recv(sockC[0],data_byte,513,0);
        write(f,&data_byte[1],size-1);
    }
    close(f);

    cldpid=fork();
    if (cldpid==0){execl(task.task_n,XXX);}
    else
    {
    wait();
    f=open("result",O_RDONLY,0600);
    data_byte[0]=30;
    size=read(f,&data_byte[1],512);
    send(sockC[0],data_byte,size+1,0);
    close(f);
    }
    }
}
return 0;
}
```

Программа **Manager**

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<fcntl.h>
#include<signal.h>
#include<sys/socket.h>

int size,sizef,sizeb,nodesI=0,nodesE=0,sSA,rank,pid,sock[128],sockU[2],fn,fifo[2],f,init=0;
struct sockaddr_in SA;
char num,data_byte[513]; //data_byte[0] - flag; data_byte[1-512] - datas;
struct registr{char num,flag;}reg;
struct tasks{char flag,task_n[16],task_a[16],task_d[16];}task;
```

```
int fun(){
    printf("signal\n");
    read(fifo[0],&num,1);
    sock[num]=socket(AF_INET, SOCK_STREAM, 0);
    if (sock[num]==-1) {printf("Signal_EXIT_1\n"); return 1;}
    SA.sin_family = AF_INET;
    SA.sin_port = htons(3425);
    inet_aton(f_ip(num), &SA.sin_addr); // f_ip(num)="192.168.1.X"
    sSA=sizeof(SA);
    if (connect(sock[num],&SA,sSA)==-1) {printf("Signal_EXIT_2\n"); return 1;}
    reg.flag=100; reg.num=1;
    send(sock[num],&reg,sizeof(reg),0);
    recv(sock[num],&reg,sizeof(reg),0);
    if (reg.flag==101) init=1;
    return 0;
}

int main(int argc, char **argv)
{
// if (argc!=2) exit(1);
// size=atoi(argv[1]);
// if (size<1 || size>4) exit(2);
// for (rank=0;rank<size-1; rank++) {if (fork()==0) break;}
pipe(fifo);
fn=open("NODES",O_RDWR|O_CREAT,0600); pid=fork();
if (pid==-1) exit(3);

if (pid!=0) //Daspath
{
      close(fifo[1]);
      sigset(SIGUSR1,fun); do pause(); while (init!=1);

      sockU[0]=socket(AF_INET,SOCK_STREAM,0);
      SA.sin_family = AF_INET;
      SA.sin_port = htons(3426);
      inet_aton("192.168.1.15", &SA.sin_addr);
      sSA=sizeof(SA);

      for (;;)
      {
      sockU[1]=accept(sockU[0],&SA,&sSA);
      recv(sockU[1],&reg,sizeof(reg),0);

      recv(sockU[1],&task,sizeof(task),0);
```

```
        // task.flag=10; strcpy(task.task_n,"tsk.e"); strcpy(task.task_a,"2");
        // strcpy(task.task_d,"data.dta");
        send(sock[num],&task,sizeof(task),0);

        size=recv(sockU[1],data_byte,513,0);
        send(sock[num],data_byte,size,0);

        while (data_byte[0]==11 && size==513)
        {
        size=recv(sockU[1],data_byte,513,0);
        send(sock[num],data_byte,size,0);
        }
        close(f);

        size=recv(sock[num],data_byte,513,0);
        if (data_byte[0]==30) send(sockU[1],data_byte,size,0);
        }
}  // END Dispath

else //Registrator
{
        close(fifo[0]);
        sock[0]=socket(AF_INET,SOCK_STREAM,0);
        SA.sin_family = AF_INET;
        SA.sin_port = htons(3427);
        inet_aton("192.168.1.15", &SA.sin_addr);
        sSA=sizeof(SA);
        bind(sock[0],&SA,sSA);
        listen(sock[0],4);  // Упрощенный вариант подключения без порождения спец. процессов

        for(;;)
        {
        sock[nodesI+1]=accept(sock[0],&SA,&sSA);
        if (sock[nodesI]==0) exit(4);
        nodesI++;
        recv(sock[nodesI],&reg,sizeof(reg),0);

        if (reg.flag==1) {lseek(fn,reg.num,SEEK_SET); write(fn,&reg.flag,1); reg.flag=2;
write(fifo[1],&reg.num,1); kill(getppid(),SIGUSR1);}
        send(sock[nodesI],&reg,sizeof(reg),0);
        sleep(10);
        }

}
close(fn);
return 0; }
```

Датаграмные сокеты (посылающий процесс)

```
#include<sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

char msg1[] = "Hello there!\n";
char msg2[] = "Bye bye!\n";

int main()
{
int sock;
struct sockaddr_in addr;

sock = socket(AF_INET, SOCK_DGRAM, 0);
if(sock < 0)
   {
perror("socket");
exit(1);
   }

addr.sin_family = AF_INET;
addr.sin_port = htons(3425);
addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
sendto(sock, msg1, sizeof(msg1), 0, (structsockaddr *)&addr, sizeof(addr));

connect(sock, (structsockaddr *)&addr, sizeof(addr));
send(sock, msg2, sizeof(msg2), 0);

close(sock);

return 0;
}
```

Датаграмные сокеты (принимающий процесс)

```
#include<sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>

int main()
{
int sock;
struct  sockaddr_in  addr;
char buf[1024];
```

```
int bytes_read;

sock = socket(AF_INET, SOCK_DGRAM, 0);
if(sock < 0)
   {
perror("socket");
exit(1);
   }

addr.sin_family = AF_INET;
addr.sin_port = htons(3425);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
if(bind(sock, (structsockaddr *)&addr, sizeof(addr))< 0)
   {
perror("bind");
exit(2);
   }

while(1)
   {
bytes_read = recvfrom(sock, buf, 1024, 0, NULL, NULL);
buf[bytes_read] = '\0';
printf(buf);
   }

return 0;
}
```

Посылающий процесс с низкоуровневым сокетом

Ethernet - *заголовок*	***IP*** - *заголовок*	***UDP*** - *заголовок*	**Данные**

Рис. 1. Формат пакета UDP-протокола

Порт отправителя (16 бит)	Порт получателя (16 бит)
Длина (заголовок + данные) (16 бит)	Контрольная сумма (16 бит)

Рис. 2. Формат заголовкаUDP-протокола

```
#include<sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

Struct  UdpHeader
{
   u_short  src_port;    // Порт отправителя
   u_short  targ_port;  // Порт получателя
   u_short  length;       // Длина
   u_short  checksum;  // Контрольная сумма
};

char message[] = "Hello there!\n";
char msgbuf[1024];

int main()
{
int sock;
struct  sockaddr_in  addr;
struct UdpHeader header;

sock = socket(AF_INET, SOCK_RAW, IPPROTO_UDP);
if(sock < 0)
   {
perror("socket");
exit(1);
   }

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);

header.targ_port = htons(3425);
header.length = htons(sizeof(header)+sizeof(message));
header.checksum = 0;

memcpy((void *)msgbuf, (void *)&header, sizeof(header));
memcpy((void *)(msgbuf+sizeof(header)), (void *)message, sizeof(message));

sendto(sock, msgbuf, sizeof(header)+sizeof(message), 0,
        (structsockaddr *)&addr, sizeof(addr));

close(sock);
return 0;
}
```

Программа сервера для параллельного обслуживания запросов от клиентов

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main()
{
int sock, listener, bytes_read;
struct  sockaddr_in  addr;
char  buf[1024];

listener = socket(AF_INET, SOCK_STREAM, 0);

addr.sin_family = AF_INET;
addr.sin_port = htons(3425);
addr.sin_addr.s_addr = INADDR_ANY;
bind(listener, (structsockaddr *)&addr, sizeof(addr))

listen(listener, 1);

while(1)
   {
sock = accept(listener, NULL, NULL);

switch(fork())
      {
case -1:perror("fork");break;
case 0:
close(listener);
while(1)
         {
bytes_read = recv(sock, buf, 1024, 0);
if(bytes_read<= 0) break;
send(sock, buf, bytes_read, 0);
         }
close(sock); exit(0);

default:close(sock);
      }
   }

close(listener);
return 0;}
```