**Операции ввода-вывода**

**open** — открывает или создает файл

#include <sys/stat.h>

#include <fcntl.h>

int open(

const char path, /\*полное имя файла\*/

int flags, /\*флаги\*/

mode\_t perms /\*права доступа(если создается новый)\*/

);

/\*возврвщает дескр. файла, или -1 в случае ошибки\*/

O\_RDONLY - только чтение

O\_WRONLY - только запись

O\_RDWR - чтение и запись

O\_APPEND - открыть для добавления в конец файла

O\_CREAT - если файла нет, он создастся(тогда следует указать права доступа)

O\_TRUNC - удалить содержимое файла

O\_EXCL - ошибка при попытке открыть существующий файл(в комбинации с O\_CREAT)

**creat** - создает новый или очищает существующий файл и открывает его на запись

#include <sys/stat.h>

#include <fcntl.h>

int creat(

const char path, /\*полное имя файла\*/

mode\_t perms /\*права доступа\*/

);

/\*возврвщает дескр. файла, или -1 в случае ошибки

**write** - выполняет запись в файловый дескриптор

#include <unistd.h>

ssize\_t write (int fd, const void \*buf, size\_t nbytes);

/\*возвращает колличество записанных байт или -1 в случае ошибки\*/

**read** — запись в файловый дескриптор

#include <unistd.h>

ssize\_t read (int fd, const void \*buf, size\_t nbytes);

/\*возвращает колличество прочитанных байт или -1 в случае ошибки\*/

**lseek** - устанавливает и возвращает текущую позицию в файле

#include <unistd.h>

off\_t lseek(

int fd,

off\_t pos,

int whence /\*интерпретация аргумента pos\*/

);

/\*возвращает новую позицию в файле или -1 в случае ошибки\*/

SEEK\_SET - pos содержит смещение от начала файла

SEEK\_CUR - pos содержит смещение от текущей позиции в файле

SEEK\_END - pos содержит смещение от конца файла

lseek(fd,pos,SEEK\_SET) - установка абсолютной позиции в файле

lseek(fd,0,SEEK\_END) - перемещение в конец файла

lseek(fd,0,SEEK\_CUR) - получение текущей позиции

**struct stat** - структура для приема сведений из stat, lstat, fstat

struct stat{

...

inc\_t st\_inc; /\*номер индексного узла\*/

mode\_t st\_mode; /\*режим\*/

nlink\_t st\_nlink; /\*кол-во жестких ссылок\*/

cff\_t st\_size; /\*размер в байтах\*/

time\_t st\_atime; /\*время последнего обращения\*/

time\_t st\_mtime; /\*время последнего изменения\*/

}

**st\_mode** - макросы для проверки соответствия заданному типу

S\_ISDIR /\*каталог\*/

S\_ISFIFO /\*именованный или неименованный канал\*/

S\_ISREG /\*обычный файл\*/

S\_ISSOCK /\*сокет\*/

использование: if (S\_ISSOCK(buf.st\_mode))

**stat** - возвращает сведения о файле по его имени

#include <sys/stat.h>

int stat(

const char \*path, /\*полное имя файла\*/

struct stat \*buf /\*возвращаемая информация\*/

);

/\*возвращает 0 - в случае успеха, -1 - в случае ошибки\*/

**lstat** - возвращает сведения о файле по его имени без разыменования символических ссылок

#include <sys/stat.h>

int lstat(

const char \*path, /\*полное имя файла\*/

struct stat \*buf /\*возвращаемая информация\*/

);

/\*возвращает 0 - в случае успеха, -1 — в случае ошибки\*/

**fstat** - возвращает сведения о файле по его дескриптору

#include <sys/stat.h>

int fstat(

const int fd, /\*полное имя файла\*/

struct stat \*buf /\*возвращаемая информация\*/

);

/\*возвращает 0 - в случае успеха, -1 — в случае ошибки\*/

**каталоги**

**opendir** - открывает каталог

#include <dirent.h>

DIR \*opendir(const char \*path);

/\*возвращает указатель на DIR или NULL в случае ошибки\*/

**closedir** - закрывает каталог

#include <dirent.h>

int closedir(DIR \*dir);

/\*0 - в случае успеха, -1 - в случае ошибки\*/

**readdir** - читает запись из каталога

#include <dirent.h>

struct dirent \*readdir(DIR \*dirp);

/\*возвращает указатель на структуру или NULL по достижении конца каталога или в случае ошибки\*/

**struct dirent** - структура для размещения одной записи вызовом readdir

struct dirent{

inc\_t d\_inc; /\* номер индексного узла \*/

char d\_name[]; /\* имя \*/

};

**rewinddir** - переход в начало каталога

#include <dirent.h>

void rewinddir(DIR \*dirp);

**mkdir** - создает каталог

#include <sys/stat.h>

int mkdir(const char \*path, mode\_t perms);

/\*0 - успех, -1 - неудача\*/

**rmdir** - удаляет каталог

#include <unistd.h>

int rmdir(const char \*path);

/\*то же самое\*/

**еще**

**fcntl** - выполняет управляющие операции над открытым файлом

#include <unistd.h>

#include <fcntl.h>

int fcntl(

int fd, /\*дескрптр\*/

int op, /\*операция\*/

.../\*необязат., зависит от вида операции\*/

);

F\_DUPFD - создать дубликат дескриптора

**Процессы и потоки**

**execl** - запускает программу; входные пар-ры передаются в виде списка

#include <unistd.h>

int execl(

const char \*path, /\*полный путь к программе\*/

const char \*arg0, /\*1й аргумент (arg[0] - имя файла)\*/

const char \*arg1,

...,

NULL

);

/\* -1 в случае ошибки, в случае успеха возврат не предусмотрен, т. к. точка возврата будет утеряна \*/

**execv** - запускает программу; входные пар-ры передаются в виде массива

#include <unistd.h>

int execv(

const char \*path, /\*полный путь к программе\*/

char \*const argv[] /\*массив аргументов\*/

);

/\* -1 в случае ошибки\*/

**fork** - создает новый процесс

#include <unistd.h>

pid\_t fork(void);

/\*возвращает идентификатор дочернего процесса или 0 в случае успеха и -1 в случае ошибки\*/

**exit** - завершает процесс с обращением к коду сборки мусора

#include <stdlib.h>

void exit(int status);

ошибка — exit(EXIT\_FAILURE);

**waitpid** - ожидает изменения состояния дочернего процесса

#include <sys/wait.h>

pid\_t waitpid(

pid\_t pid, /\*идентификатор процесса или группы процессов\*/

int \*statusp, /\*указатель на статус или NULL\*/

int options /\*флаги\*/

);

/\*в случае успеха - идентификатор процесса или 0, в случае ошибки -1\*/

аргумент pid:

>0 ожидать изменения состояния дочернего процесса с указанным идентификатором

-1 ожидать изменения состояния любого дочернего процесса

0 любой дочерн. процесс из той же группы, что и вызывающий

<-1 любой дочерн. принадл. к группе с идетификатором -pid

opt = 0 - ждать завершения

waitpid(pid[i],NULL,0);

**wait** - ожидает завершения дочернего процесса

#include <sys/wait.h>

pid\_t wait(int \*statusp /\*указатель на статус или NULL\*/);

/\*возвр. идентификатор процесса или -1 в случае ошибки\*/

WIFEXITED(statusp) - true, если потомок завершил работу обычным образом

WEXITSTATUS(statusp) - если WIFEXITED вернул true, младшие 8 бит представляют собой аргумент вызова exit

WIFSIGNALED(statusp) - true, если потомок завершился аварийно(по сигналу)

**getpid** - возвращает идентификатор родительского процесса

#include <unistd.h>

pid\_t getpid(void);

/\*возвращает идетификатор процесса\*/

**getppid** - возвращает идентификатор родительского процесса

#include <unistd.h>

pid\_t getppid(void);

/\*возвращает идетификатор родительского процесса\*/

**nice** - изменяет значение пар-ра nice

#include <unistd.h>

int nice(int incr/\*приращение\*/);

/\*возвращает новое значение nice-NZERO или -1 в случае ошибки\*/

не суперпользователи могут только повысить дружелюбность

**Межпроцессорное взаимодействие**

**pipe** - создает канал

#include <unistd.h>

int pipe(int pfd[2]);

/\*0 в случае успеха, -1 в случае ошибки\*/

pfd[0] - чтение, pfd[1] — запись

**dup** - дублирует дескриптор файла

#include <unistd.h>

int dup(int fd /\*дублируемый дескриптор файла\*/);

/\*возвращает новый дескриптор файла или -1 в случае ошибки\*/

~ fcntl(fd, F\_DUPFD, 0)

**dup2** - дублирует дескриптор файла

#include <unistd.h>

int dup2(

int fd, /\*дублируемый\*/

int fd2 /\*используемый\*/

);

/\*возвращает новый дескриптор файла или -1 в случае ошибки\*/

~ close(fd2);

fcntl(fd, F\_DUPFD, fd2);

**IPC**

**очереди сообщений**

**msgget** - возвращает идентификатор очреди сообщений

#include <sys/msg.h>

int msgget(key\_t key, int flags);

/\*возвращает идентификатор очереди сообщений или -1 в случае ошибки\*/

IPC\_CREAT - при необходимости создается новая очередь

IPC\_EXCL - неудача, если лчередь уже существует

ключ - IPC\_PRIVATE - создает совершенно новую очередь

key\_t msg\_key = 1234;

int msgid = msgget(msg\_key,0666|IPC\_CREAT|IPC\_EXCL); - создать очередь с нужным ключом

int msgid = msgget(IPC\_PRIVATE, 0666); - создать совершенно новую очередь

**msgctl** - управляет очередью сообщений

#include <sys/msg.h>

int msgctl(

int msgid, /\*идентификатор\*/

int cmd, /\*команда\*/

struct msqid\_ds \*data /\*пар-ры команды\*/

);

/\*0 - успех, -1 - ошибка\*/

struct msqid\_ds

{

msgqnum\_t msg\_qnum; /\*кол-во сообщений в очереди\*/

msglen\_t msg\_qbytes; /\*макс размер очереди в байтах\*/

pid\_t msg\_lspid; /\*идент-р последнего процесса, вызвавшего msgsnd\*/

pid\_t msg\_lrpid; /\*идент-р последнего процесса, вызвавшего msgrcv\*/

...

};

аргумент cmd:

IPC\_RMID - удаляет очередь. data = NULL

IPC\_STAT - заполняет структуру, на кот-у указывает data

IPC\_SET - ченить поменять

узнать макс. длину:

struct msqid\_ds dat;

if(msgctl(msgid, IPC\_STAT, &dat)==-1) perror("OIOI");

cout << dat.msg\_qbytes << endl;

**struct msg** - типичная структура для работы с сист. вызовами msgsnd, msgrcv

struct msg{

long mtype;

char mtextp[MTEXTSIZE];

};

**msgsnd** - помещает сообщение в очередь

#include <sys/msg.h>

int msgsnd(

int msqid, /\*иднтфктр\*/

const void \*msgp, /\*сообщение\*/

size\_t msgsize, /\*р-р сообщения\*/

int flags /\*флаги\*/

);

/\*0 - успех, -1 - ошибка\*/

**msgrcv** - извлекает сообщения из очереди

#include <sys/msg.h>

ssize\_t msgrcv(

int msqid,

void \*msgp,

size\_t mtextsize, /\*р-р буфера mtext\*/

long msgtype, /\*тип запрошенного сообщения\*/

int flags /\*флаги\*/

);

/\*число байт в mtext или -1 в случае ошибки\*/

аргумент msgtype:

0 - 1е сообщение из очереди

>0 - 1ое сообщение заданного типа

<0 - 1ое сообщение с наименьшим значением типа, которое меньше или равно абсолютному значению msgtype

ставим IPC\_NOWAIT - чтоб никто не блокировался, а возвращал управление в программу

**семафоры**

semid = **semget**(semkey,1,0666|IPC\_CREAT|IPC\_EXCL); //также как с очередями, 2й аргумент - число семафоров в наборе

**semctl** - управляет набором семафоров

#include <sys/sem.h>

int semctl(

int semid,

int semnum,

int cmd,

union semun arg

);

/\*0 - успех, -1 - ошибка\*/

semctl(semid,0,SETVAL,3); - инициализировать начально значение

union semun{

int val; /\* целое число \*/

struct semid\_ds \*buf; /\* указатель на структуру \*/

unsigned short \*array; /\* массив \*/

}; // короче ненужная фигня. не пригождается вроде

**semop** - управляет набором семафоров

#include <sys/sem.h>

int semop(

int semid,

struct sembuf \*sops, /\*операции\*/

size\_t nsops /\*кол-во операций\*/

);

struct sembuf{

unsigned short sem\_num; /\*номер семафора\*/

short sem\_op; /\*операция над семафором\*/

short sem\_flg; /\*флаги операции\*/

};

sem\_op:

>0 sem\_op добавляется к значению семафора

<0 если абс. значение семафора < sem\_op процесс блокируется пока они по крайней мере не сравняются

0 блокируется пока значение семафора не станет = 0

/\*уменьшение на 1\*/

void p(int num,int semid)

{ sem\_buf.sem\_num=num;

sem\_buf.sem\_op=-1;

semop(semid,&sem\_buf,1);

}

/\* увеличение на 1\*/

void v(int num,int semid)

{ sem\_buf.sem\_num=num;

sem\_buf.sem\_op=1;

semop(semid,&sem\_buf,1);

}

**разделяемая память**

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <stdio.h>

#include <iostream>

#include <errno.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/wait.h>

using namespace std;

int semid;

void erro(const char \*mes)

{

perror(mes);

exit(EXIT\_FAILURE);

}

void semset(int num, int op, sembuf sm)

{

sm.sem\_num = num;

sm.sem\_op=op;

semop(semid,&sm,1);

}

int main()

{

int shmid;

int pid[4];

sembuf sem\_buf;

semid = semget(IPC\_PRIVATE, 5, 0666); if (semid == -1) perror("semafors");

shmid = shmget(IPC\_PRIVATE, 100, 0666); if (shmid == -1) perror("memory");

semctl(semid, 0, SETVAL, 0);

semctl(semid, 1, SETVAL, 0);

semctl(semid, 2, SETVAL, 0);

semctl(semid, 3, SETVAL, 1);

semctl(semid, 4, SETVAL, 0);

switch(pid[0] = fork())

{

case -1: erro("process 1"); break;

case 0:

void\* str = shmat(shmid, NULL, 0);

semset(3, -1, sem\_buf);

strcpy((char\*)str, "belii, belii inei \n");

semset(0, 1, sem\_buf);

semset(4, 1, sem\_buf);

shmdt(str);

exit(EXIT\_SUCCESS);

break;

}

switch(pid[1] = fork())

{

case -1: erro("process 1"); break;

case 0:

void\* str = shmat(shmid, NULL, 0);

semset(0, -1, sem\_buf);

strcpy((char\*)str, "leg na provoda \n");

semset(1, 1, sem\_buf);

semset(4, 1, sem\_buf);

shmdt(str);

exit(EXIT\_SUCCESS);

break;

}

switch(pid[2] = fork())

{

case -1: erro("process 1"); break;

case 0:

void\* str = shmat(shmid, NULL, 0);

semset(1, -1, sem\_buf);

strcpy((char\*)str, "v nebe temno-sinem \n");

semset(2, 1, sem\_buf);

semset(4, 1, sem\_buf);

shmdt(str);

exit(EXIT\_SUCCESS);

break;

}

switch(pid[3] = fork())

{

case -1: erro("process 1"); break;

case 0:

void\* str = shmat(shmid, NULL, 0);

semset(2, -1, sem\_buf);

strcpy((char\*)str, "sinya voda \n");

semset(3, 1, sem\_buf);

semset(4, 1, sem\_buf);

shmdt(str);

exit(EXIT\_SUCCESS);

break;

}

//sleep(3);

void \*str = shmat(shmid, NULL, 0);

for (int i=0; i<4; i++)

{

semset(4, -1, sem\_buf);

cout << (char\*)str;

}

shmdt(str);

semctl(semid, IPC\_RMID, NULL);

shmctl(shmid, IPC\_RMID, NULL);

exit(EXIT\_SUCCESS);

}

---------------------------------------------------------------------

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

void semSet(sembuf &x, int num, int op) {

x.sem\_num = num;

x.sem\_op = op;

x.sem\_flg = 0;

}

void start(sembuf data[3], int sem, int i) {

semSet(data[0], 4, 1);

semop(sem, &data[0], 1);

semSet(data[1], i, -1);

semop(sem, &data[1], 1);

semSet(data[2], 4, 0);

semop(sem, &data[2],1);

}

void out(void \* mem, int sem, char \* row, int process) {

sembuf data[1];

semSet(data[0], process, 0);

semop(sem, data, 1);

strcpy((char\*)mem, row);

semSet(data[0], process, 1);

semop(sem, data, 1);

semSet(data[0], 4, -1);

semop(sem, data, 1);

}

int main (int argc, char\*\* argv){

int sem = semget(0, 5, IPC\_CREAT|0666);

semctl(sem, 0, SETVAL, 1);

semctl(sem, 1, SETVAL, 1);

semctl(sem, 2, SETVAL, 1);

semctl(sem, 3, SETVAL, 1);

semctl(sem, 4, SETVAL, 0);

int shmid = shmget(1, 100, IPC\_CREAT|0666);

int p1, p2, p3, p4;

if((p1=fork())==0){

void\* mem = shmat(shmid, 0, 0);

out(mem, sem, "Row 1\n", 0);

out(mem, sem, "Row 3\n", 0);

shmdt(mem);

exit(0);

} else if((p2=fork())==0){

void\* mem = shmat(shmid, 0, 0);

out(mem, sem, "Row 2\n", 1);

out(mem, sem, "Row 4\n", 1);

shmdt(mem);

exit(0);

} else {

void\* mem = shmat(shmid, NULL, 0);

sembuf data[3];

for(int i=0; i<2; i++){

for(int j=0;j<2;j++) {

start(data,sem,j);

printf("%s", mem);

}

}

semctl(sem, 0, IPC\_RMID, NULL);

shmctl(shmid, IPC\_RMID, NULL);

}

return 0;

}

**СОКЕТЫ**

**socket** - создает объект для организации сетевых взаимодействий

#include <sys/socket.h>

int socket(

int domain, /\*домен адреса (AF\_UNIX, AF\_INET и т.п.)\*/

int type, /\*SOOK\_STREAM, SOOK\_DGRAM ...\*/

int protocol /\*детализирует type, обычно 0\*/

);

/\*возвр-ет файловый дескрптр или -1 в случае ошибки\*/

sockid = socket(AF\_UNIX,SOCK\_STREAM,0);

**bind** - присваивает адрес сокету

#include <sys/socket.h>

int bind(

int socket\_fd, /\*дескриптор сокета\*/

const struct sockaddr \*sa, /\*адрес сокета\*/

socklen\_t sa\_len /\*длина адреса\*/

);

/\*возвращает 0 в случае успеха, -1 в случае ошибки\*/

struct sockaddr st\_addr;

st\_addr.sa\_family = AF\_UNIX;

sprintf(st\_addr.sa\_data,"%s", "INADDR\_ANY");

bind(sockid, &st\_addr, sizeof(struct sockaddr));

**listen** - подготавливает сокет к приему запросов на соединение и устанавливает ограничение на р-р очереди запросов

#include <sys/socket.h>

int listen(

int socket\_fd, /\*дескриптор сокета\*/

int backlog /\*макс. число запросов на соединение в очереди\*/

);

/\*0 в случае успеха, -1 в случае ошибки\*/

listen(sockid, 10); - для сервера

**accept** - принимает новое соединение и создает новый сокет

#include <sys/socket.h>

int accept(

int socket\_fd, /\*дескрптр\*/

struct sockaddr \*sa,

socklen\_t \*sa\_len

);

/\*возвр-ет дескрпт, или -1 в случае ошибки\*/

sockid1 = accept(sockid, 0, 0); - для сервера

**connect** - устанавливает соединение

#include <sys/socket.h>

int connect(

int socket\_fd, /\*дескрпт\*/

const struct sockaddr \*sa, /\*адрес сокета\*/

socklen\_t sa\_len /\*длина адреса\*/

);

/\*0 успех, -1 - ошибка\*/

sockaddr st\_addr; - для клиента

in\_addr addr;

st\_addr.sa\_family = AF\_UNIX;

addr.s\_addr = INADDR\_ANY;

sprintf(st\_addr.sa\_data,"%s", "INADDR\_ANY");

//connection to server

connect(sockid, &st\_addr, sizeof(struct sockaddr));

**send** - передает данные в сокет

#include <sys/socket.h>

ssize\_t send(

int socket\_fd, /\*дескрпт\*/

const void \*data, /\*данные для передачи\*/

size\_t length, /\*объем передаваемых данных\*/

int flags /\*флаги\*/

);

/\*число перед. байт, или -1\*/

send(sockid, path, 30\*sizeof(char),MSG\_WAITALL); - ыжпэбвжыдирьелд

**recv** - принимает данные из сокета

#include <sys/socket.h>

ssize\_t recv(

int socket\_fd,

void \*buffer,

size\_t length,

int flags

);

/\*число принятых байт или -1\*/

recv(sockid1, path, 30\*sizeof(char), MSG\_DONTROUTE);

#include <iostream>

#include <fstream>

#include <sys/types.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <cstdlib>

#include <netinet/in.h>

#include <unistd.h>

#include <dirent.h>

#include <stdio.h>

using namespace std;

ofstream out;

int main()

{ int sockid;

sockaddr st\_addr;

in\_addr addr;

char mask[50], path[30], buf[1000];

//create socket

cout << "Client: creating socket." << endl;

if((sockid = socket(AF\_UNIX, SOCK\_STREAM, 0))==-1)

cout << "Client: Can't create socket." << endl, exit(1);

cout<<"Client: succsessfully created socket."<<endl;

//initialization of struct sockaddr & in\_addr

st\_addr.sa\_family = AF\_UNIX;

addr.s\_addr = INADDR\_ANY;

sprintf(st\_addr.sa\_data,"%s", "INADDR\_ANY");

//connection to server

cout << "Client: Connecting to server." << endl;

if (connect(sockid, &st\_addr, sizeof(struct sockaddr))==-1)

{

cout << "Client: Error conecting server." << endl;

close(sockid);

exit(1);

}

cout << "Client: Sucsessfully connected to server." << endl;

//ask for file path & mask

in\_path: cout << "Input file path: " << endl;

cin >> path;

if(!(opendir(path))) {cout << "It's not directory" << endl; goto in\_path;}

cout << "Input file mask: " << endl;

cin >> mask;

//sending to server information

cout << "Client: Sending to server search file info." << endl;

if (send(sockid, path, 30\*sizeof(char),MSG\_WAITALL)==-1) // send path

{

cout << "Client: Error sending info." << endl;

close(sockid);

exit(1);

}

if (send(sockid, mask, 30\*sizeof(char),MSG\_WAITALL)==-1) // send mask

{

cout << "Client: Error sending info." << endl;

close(sockid);

exit(1);

}

cout << "Client: Send file info sucsessfully." << endl;

if ((recv(sockid, buf, 1000\*sizeof(char), 0))!=-1) // receive file names

cout << "Client: Received from server info. " << endl;

out.open("output.txt");

if(!out) cout << "Error in open file 'output.txt'" << endl;

out << "Path is " << path << endl;

out << "Mask is " << mask << endl;

out << buf;

close(sockid);

cout << "Client: Closed socket." << endl;

}

------------------------------------------------------------------

#include <iostream>

#include <sys/types.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <cstdlib>

#include <netinet/in.h>

#include <unistd.h>

#include <stdio.h>

using namespace std;

int main()

{

int sockid, sockid1; // socked identificator

struct sockaddr st\_addr;

char path[30], mask[30];

char data[1000];

int pipeFd[2];

remove("INADDR\_ANY");

if ((sockid = socket(AF\_UNIX,SOCK\_STREAM,0)) == -1) // creating socket

cout << "Server: Error in creating socket." << endl, exit(1);

//initialization of struct sockaddr

st\_addr.sa\_family = AF\_UNIX;

sprintf(st\_addr.sa\_data,"%s", "INADDR\_ANY");

// binding socket

if(bind(sockid, &st\_addr, sizeof(struct sockaddr))==-1)

{

cout << "Server: Error binding." << endl;

close(sockid);

exit(1);

}

// turn on TCP-connections

cout << "Server: is trying to listen socket with id = " << sockid << endl;

if(listen(sockid, 10)==-1)

{

cout<<"Error listening for server1!"<<endl;

close(sockid);

exit(1);

}

cout << "Server: Listening." << endl;

cout << "Server: Creating another socket for receiving messages." << endl;

sockid1 = accept(sockid, 0, 0);

cout << "Server: Created another socket with id = " << sockid1 << endl;

cout << "Server: Waiting for a message from client." << endl;

if(recv(sockid1, path, 30\*sizeof(char), MSG\_DONTROUTE))

cout << "Server: Received from client info: " << path << endl;

if(recv(sockid1, mask, 30\*sizeof(char), MSG\_DONTROUTE))

cout<<"Server: Received from client info: " << mask << endl;

pipe(pipeFd);

if (!fork())

{

close(pipeFd[0]);

close(1);

dup(pipeFd[1]);

close(pipeFd[1]);

execlp("find", "find",path,"-name", mask, "-print",NULL);

exit(0);

}

close(pipeFd[1]);

for (int i = 0; ; i++)

{

if (read(pipeFd[0], &data[i], 1) == 0)

{

data[i] = NULL;

break;

}

}

if(send(sockid1,data,sizeof(data)-10,MSG\_WAITALL)!=-1)

cout << "Server: Send data." << endl;

cout << "Server: Closing socket." << endl;

close(sockid1);

close(sockid);

cout << "Server: Closed socket." << endl;

return 0;

}