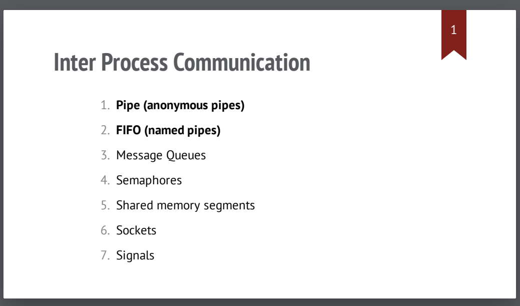
Operating systems 1, Lecture 8

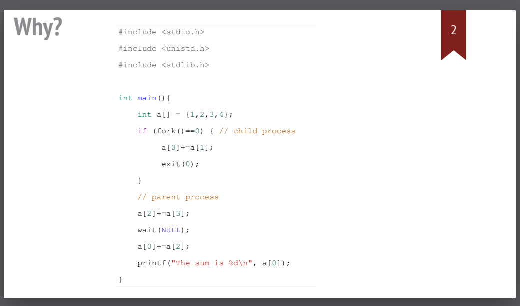
Sanda-Maria AVRAM, Ph. D.

Leave a feedback



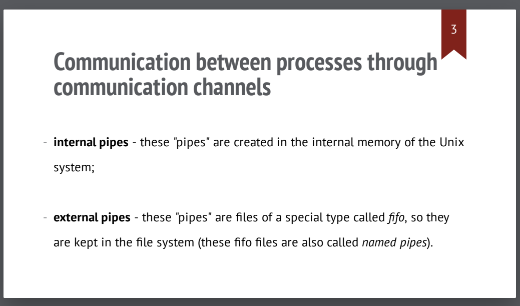
Inter Process Communication

1. **Pipe (anonymous pipes)**
2. **FIFO (named pipes)**
3. Message Queues
4. Semaphores
5. Shared memory segments
6. Sockets
7. Signals



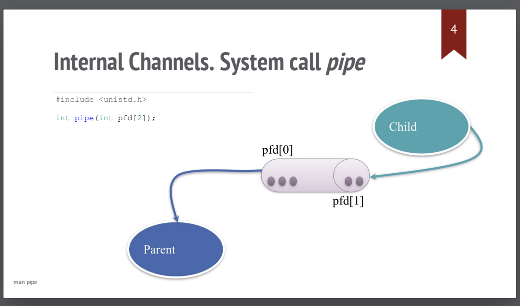
Why?

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(){
    int a[] = {1,2,3,4};
    if (fork()==0) { // child process
         a[0]+=a[1];
         exit(0);
    }
    // parent process
    a[2]+=a[3];
    wait(NULL);
    a[0]+=a[2];
    printf("The sum is %d\n", a[0]);
}
```



Communication between processes through communication channels

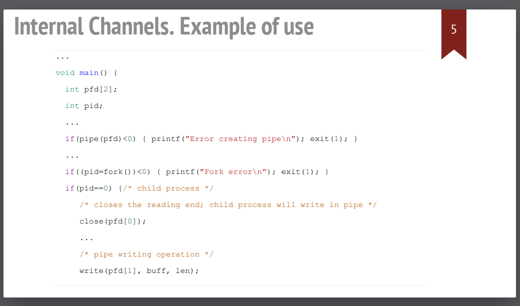
- **internal pipes** - these "pipes" are created in the internal memory of the Unix system;
- **external pipes** - these "pipes" are files of a special type called *fifo*, so they are kept in the file system (these fifo files are also called *named pipes*).



Internal Channels. System call *pipe*

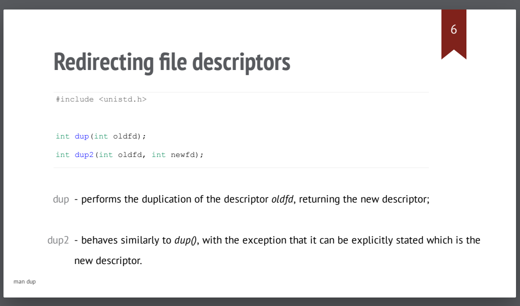
```
#include <unistd.h>
int pipe(int pfd[2]);
```

man pipe



Internal Channels. Example of use

```
...
void main() {
  int pfd[2];
  int pid;
  ...
  if(pipe(pfd)<0) { printf("Error creating pipe\n"); exit(1); }
  ...
  if((pid=fork())<0) { printf("Fork error\n"); exit(1); }
  if(pid==0) {/* child process */
     /* closes the reading end; child process will write in pipe */
     close(pfd[0]);
     ...
     /* pipe writing operation */
     write(pfd[1], buff, len);
```



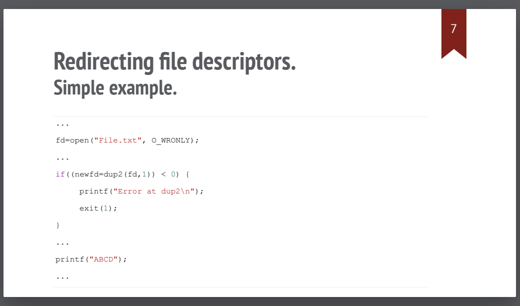
Redirecting file descriptors

```
#include <unistd.h>

int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

dup - performs the duplication of the descriptor *oldfd*, returning the new descriptor;

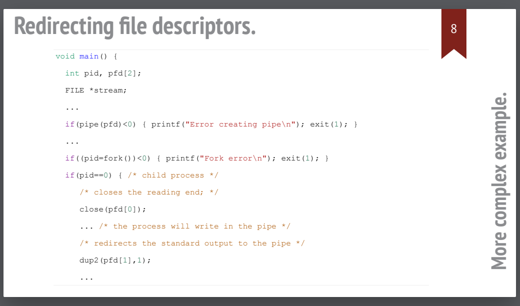
dup2 - behaves similarly to *dup()*, with the exception that it can be explicitly stated which is the new descriptor.

man dup



Redirecting file descriptors.
Simple example.

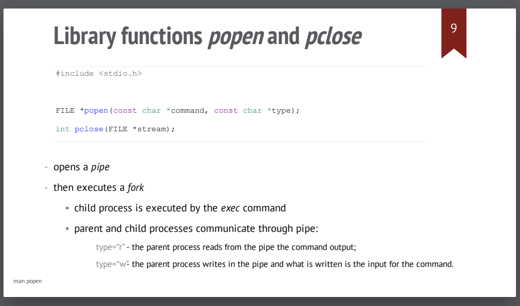
```
...
fd=open("File.txt", O_WRONLY);
...
if((newfd=dup2(fd,1)) < 0) {
     printf("Error at dup2\n");
     exit(1);
}
...
printf("ABCD");
...
```



Redirecting file descriptors.

More complex example.

```
void main() {
  int pid, pfd[2];
  FILE *stream;
  ...
  if(pipe(pfd)<0) { printf("Error creating pipe\n"); exit(1); }
  ...
  if((pid=fork())<0) { printf("Fork error\n"); exit(1); }
  if(pid==0) { /* child process */
     /* closes the reading end; */
     close(pfd[0]);
     ... /* the process will write in the pipe */
     /* redirects the standard output to the pipe */
     dup2(pfd[1],1);
     ...
```



Library functions *popen* and *pclose*

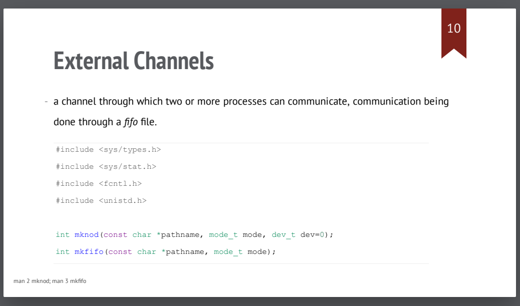
```
#include <stdio.h>

FILE *popen(const char *command, const char *type);
int pclose(FILE *stream);
```

- opens a *pipe*
- then executes a *fork*
  - child process is executed by the *exec* command
  - parent and child processes communicate through pipe:

    type="r" - the parent process reads from the pipe the command output;

    type="w"- the parent process writes in the pipe and what is written is the input for the command.

man popen

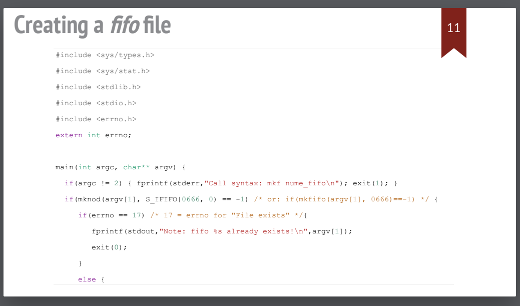


External Channels

- a channel through which two or more processes can communicate, communication being done through a *fifo* file.

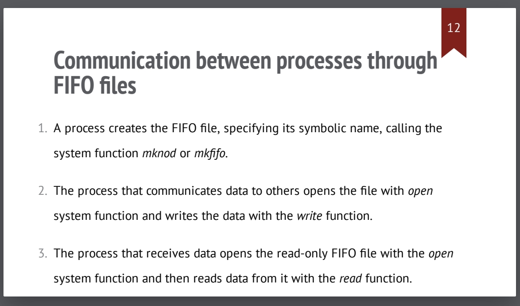
```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int mknod(const char *pathname, mode_t mode, dev_t dev=0);
int mkfifo(const char *pathname, mode_t mode);
```

man 2 mknod; man 3 mkfifo



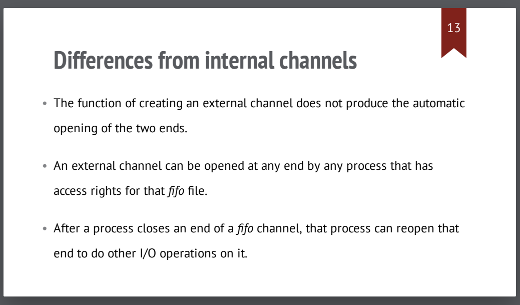
Creating a *fifo* file

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
extern int errno;

main(int argc, char** argv) {
  if(argc != 2) { fprintf(stderr,"Call syntax: mkf nume_fifo\n"); exit(1); }
  if(mknod(argv[1], S_IFIFO|0666, 0) == -1) /* or: if(mkfifo(argv[1], 0666)==-1) */ {
     if(errno == 17) /* 17 = errno for "File exists" */{
        fprintf(stdout,"Note: fifo %s already exists!\n",argv[1]);
        exit(0);
     }
     else {
```



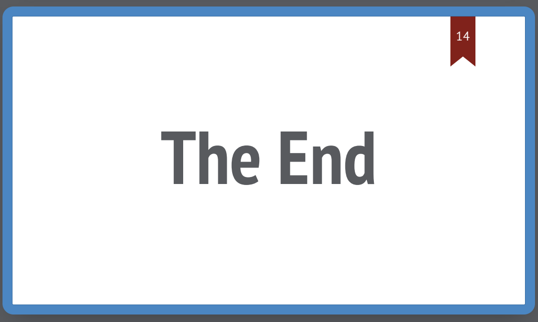
Communication between processes through FIFO files

1. A process creates the FIFO file, specifying its symbolic name, calling the system function *mknod* or *mkfifo*.
2. The process that communicates data to others opens the file with *open* system function and writes the data with the *write* function.
3. The process that receives data opens the read-only FIFO file with the *open* system function and then reads data from it with the *read* function.



Differences from internal channels

- The function of creating an external channel does not produce the automatic opening of the two ends.
- An external channel can be opened at any end by any process that has access rights for that *fifo* file.
- After a process closes an end of a *fifo* channel, that process can reopen that end to do other I/O operations on it.



The End