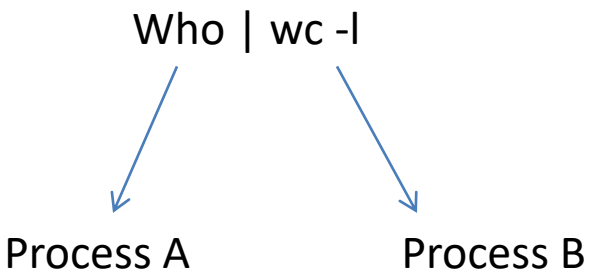
Pipe

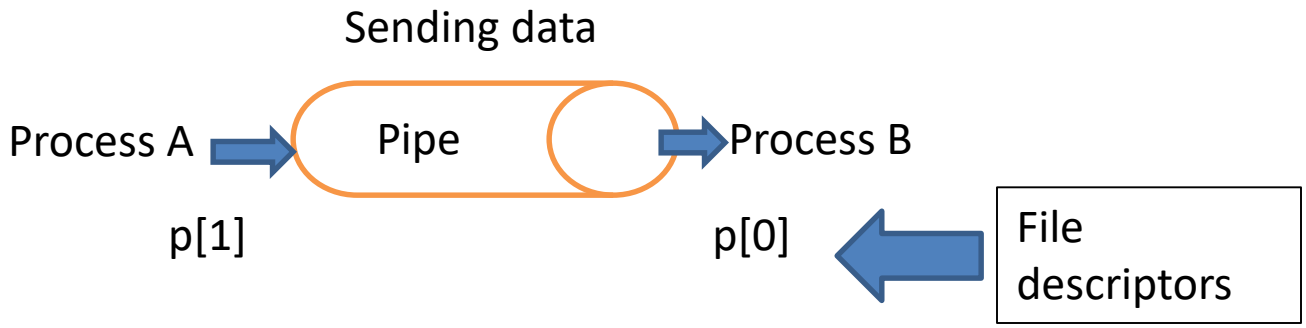
Pipe

- Interprocess communication primitive



Pipe

- Interprocess communication primitive




```
Main()
{
        int p[2];
        pipe(p);
        printf("%d %d",p[0],p[1]);
}
```

Pipe returns -1, if unsuccessful

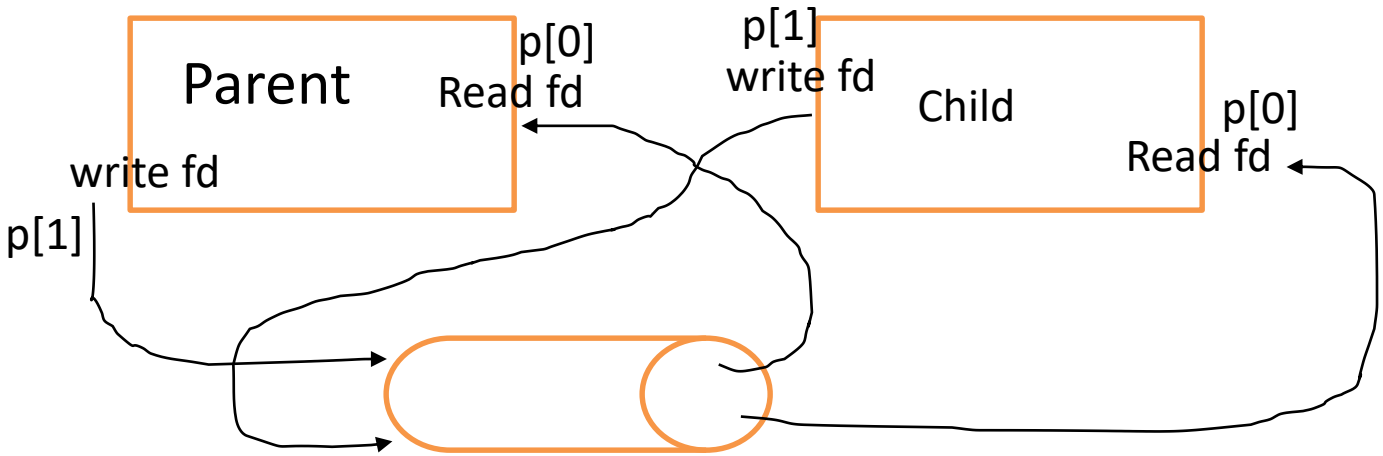
File descriptor table

File descriptor (integer)	File name
0	stdin
1	stdout
2	stderr

Use open(), read(), write() system calls to access files

Open() creates a file and returns fd (minimum value)

fd=open(path, O_WRONLY|O_CREAT)




```
Main()
{
        char *msg="hello";
        char buff[MAX];
        pipe(p);
        pid=fork();
```

```
#define MAX ****
Main()
{
        char *msg="hello";
        char buff[MAX];
        pipe(p);
        pid=fork();
        if(pid>0)
                write(p[1], msg1, MAX);
        else
        {
                sleep(1);
                read(p[0],buf, MAX);
                printf("%s", buff);
        }
}
```

Anybody can write (parent or child)

```
#define MAX ****
main()
{
        char *msg="hello";
        char buff[MAX];
        pipe(p);
        pid=fork();
        if(pid>0)
                write(p[1], msg1, MAX);
        else
        {
                sleep(1);
                write(p[1], msg1, MAX);
                for(i=0;i<2;i++)
                {
                        read(p[0],buf, MAX);
                        printf("%s", buff);
                }
        }
}
```

```
main()
{
        char *msg="hello";
        char buff[MAX];
        pipe(p);
        pid=fork();
        if(pid==0)
                printf("child exiting")
        else
        {
                read(p[0],buf, MAX);

        }
}
```

Read will wait since write end of parent is open.

Role of close()

```
main()
{
        char *msg="hello";
        char buff[MAX];
        pipe(p);
        pid=fork();
        if(pid==0)
        {
                printf("child exiting")
        }
        else
        {
                close(p[1]);
                read(p[0],buf, MAX);
        }
}
```

- Closing write end
- Read will return immediately.

Role of close()

```
main()
{
        char *msg="hello";
        char buff[MAX];
        pipe(p);
        pid=fork();
        if(pid==0)
        {
                sleep(5);
                printf("child exiting")
        }
        else
        {
                close(p[1]);
                read(p[0],buf, MAX);
        }
}
```

What will happen?

Closing the read end

```
main()
{
        char *msg="hello";
        char buff[MAX];
        pipe(p);
        pid=fork();
        if(pid==0)
        {

                printf("child exiting")
        }
        else
        {
                write(p[1],buf, MAX);
        }
}
```

Write will return successfully

```
main()
{
        char *msg="hello";
        char buff[MAX];
        pipe(p);
        pid=fork();
        if(pid==0)
        {

                printf("child exiting")
        }
        else
        {       sleep(1);
                close(p[0]);
                write(p[1],buf, MAX);
        }
}
```

- All the read ends are closed!
- Write returns -1
- Kernel generates SIGPIPE signal
- Terminates with "Broken pipe" message

int dup(int *oldfd***);**

The dup(fd) system call copies the file descriptor entry of fd into the FIRST EMPTY ENTRY in the file descriptor table.

File descriptor (integer)	File name
0	stdin
1	stdout
2	stderr

```c
int main()
{
    // open() returns a file descriptor file_desc to a
    // the file "dup.txt" here"

    int file_desc = open("dup.txt", O_WRONLY | O_APPEND);

    if(file_desc < 0)
        printf("Error opening the file\n");

    // dup() will create the copy of file_desc as the copy_desc
    // then both can be used interchangeably.

    int copy_desc = dup(file_desc);

    // write() will write the given string into the file
    // referred by the file descriptors

    write(copy_desc,"This will be output to the file named dup.txt\n", 46);

    write(file_desc,"This will also be output to the file named dup.txt\n", 51);

    return 0;
}
```

File descriptor table

File descriptor (integer)	File name
0	stdin
1	stdout
2	stderr

Use open(), read(), write() system calls to access files

Open() creates a file and returns fd (minimum value)

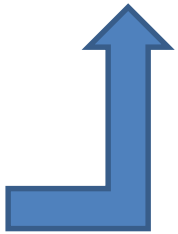
fd=open(path, O_WRONLY|O_CREAT)

Think what happens in case of redirection?

ls>file

File descriptor table

File descriptor (integer)	File name
0	stdin
1	stdout
2	stderr



fd_new=dup(fd)

Copy of fd

- Updates the FDT
- Inserts the fd_new at the first empty entry of FDT

int dup(int fd)

```c
#include<stdio.h>
#include<fcntl.h>
int main()
{

        int old,new;
        old=open("input.txt", O_WRONLY|O_CREAT);

//      new=dup2(old,1);

        close(1);
        new=dup(old);
        printf("hello world\n");
}
```

```
#include<fcntl.h>
int main()
{

        int fd[2],i,f,g,x, pid;
        char buf[20];
        pipe(fd);


        pid=fork();
        if(pid==0)
        {
                close(1);
                dup(fd[1]);
                printf("hello\n");
        }
        else
        {
                read(fd[0],buf,10);
                printf("%s\n",buf);


        }



}
```

popen()

- popen()
- FILE *popen(char *command, char *type)

```
FILE *fp;
int status;
char path[PATH_MAX];
fp = popen("ls -l", "r");
if (fp == NULL) /* Handle error */;

while (fgets(path, PATH_MAX, fp) != NULL)
printf("%s", path);
status = pclose(fp);
```

Named pipe or FIFO

- Named pipe or FIFO

/etc/mknod testpipe p

```
char * phrase = “Stuff this in your pipe
and smoke it”;

int main ()

{ int fd1;
fd1 = open ( “mypipe”, O_WRONLY );
write (fd1, phrase, strlen ( phrase)+1 );
close (fd1);

}
```

```
int main ()
{
int fd1;

char buf [100];

fd1 = open ( “mypipe”, O_RDONLY );
read ( fd1, buf, 100 );
printf ( “%s\n”, buf );
close (fd1);

}
```

- **int mknod(const char ****pathname***, mode_t** *mode***, dev_t** *dev***);**

- mknod("/tmp/MYFIFO", S_IFIFO|0666, 0)