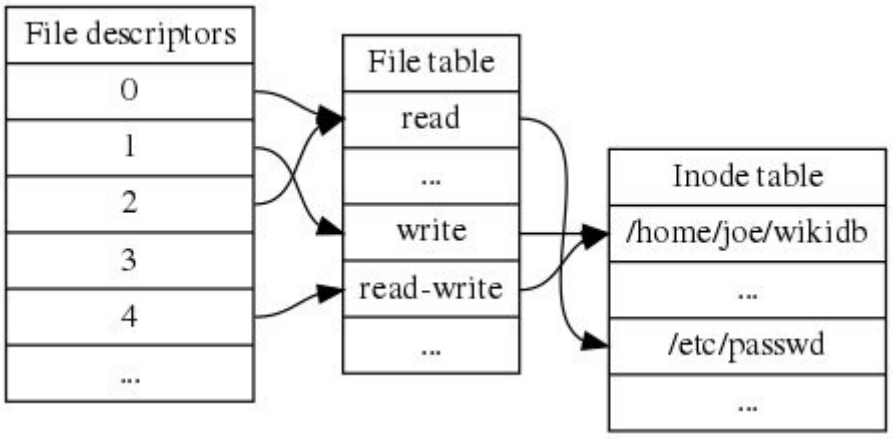


dup() vs dup2()

```
//Ilustrarea dup()
#include<stdio.h>
#include <unistd.h>
#include <fcntl.h>

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;
}
```

```
//Ilustrarea dup2()
#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>

int main()
{
    int file_desc = open("tricky.txt",O_WRONLY | O_APPEND);

    // here the newfd is the file descriptor of stdout (i.e. 1)
    dup2(file_desc, 1) ;

    // All the printf statements will be written in the file
    // "tricky.txt"
    printf("I will be printed in the file tricky.txt\n");

return 0;
}
```

I/O System Calls

Create - utilizat pentru a crea un fisier gol

Sintaxa in C: `int creat(char *filename, mode_t mode)`
Parametri:

- filename: numele fisierului pe care doriti sa il creati
- mode: indica permisiunile fisierului

Ce returneaza:

- primul file descriptor (fd) neutilizat (deobicei 3, pentru primul apel open() utilizat, intrucat file descriptorii 0,1,2 sunt rezervati)
- -1, in caz de eroare

```
// Ilustrarea apelului de sistem open
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>
extern int errno;
int main()
{
    // if file does not have in directory
    // then file foo.txt is created.
    int fd = open("foo.txt", O_RDONLY | O_CREAT);

    printf("fd = %d/n", fd);

    if (fd ==-1)
```

```
    {
        // print which type of error have in a code
        printf("Error Number % d\n", errno);

        // print program detail "Success or failure"
        perror("Program");
    }
    return 0;
}
```

Open - utilizat pentru a deschide un fisier pentru operatii de read, write sau ambele

Sintaxa in C:

```
#include<sys/types.h>
#includ<sys/stat.h>
#include <fcntl.h>
int open (const char* Path, int flags [, int mode ]);
```

Parametri:

- path: path-ul catre fisierul pe care doriti sa il folositi
- flags: seteaza modul de utilizare
    - O_RDONLY - doar pentru citire
    - O_WRONLY - doar pentru scriere
    - O_RDWR - pentru scriere si citire
    - O_CREAT - creeaza un fisier daca nu exista
    - O_EXCL - previne crearea, daca fisierul exista

Close - utilizat pentru a comunica sistemului de operare ca s-a incetat utilizarea unui file descriptor si pentru a se inchide fisierul care avea drept pointer acel file descriptor

Sintaxa in C:

```
#include <fcntl.h>
int close(int fd);
```

Parametri:

- fd: file descriptor

Ce returneaza:

- 0, pentru succes
- -1, pentru eroare

```
// Ilustrarea apelului de sistem close
#include<stdio.h>
#include <fcntl.h>
int main()
{
```

```
    int fd1 = open("foo.txt", O_RDONLY);
    if (fd1 < 0)
    {
        perror("c1");
        exit(1);
    }
    printf("opened the fd = % d\n", fd1);

    // Using close system Call
    if (close(fd1) < 0)
    {
        perror("c1");
        exit(1);
    }
    printf("closed the fd.\n");
}
```

Read - utilizat pentru a citi n bytes din input

Sintaxa in C: `size_t read (int fd, void* buf, size_t cnt);`

Parametri:

- fd: file descriptor
- buf: buffer din care se citesc date
- cnt: lungimea buffer-ului

Ce returneaza:

- numarul de bytes cititi, in caz de succes
- 0, cand se ajunge la end of file (EOF)
- -1, in caz de eroare

```
// Ilustrarea apelului de sistem read
#include<stdio.h>
#include <fcntl.h>
int main()
{
  int fd, sz;
  char *c = (char *) calloc(100, sizeof(char));

  fd = open("foo.txt", O_RDONLY);
  if (fd < 0) { perror("r1"); exit(1); }

  sz = read(fd, c, 10);
  printf("called read(% d, c, 10).  returned that"
         " %d bytes  were read.\n", fd, sz);
  c[sz] = '\0';
  printf("Those bytes are as follows: % s\n", c);
}
```

Write - utilizat pentru a scrie n bytes in fisierul asociat file descriptorului

Sintaxa in C:

```
#include <fcntl.h>
size_t write (int fd, void* buf, size_t cnt);
```

Parametri:

- fd: file descriptor
- buf: buffer in care se scriu date
- cnt: lungimea buffer-ului

Ce returneaza:

- numarul de bytes scrisi, in caz de succes
- 0, cand se ajunge la end of file (EOF)
- -1, in caz de eroare

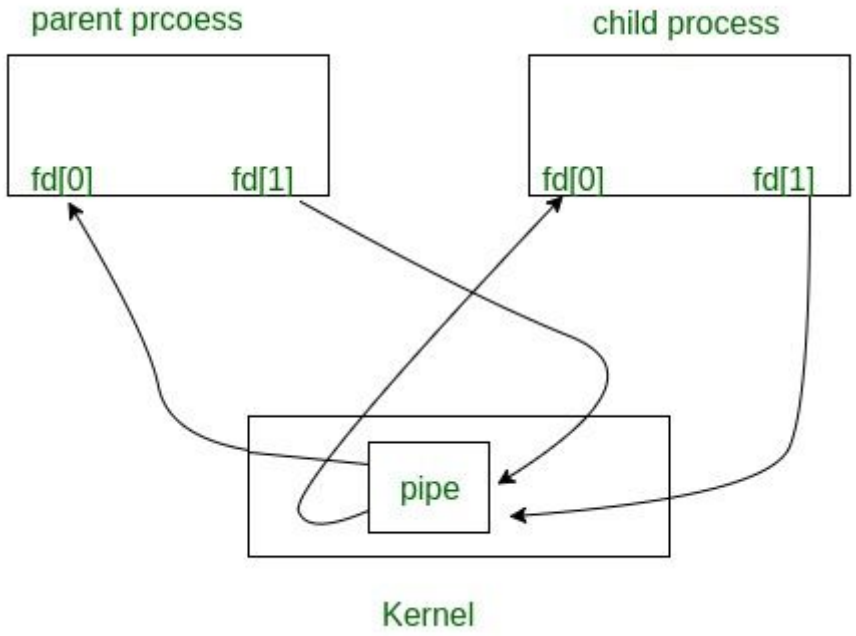
```
// Ilustrarea apelului de sistem write
#include<stdio.h>
#include <fcntl.h>
main()
{
  int sz;

  int fd = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fd < 0)
  {
     perror("r1");
     exit(1);
  }

  sz = write(fd, "hello\n", strlen("hello\n"));

  printf("called write(% d, \"hello\\n\", %d)."
    " It returned %d\n", fd, strlen("hello\n"), sz);

  close(fd);
}
```

Ilustrarea pipe system call in C

```
#include <stdio.h>
#include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";

int main()
{
```

```
    char inbuf[MSGSIZE];
    int p[2], i;

    if (pipe(p) < 0)
        exit(1);

    /* write pipe */

    write(p[1], msg1, MSGSIZE);
    write(p[1], msg2, MSGSIZE);
    write(p[1], msg3, MSGSIZE);

    for (i = 0; i < 3; i++) {
        /* read pipe */
        read(p[0], inbuf, MSGSIZE);
        printf("% s\n", inbuf);
    }
    return 0;
}
```

Partajarea unui pipe de catre procesul parinte si procesul fiu



Exemplu de program care ilustreaza comportamentul unui pipe cu nume. (FIFO)

Program 1

```
// This side writes first, then reads
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;

    // FIFO file path
    char * myfifo = "/tmp/myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>, <permission>)
    mkfifo(myfifo, 0666);

    char arr1[80], arr2[80];
    while (1)
    {
        // Open FIFO for write only
        fd = open(myfifo, O_WRONLY);

        // Take an input arr2ing from user.
        // 80 is maximum length
        fgets(arr2, 80, stdin);

        // Write the input arr2ing on FIFO
        // and close it
        write(fd, arr2, strlen(arr2)+1);
        close(fd);

        // Open FIFO for Read only
        fd = open(myfifo, O_RDONLY);

        // Read from FIFO
        read(fd, arr1, sizeof(arr1));

        // Print the read message
        printf("User2: %s\n", arr1);
        close(fd);
    }
    return 0;
}
```

Program 2

```
// This side reads first, then reads
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd1;

    // FIFO file path
    char * myfifo = "/tmp/myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>,<permission>)
    mkfifo(myfifo, 0666);

    char str1[80], str2[80];
    while (1)
    {
        // First open in read only and read
        fd1 = open(myfifo,O_RDONLY);
        read(fd1, str1, 80);

        // Print the read string and close
        printf("User1: %s\n", str1);
        close(fd1);

        // Now open in write mode and write
        // string taken from user.
        fd1 = open(myfifo,O_WRONLY);
        fgets(str2, 80, stdin);
        write(fd1, str2, strlen(str2)+1);
        close(fd1);
    }
    return 0;
}
```

Exercitii

1. Sa se scrie un program in care se foloseste un singur pipe pentru comunicare si in care:
    - tatal scrie intr-un pipe un numar
    - fiul verifica daca numarul este prim si transmite prin pipe tatalui raspunsul(yes,no)
    - tatal va afisa raspunsul primit.

2. Sa se scrie un program care sa simuleze comanda:
   cat prog.c | *grep "include"* > *prog.c* , folosind fifo-uri si dup().