Sisteme de operare 1

Curs 8

Titular curs,

Dr. Dragoș Sanda Maria

Comunicare intre procese Unix. (Inter Process Communication)

1. ***Pipe (anonymous pipes)***
2. ***FIFO - pipe cu nume (named pipes)***
3. *Cozi de mesaje*
4. *Semafoare*
5. *Segmente de memorie partajata*
6. *Socketuri (sockets)*
7. *Semnale*

De ce?

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

int main(){
    int a[] = {1,2,3,4};
    if (fork()==0) { //procesul fiu
        a[0]+=a[1];
        exit(0);
    }
    //procesul parinte
    a[2]+=a[3];
    wait(NULL);
    a[0]+=a[2];
    printf("Suma este %d\n", a[0]);
}
```

Comunicare intre procese prin intermediul canalelor de comunicatie

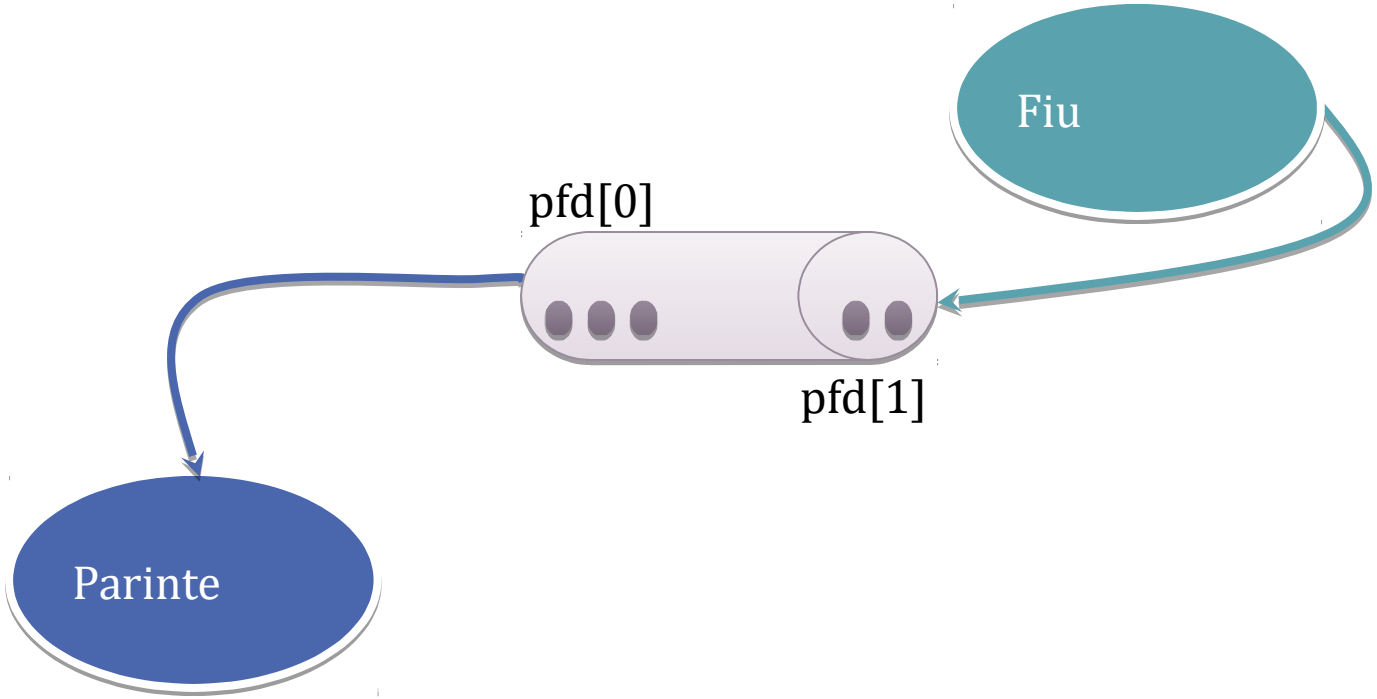
- ***pipe*-uri interne:** aceste "conducte" sunt create in memoria interna a sistemului Unix respectiv;

- ***pipe*-uri externe:** aceste "conducte" sunt fisiere de un tip special, numit *fifo*, deci sunt pastrate in sistemul de fisiere (aceste fisiere *fifo* se mai numesc si ***pipe*-uri cu nume**).

Canale Interne. Apelul sistem *pipe*

#include <unistd.h>

int pipe(int pfd[2]);



man pipe

Canale Interne. Exemplu de utilizare

```
...
void main() {
    int pfd[2];
    int pid;
    ...
    if(pipe(pfd)<0) {
            printf("Eroare la crearea pipe-ului\n");
            exit(1); }
    ...
    if((pid=fork())<0) {
            printf("Eroare la fork\n"); exit(1); }
    if(pid==0) { /* procesul fiu */
            /* inchide capatul de citire; */
            /* procesul fiu va scrie in pipe */
            close(pfd[0]);
            ...
             /* operatie de scriere in pipe */
            write(pfd[1], buff, len);
            ...
            /* la sfarsit inchide si capatul utilizat */
            close(pfd[1]);
        ...
        } else {     /* procesul parinte */
            /* inchide capatul de scriere; */
            /* procesul parinte va citi din pipe */
            close(pfd[1]);
            ...
            /* operatie de citire din pipe */
            read(pfd[0],buff,len);
            ...
             /* la sfarsit inchide si capatul utilizat */
            close(pfd[0]);
            ...
        }

}
```

Redirectarea descriptorilor de fisier

#include <unistd.h>

int dup(int oldfd);

int dup2(int oldfd, int newfd);

- ***dup( )*** realizeaza duplicarea descriptorului *oldfd*, returnand noul descriptor

- ***dup2( )*** se comporta in mod asemanator cu *dup()*, cu deosebirea ca poate fi indicat explicit care sa fie noul descriptor.

man dup

Redirectarea descriptorilor de fisier. Exemplu simplu.

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

Redirectarea descriptorilor de fisier. Exemplu.

```
void main() {
   int pid, pfd[2];     FILE *stream;
   ...
   if(pipe(pfd)<0) {
         printf("Eroare la crearea pipe-ului\n");
         exit(1);
   }
   ...
   if((pid=fork())<0) {
         printf("Eroare la fork\n");
         exit(1);
   }
   if(pid==0) {   /* procesul fiu */
        close(pfd[0]);  /* inchide capatul de citire; */
         ...                    /* procesul va scrie in pipe */
        dup2(pfd[1],1);  /* redirecteaza iesirea std. spre pipe */
        ...
         execlp("ls","ls","-l",NULL); /* executa ls */
         printf("Eroare la exec\n);
    }
   else {   /* procesul parinte */
        close(pfd[1]); /* inchide capatul de scriere; */
         ...                    /* procesul va citi din pipe */

       /* deschide un stream (FILE *) pentru capatul de citire */
        stream=fdopen(pfd[0],"r");
        /* citire din pipe, folosind stream-ul asociat */
        fscanf(stream,"%s",string);
            ...

        close(pfd[0]);  /* la sfarsit inchide si capatul utilizat */
        exit(0);
    }
}
```

Funcţiile de bibliotecă *popen* şi *pclose*

#include <stdio.h>

FILE *popen(const char *command, const char *type);

int pclose(FILE *stream);

- deschide un pipe
- apoi execută un fork
  - procesul fiu execută prin exec comanda
  - procesele tată şi fiu comunică prin pipe:
    - type="r" - procesul tată citeşte din pipe ieşirea standard dată de comanda
    - type="w" - procesul tată scrie în pipe iar ceea ce scrie se constituie în intrarea standard pentru comandă

man popen

Canale Externe

- un canal prin care pot comunica doua sau mai multe procese, comunicatia facandu-se printr-un fisier de tip *fifo*.

```
#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

Crearea unui fisier *fifo*

```
#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, "Sintaxa apel: mkf nume_fifo\n"); exit(1); }
     if( mknod(argv[1], S_IFIFO|0666, 0) == -1 ) /* sau: if(mkfifo(argv[1], 0666) == -1 ) */ {
           if(errno == 17) /* 17 = errno for "File exists" */{
                 fprintf(stdout,"Note: fifo %s exista deja !\n",argv[1]);
                 exit(0);}
           else {
                 fprintf(stderr,"Error: creare fifo imposibila, errno=%d\n",errno);
                 perror(0);
                 exit(2); }}
}
```

Comunicarea între procese prin fişiere FIFO

1. Un proces crează fişierul FIFO, precizând numele său simbolic, apelând funcţia de sistem *mknod* sau *mkfifo*.

2. Procesul care comunică date altora deschide fişierul cu funcţia de sistem *open* şi scrie datele respective cu funcţia *write*.

3. Procesul care primește date deschide fișierul FIFO în citire cu funcţia de sistem *open* și apoi citește datele din el cu funcţia *read*.

Deosebiri fata de canalele interne

1. Functia de creare a unui canal extern nu produce si deschiderea automata a celor doua capete.

2. Un canal extern poate fi deschis, la oricare din capete, de orice proces care are drepturi de acces pentru acel fisier *fifo*.

3. Dupa ce un proces inchide un capat al unui canal *fifo*, acel proces poate redeschide din nou acel capat pentru a face alte operatii I/O asupra sa.