- ```
  if((pid=fork())==0)
      } /* fiu */ }
  else
      {/* tata */}
  ```

- ```
  while(1)
    { parse-command();
      if((pid=fork())==0)
          { /* fiu */}
      else { /* tata */}
  ```

int wait(int *stare)

Codul de retur = { -1 => eroare => errno = ECHLD
                 { >0 => pid proces zombie

Când folosim „fork" trebuie să folosim și „wait".
„stare" trebuie să fie o adresă validă.

```
int *stare;
wait(stare);
```
} Gneșit

```
int *stare;
stare = (int *) malloc (sizeof(int));
wait(stare);
```
} Corect.

! La sfârșit să nu se uite eliberarea memoriei!

```
int stare;
wait(&stare)
```
} cel mai sigur și corect (cea mai bună variantă)

wait(NULL); ⇒ { NULL / ≠ NULL
↳ nu avem informații despre cum s-a terminat

#define DIM 1000
#define MAX(i,j) i<j?j:i
↳ cel mai corect: (i)<(j)?(j):(i)

MAX(a+2, b+c) // a+2<b+c? b+c : a+2

wait.h ← avem în ea macro-urile necesare când lucrăm cu stări.

Macro-uri

- WIFEXITED(stare) ← dacă s-a finalizat prin exit.
  WEXITSTATUS(stare) ← statusul la finalizare

```
if (WIFEXITED(stare))
    WEXITSTATUS(stare);
```

- WIFSIGNALED(stare) ← dacă s-a terminat printr-un semnal
  WITERMSIG(stare) ← ce semnal a fost.

```
int waitpid(int pid, int *stare, int opt)
pid > 0
pid = -1
opt WNOHANG => errno = EAGAIN
opt WUNTRACED
```

- WIFSTOPPED(stare) ← dacă a intrat în starea stopat
  WSTOPSIG(stare)

```
while(1)
{ parse-command();
  if ((pid=fork())==0) { /* fiu */ }
      else { wait(NULL); }
```

Lansarea programului din main.

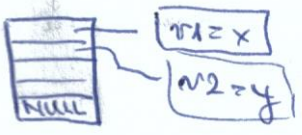
- int execl (char *exe, char *arg0, char *arg1, ..., char *argn, NULL)
  ↳ cale completă

  Exemplu: execl ("/bin/ls", "ls", "-l", NULL)

- int execlp (char *exe, char *arg0, ..., char *argn, NULL)
  ↳ nu e nevoie de cale completă

PATH
export PATH = .:$PATH

- int execle (char *exe, char *arg0, ..., char *argn, NULL, char *arge)
  ↳ cale completă



! arge trebuie construit în prealabil

- int execv (char *exe, char **argv)
  ↳ cale completă

! argv trebuie construit în prealabil.

- int execvp (char *exe, char **argv)
  ↳ nu e nevoie de cale completă

- int execve (char *exe, char **argv, char **arge)

Observație generală:

Cele care au "p" în coadă nu au nevoie de cale completă.
Cele care au "e" în coadă au și arge.

Comunicare între procese

1) comunicare naivă (nerecomandat)
2) fișiere de tip special: pipe, socket

pipe: 

socket: la fel ca pipe, dar bidirecțional

3) IPC System V -> cozi de mesaje, vectori de semafoare, zone de memorie partajată

4) semnale

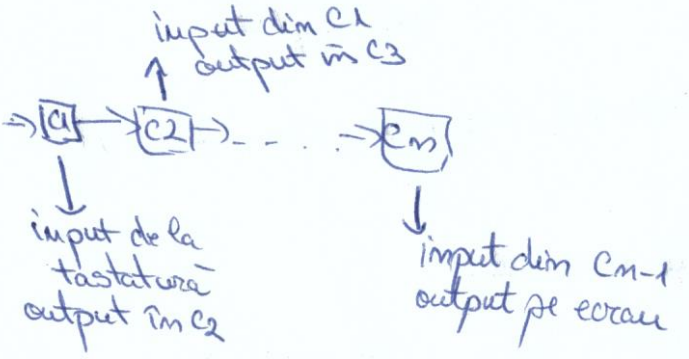
Fișiere standard

-> stdin -> descriptor 0
-> stdout -> descriptor 1
-> stderr -> descriptor 2

Obs.:
În UNIX, perifericele sunt considerate cazuri speciale de fișiere.

$ sort < f -> citirea din fișier; datele sunt preluate din fișierul f.
$ sort > f -> scrierea în fișier; datele sunt afișate în fișierul f.
$ sort >> f -> dacă fișierul f există, nu distruge conținutul ci adaugă la sfârșit datele

$ sort < f >> g
$ c1 | c2 | c3 | ... | cn
↳ comenzi



IPC System V

$ ipcs

identificator -> intern
-> extern

key_t ftok (char *f, int val)