Universitatea de Vest din Timişoara

Programare concurentă și distribuită - Lab 3

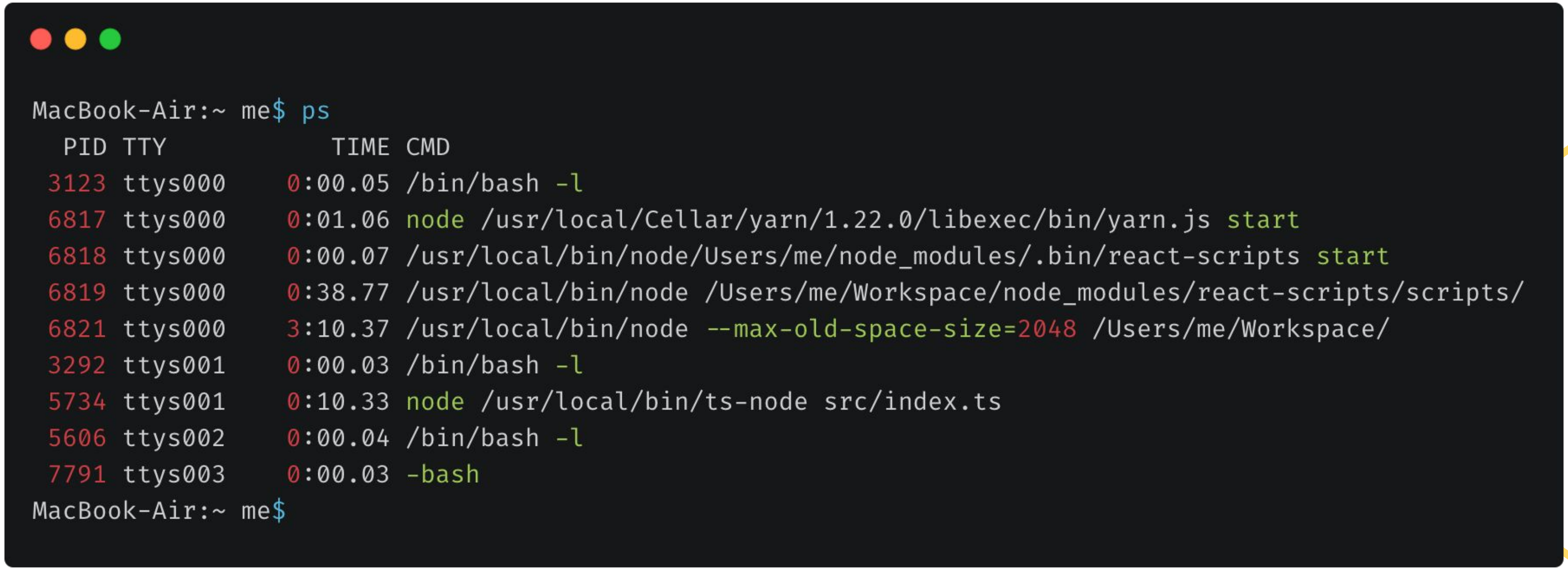
Faculty of Mathematics and Informatics
Department of Computer Science

Conținut

Laborator 3

PS? Ce înseamnă?

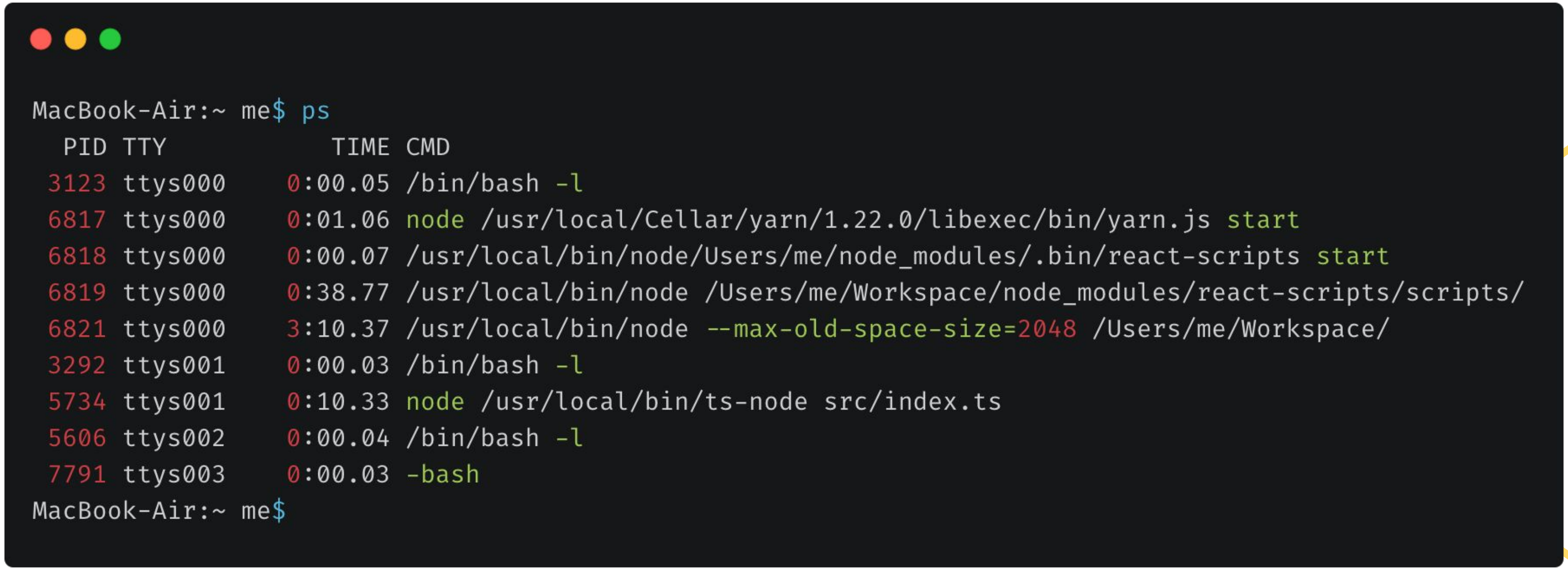
Scenariu - dacă sunt programator de React / Node JS
ps va arăta cam așa:



```
MacBook-Air:~ me$ ps
  PID TTY           TIME CMD
 3123 ttys000    0:00.05 /bin/bash -l
 6817 ttys000    0:01.06 node /usr/local/Cellar/yarn/1.22.0/libexec/bin/yarn.js start
 6818 ttys000    0:00.07 /usr/local/bin/node/Users/me/node_modules/.bin/react-scripts start
 6819 ttys000    0:38.77 /usr/local/bin/node /Users/me/Workspace/node_modules/react-scripts/scripts/
 6821 ttys000    3:10.37 /usr/local/bin/node --max-old-space-size=2048 /Users/me/Workspace/
 3292 ttys001    0:00.03 /bin/bash -l
 5734 ttys001    0:10.33 node /usr/local/bin/ts-node src/index.ts
 5606 ttys002    0:00.04 /bin/bash -l
 7791 ttys003    0:00.03 -bash
MacBook-Air:~ me$
```

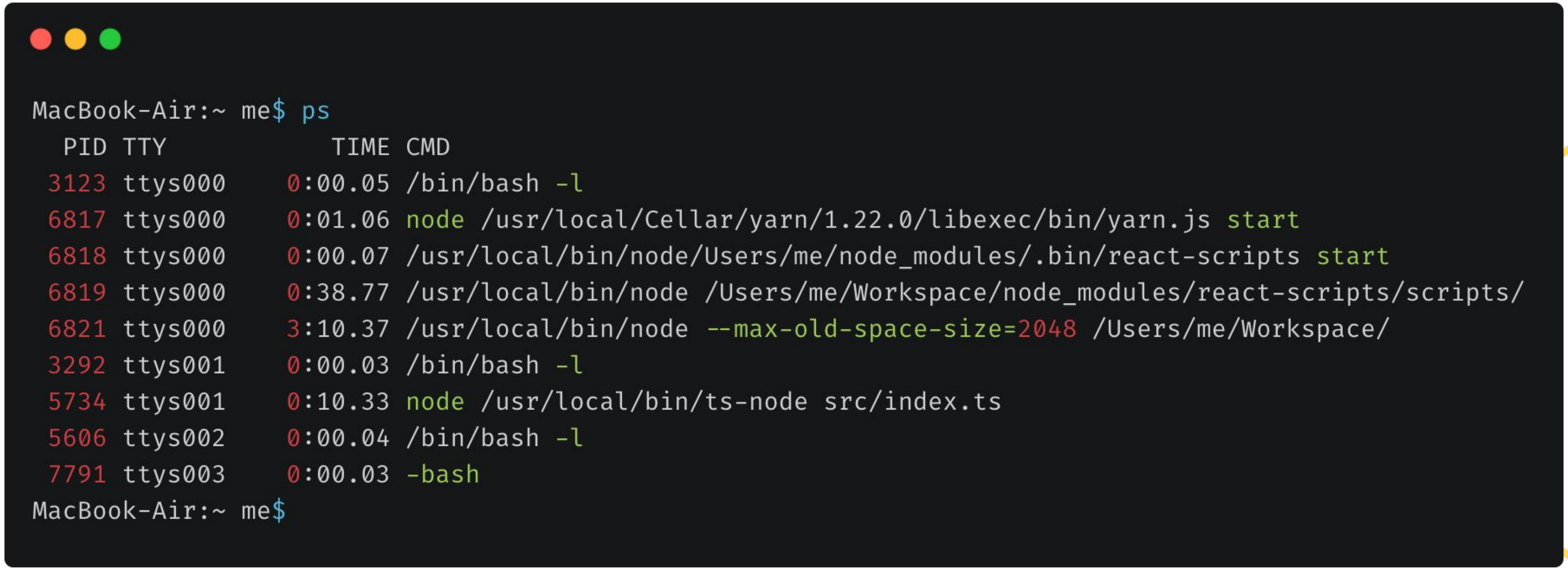
PS = processes status

Ce înseamnă coloana PID?



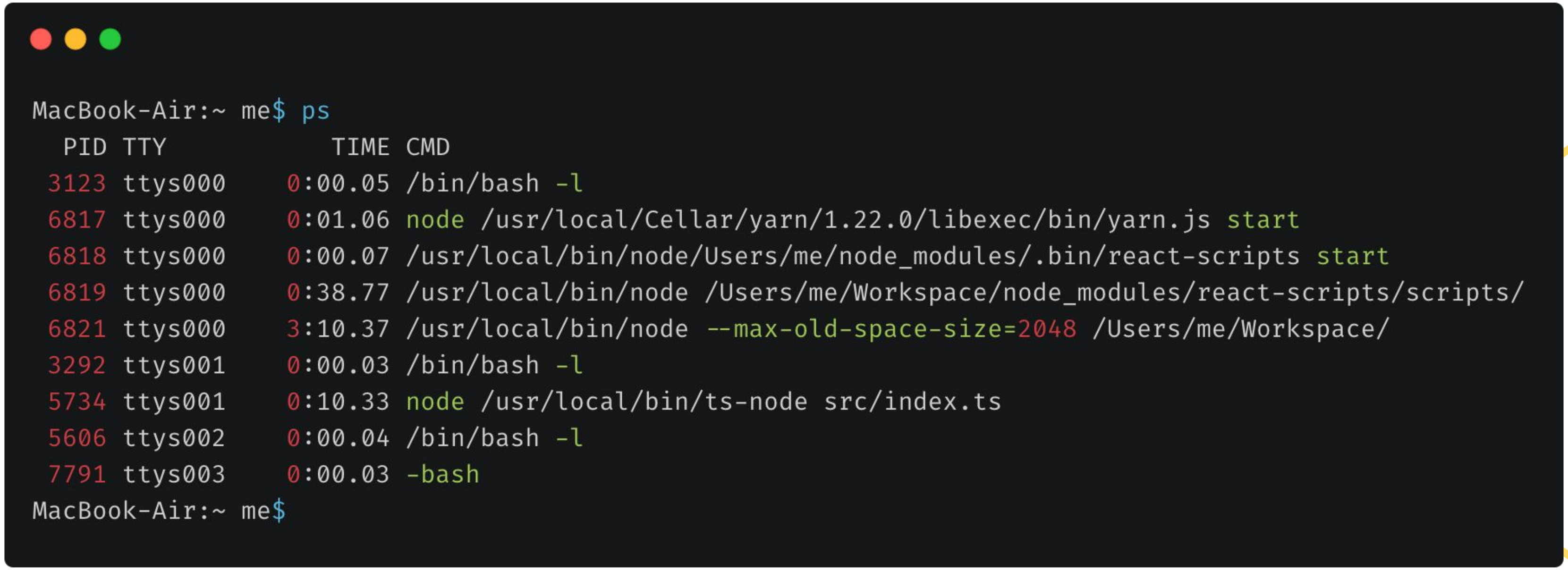
```
MacBook-Air:~ me$ ps
  PID TTY           TIME CMD
 3123 ttys000    0:00.05 /bin/bash -l
 6817 ttys000    0:01.06 node /usr/local/Cellar/yarn/1.22.0/libexec/bin/yarn.js start
 6818 ttys000    0:00.07 /usr/local/bin/node/Users/me/node_modules/.bin/react-scripts start
 6819 ttys000    0:38.77 /usr/local/bin/node /Users/me/Workspace/node_modules/react-scripts/scripts/
 6821 ttys000    3:10.37 /usr/local/bin/node --max-old-space-size=2048 /Users/me/Workspace/
 3292 ttys001    0:00.03 /bin/bash -l
 5734 ttys001    0:10.33 node /usr/local/bin/ts-node src/index.ts
 5606 ttys002    0:00.04 /bin/bash -l
 7791 ttys003    0:00.03 -bash
MacBook-Air:~ me$
```

PID = Process ID



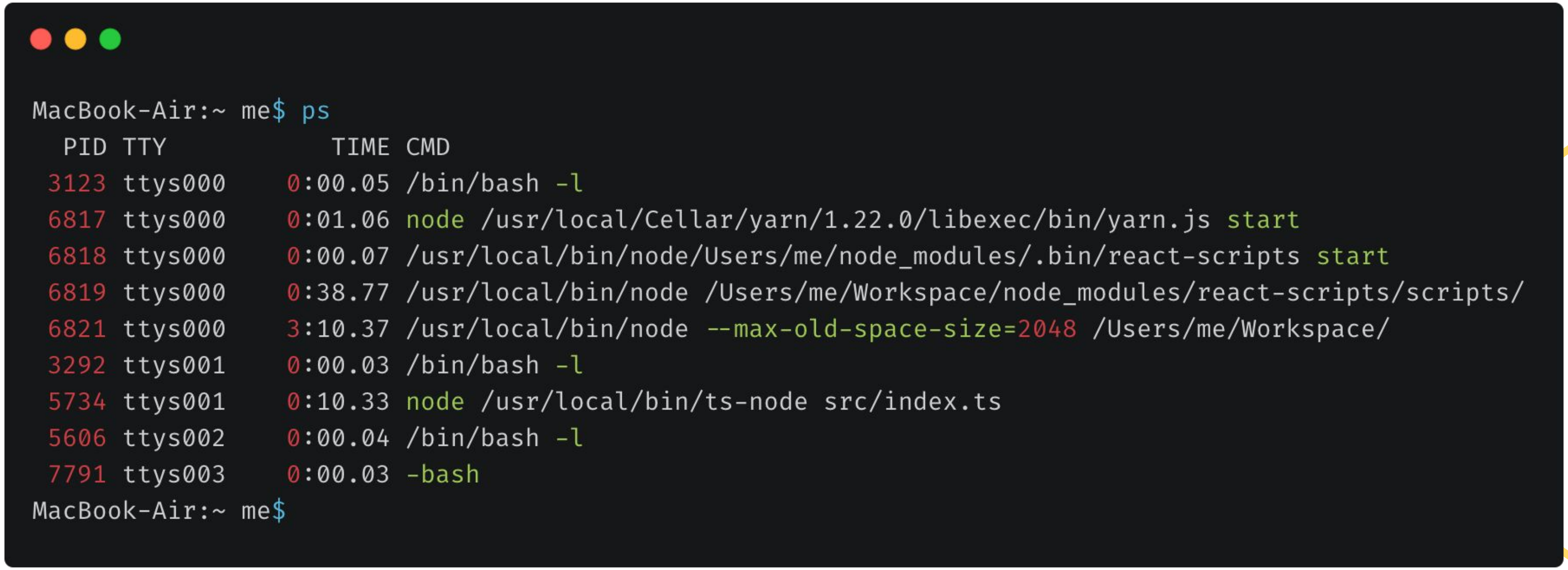
```
MacBook-Air:~ me$ ps
  PID TTY           TIME CMD
 3123 ttys000    0:00.05 /bin/bash -l
 6817 ttys000    0:01.06 node /usr/local/Cellar/yarn/1.22.0/libexec/bin/yarn.js start
 6818 ttys000    0:00.07 /usr/local/bin/node/Users/me/node_modules/.bin/react-scripts start
 6819 ttys000    0:38.77 /usr/local/bin/node /Users/me/Workspace/node_modules/react-scripts/scripts/
 6821 ttys000    3:10.37 /usr/local/bin/node --max-old-space-size=2048 /Users/me/Workspace/
 3292 ttys001    0:00.03 /bin/bash -l
 5734 ttys001    0:10.33 node /usr/local/bin/ts-node src/index.ts
 5606 ttys002    0:00.04 /bin/bash -l
 7791 ttys003    0:00.03 -bash
MacBook-Air:~ me$
```

TTY: Identifică terminalul de unde a fost executat procesul.



```
MacBook-Air:~ me$ ps
  PID TTY           TIME CMD
 3123 ttys000    0:00.05 /bin/bash -l
 6817 ttys000    0:01.06 node /usr/local/Cellar/yarn/1.22.0/libexec/bin/yarn.js start
 6818 ttys000    0:00.07 /usr/local/bin/node/Users/me/node_modules/.bin/react-scripts start
 6819 ttys000    0:38.77 /usr/local/bin/node /Users/me/Workspace/node_modules/react-scripts/scripts/
 6821 ttys000    3:10.37 /usr/local/bin/node --max-old-space-size=2048 /Users/me/Workspace/
 3292 ttys001    0:00.03 /bin/bash -l
 5734 ttys001    0:10.33 node /usr/local/bin/ts-node src/index.ts
 5606 ttys002    0:00.04 /bin/bash -l
 7791 ttys003    0:00.03 -bash
MacBook-Air:~ me$
```

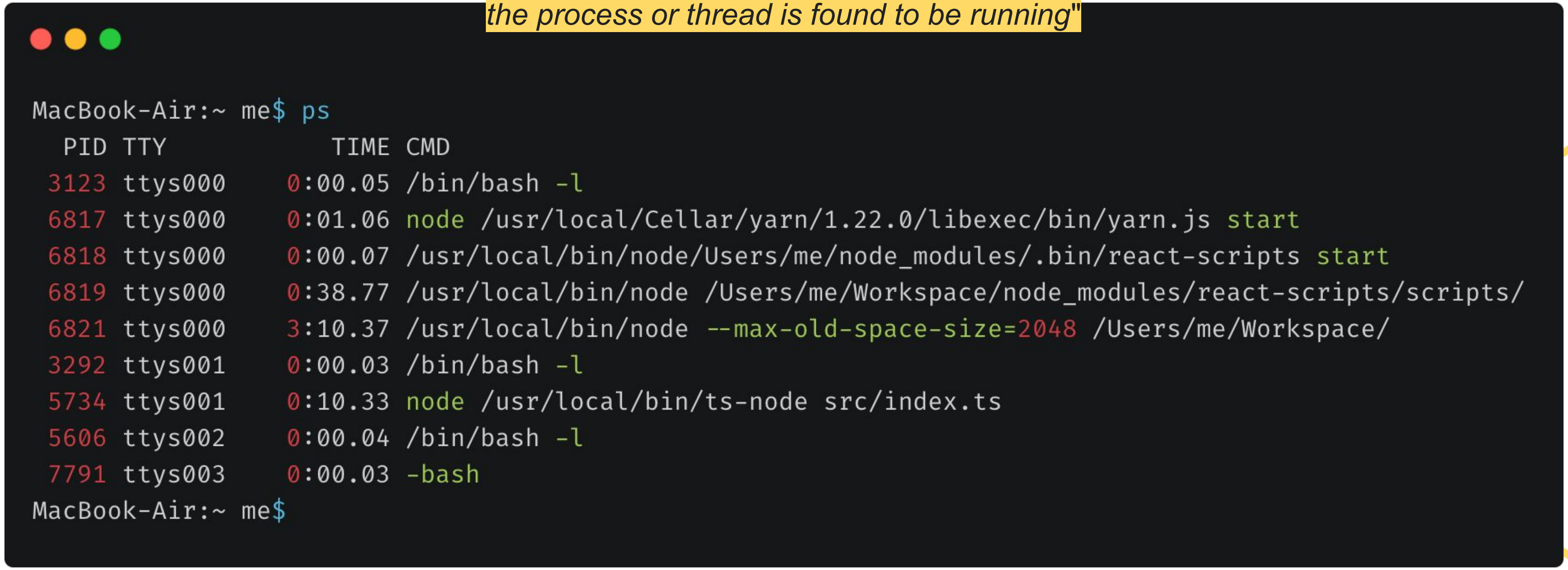
TIME: Timpul de procesare al procesului?



```
MacBook-Air:~ me$ ps
  PID TTY           TIME CMD
 3123 ttys000    0:00.05 /bin/bash -l
 6817 ttys000    0:01.06 node /usr/local/Cellar/yarn/1.22.0/libexec/bin/yarn.js start
 6818 ttys000    0:00.07 /usr/local/bin/node/Users/me/node_modules/.bin/react-scripts start
 6819 ttys000    0:38.77 /usr/local/bin/node /Users/me/Workspace/node_modules/react-scripts/scripts/
 6821 ttys000    3:10.37 /usr/local/bin/node --max-old-space-size=2048 /Users/me/Workspace/
 3292 ttys001    0:00.03 /bin/bash -l
 5734 ttys001    0:10.33 node /usr/local/bin/ts-node src/index.ts
 5606 ttys002    0:00.04 /bin/bash -l
 7791 ttys003    0:00.03 -bash
MacBook-Air:~ me$
```

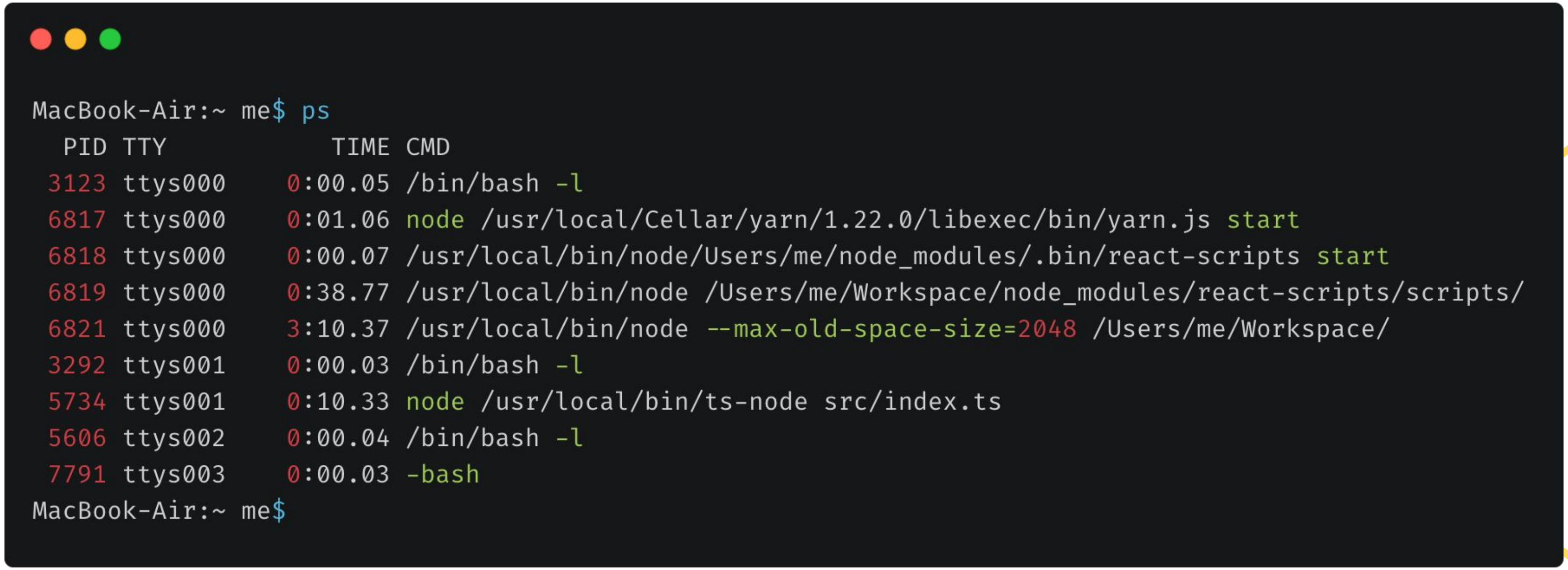
TIME: Timpul de procesare al procesului?

Mai exact - "*CPU utilization of process or thread, incremented each time the system clock ticks and the process or thread is found to be running*"



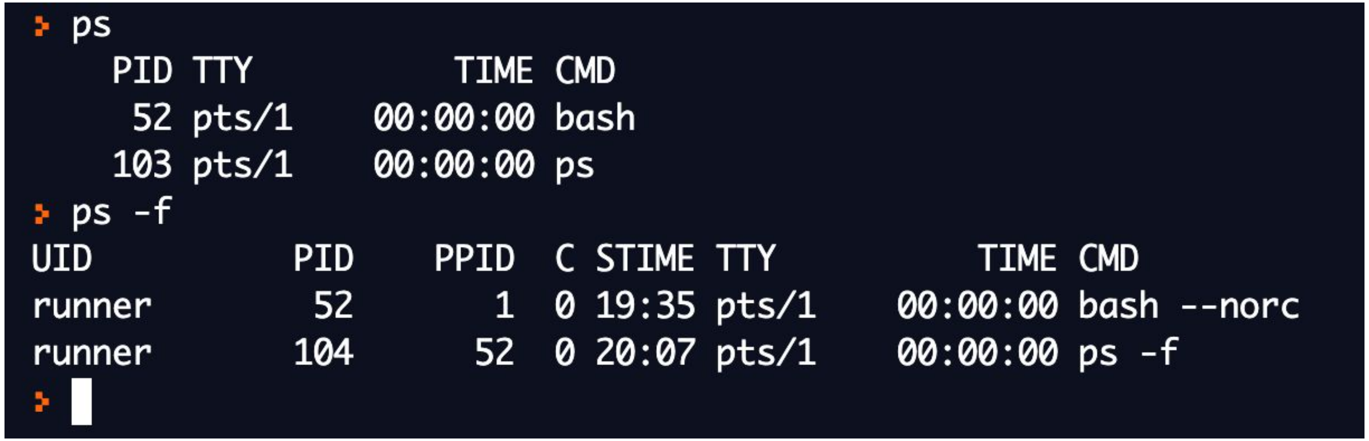
```
MacBook-Air:~ me$ ps
  PID TTY           TIME CMD
 3123 ttys000    0:00.05 /bin/bash -l
 6817 ttys000    0:01.06 node /usr/local/Cellar/yarn/1.22.0/libexec/bin/yarn.js start
 6818 ttys000    0:00.07 /usr/local/bin/node/Users/me/node_modules/.bin/react-scripts start
 6819 ttys000    0:38.77 /usr/local/bin/node /Users/me/Workspace/node_modules/react-scripts/scripts/
 6821 ttys000    3:10.37 /usr/local/bin/node --max-old-space-size=2048 /Users/me/Workspace/
 3292 ttys001    0:00.03 /bin/bash -l
 5734 ttys001    0:10.33 node /usr/local/bin/ts-node src/index.ts
 5606 ttys002    0:00.04 /bin/bash -l
 7791 ttys003    0:00.03 -bash
MacBook-Air:~ me$
```

CMD: Comanda cu care s-a lansat procesul.



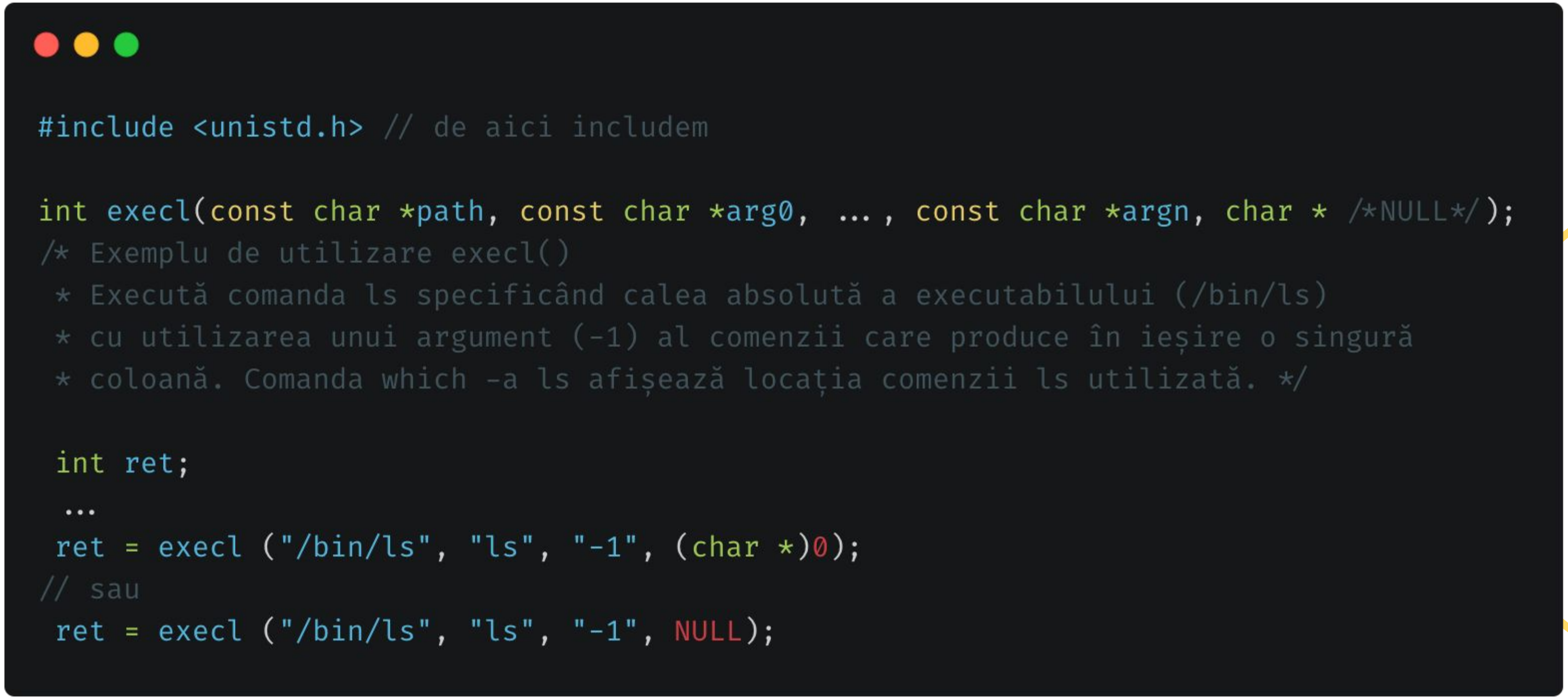
```
MacBook-Air:~ me$ ps
  PID TTY           TIME CMD
 3123 ttys000    0:00.05 /bin/bash -l
 6817 ttys000    0:01.06 node /usr/local/Cellar/yarn/1.22.0/libexec/bin/yarn.js start
 6818 ttys000    0:00.07 /usr/local/bin/node/Users/me/node_modules/.bin/react-scripts start
 6819 ttys000    0:38.77 /usr/local/bin/node /Users/me/Workspace/node_modules/react-scripts/scripts/
 6821 ttys000    3:10.37 /usr/local/bin/node --max-old-space-size=2048 /Users/me/Workspace/
 3292 ttys001    0:00.03 /bin/bash -l
 5734 ttys001    0:10.33 node /usr/local/bin/ts-node src/index.ts
 5606 ttys002    0:00.04 /bin/bash -l
 7791 ttys003    0:00.03 -bash
MacBook-Air:~ me$
```

ps vs ps -f ?
F = format



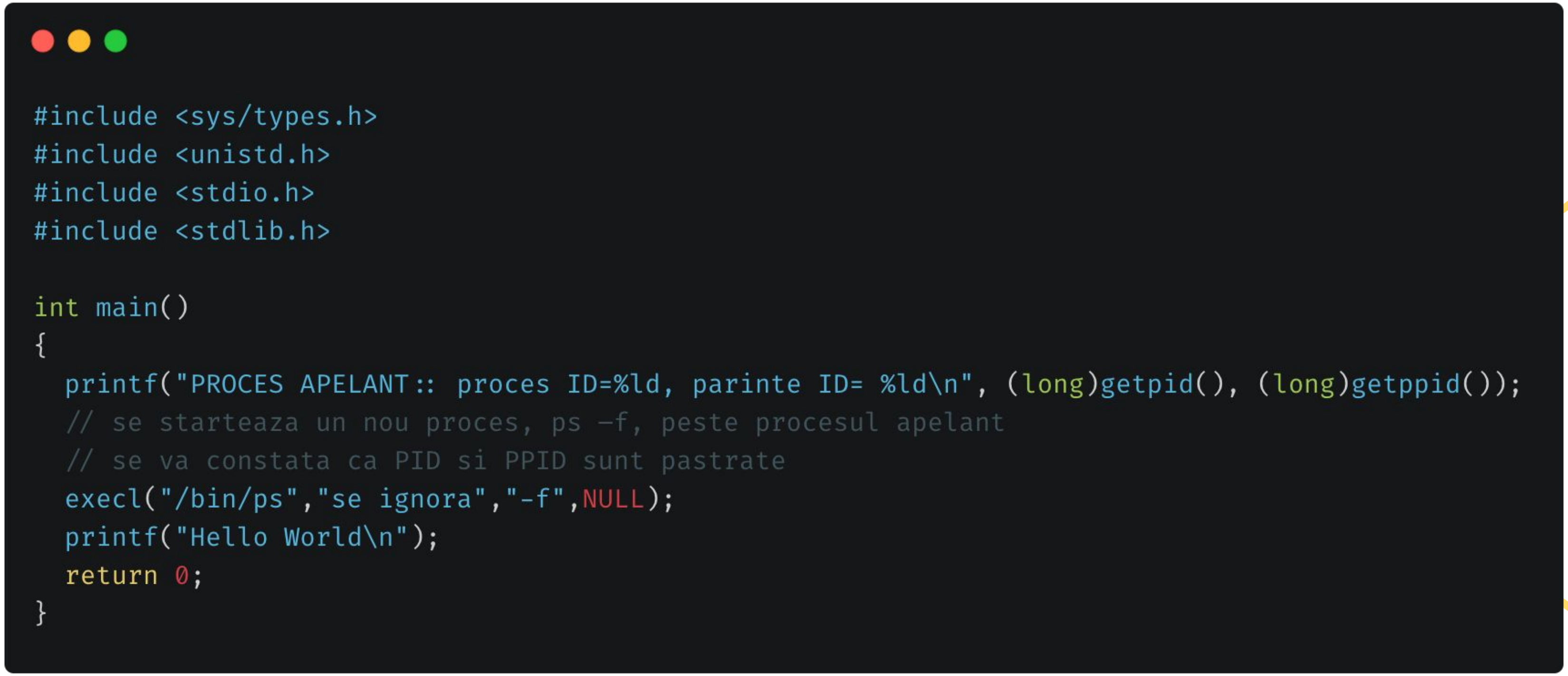
```
› ps
    PID TTY          TIME CMD
     52 pts/1    00:00:00 bash
    103 pts/1    00:00:00 ps
› ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
runner        52       1  0 19:35 pts/1    00:00:00 bash --norc
runner       104      52  0 20:07 pts/1    00:00:00 ps -f
› 
```

Exec



```c
#include <unistd.h> // de aici includem

int execl(const char *path, const char *arg0, ... , const char *argn, char * /*NULL*/);
/* Exemplu de utilizare execl()
 * Execută comanda ls specificând calea absolută a executabilului (/bin/ls)
 * cu utilizarea unui argument (-1) al comenzii care produce în ieșire o singură
 * coloană. Comanda which -a ls afișează locația comenzii ls utilizată. */

 int ret;
 ...
 ret = execl ("/bin/ls", "ls", "-1", (char *)0);
// sau
 ret = execl ("/bin/ls", "ls", "-1", NULL);
```

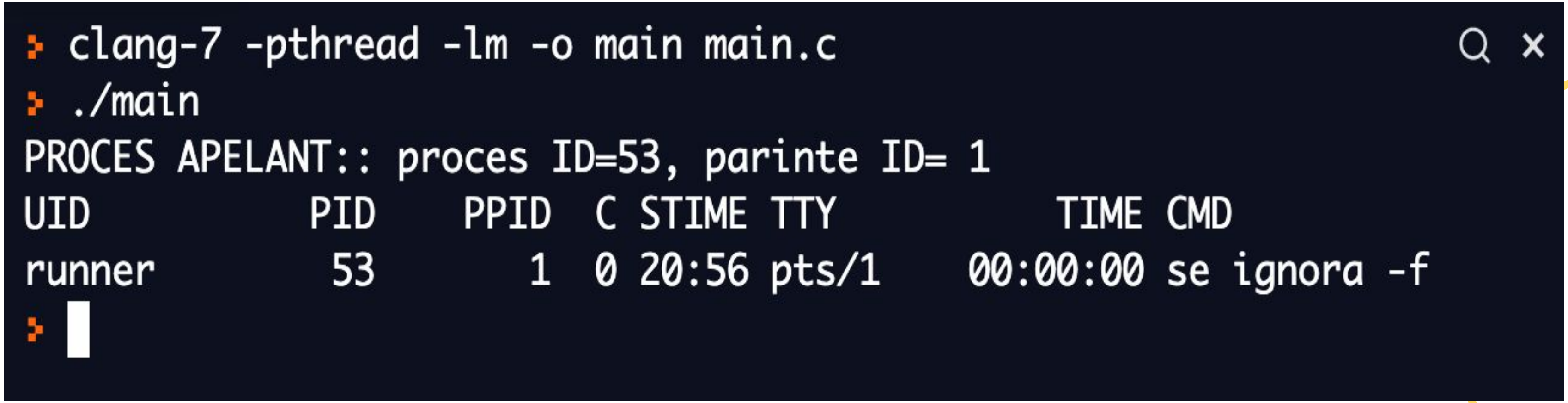
Primul program cu exec



```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  printf("PROCES APELANT:: proces ID=%ld, parinte ID= %ld\n", (long)getpid(), (long)getppid());
  // se starteaza un nou proces, ps -f, peste procesul apelant
  // se va constata ca PID si PPID sunt pastrate
  execl("/bin/ps","se ignora","-f",NULL);
  printf("Hello World\n");
  return 0;
}
```

Se afișează “Hello World”?

PID-ul nostru e 53.
Comanda executată - ps - are tot PID 53.



```
> clang-7 -pthread -lm -o main main.c
> ./main
PROCES APELANT:: proces ID=53, parinte ID= 1
UID          PID    PPID  C STIME TTY          TIME CMD
runner        53       1  0 20:56 pts/1    00:00:00 se ignora -f
>
```

Răspunsul e nu.
De ce: noul proces reacoperă procesul ce a executat apelul exec.

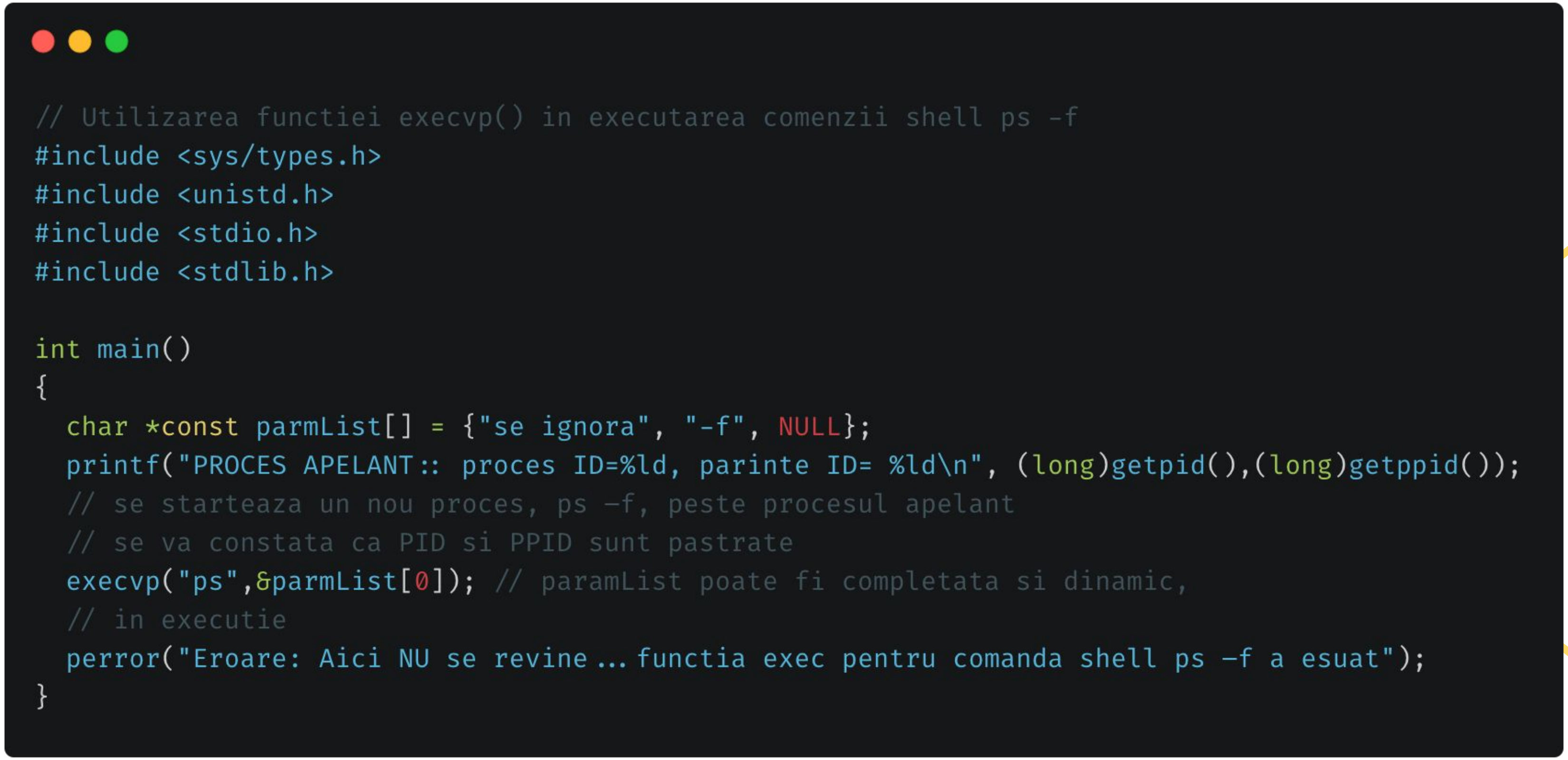
Observații:

- La fork avem două procese separate ce execută aceleași instrucțiuni.

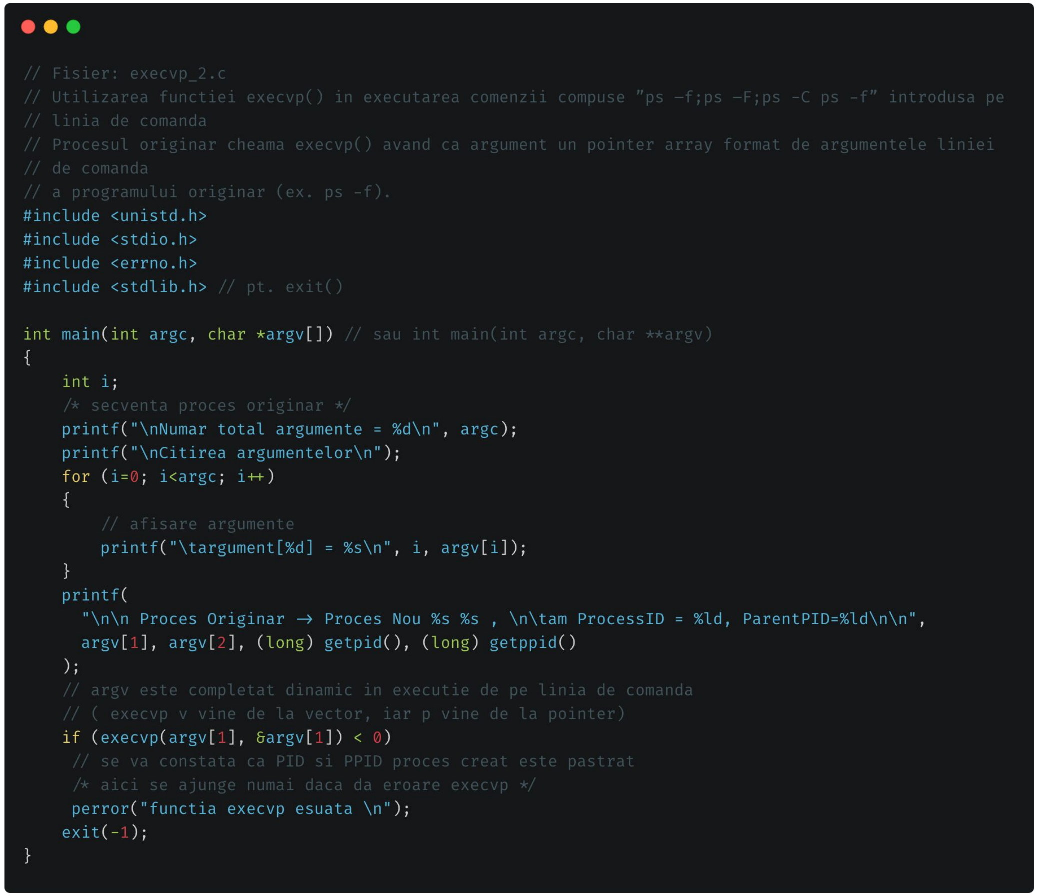
- La exec, instrucțiunile noastre sunt înlocuite.

- Noile instrucțiuni sunt executate sub același PID (și fișierele / descriptorii raman la fel).

execvp

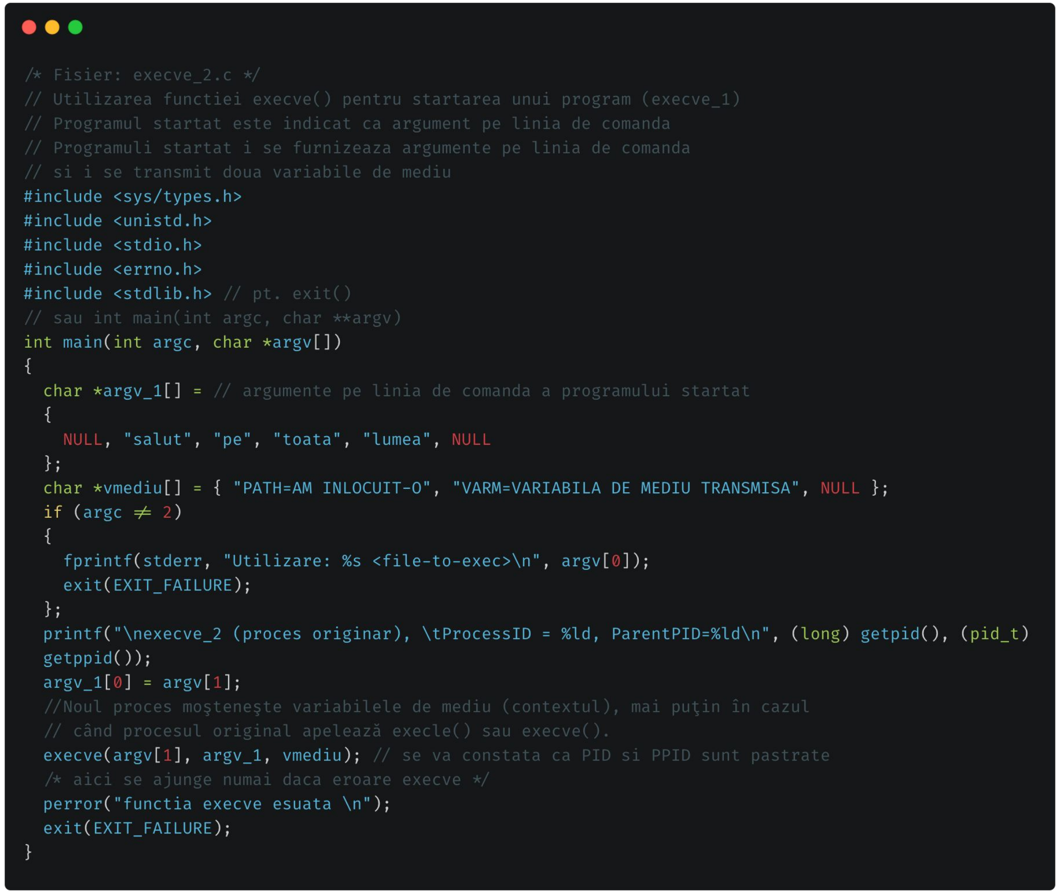


```c
// Utilizarea functiei execvp() in executarea comenzii shell ps -f
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  char *const parmList[] = {"se ignora", "-f", NULL};
  printf("PROCES APELANT :: proces ID=%ld, parinte ID= %ld\n", (long)getpid(),(long)getppid());
  // se starteaza un nou proces, ps -f, peste procesul apelant
  // se va constata ca PID si PPID sunt pastrate
  execvp("ps",&parmList[0]); // paramList poate fi completata si dinamic,
  // in executie
  perror("Eroare: Aici NU se revine ... functia exec pentru comanda shell ps -f a esuat");
}
```



```c
// Fisier: execvp_2.c
// Utilizarea functiei execvp() in executarea comenzii compuse ”ps –f;ps –F;ps -C ps -f” introdusa pe
// linia de comanda
// Procesul originar cheama execvp() avand ca argument un pointer array format de argumentele liniei
// de comanda
// a programului originar (ex. ps -f).
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h> // pt. exit()

int main(int argc, char *argv[]) // sau int main(int argc, char **argv)
{
    int i;
    /* secventa proces originar */
    printf("\nNumar total argumente = %d\n", argc);
    printf("\nCitirea argumentelor\n");
    for (i=0; i<argc; i++)
    {
        // afisare argumente
        printf("\targument[%d] = %s\n", i, argv[i]);
    }
    printf(
      "\n\n Proces Originar -> Proces Nou %s %s , \n\tam ProcessID = %ld, ParentPID=%ld\n\n",
      argv[1], argv[2], (long) getpid(), (long) getppid()
    );
    // argv este completat dinamic in executie de pe linia de comanda
    // ( execvp v vine de la vector, iar p vine de la pointer)
    if (execvp(argv[1], &argv[1]) < 0)
     // se va constata ca PID si PPID proces creat este pastrat
     /* aici se ajunge numai daca da eroare execvp */
     perror("functia execvp esuata \n");
    exit(-1);
}
```

execve



```c
/* Fisier: execve_2.c */
// Utilizarea functiei execve() pentru startarea unui program (execve_1)
// Programul startat este indicat ca argument pe linia de comanda
// Programuli startat i se furnizeaza argumente pe linia de comanda
// si i se transmit doua variabile de mediu
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h> // pt. exit()
// sau int main(int argc, char **argv)
int main(int argc, char *argv[])
{
  char *argv_1[] = // argumente pe linia de comanda a programului startat
  {
    NULL, "salut", "pe", "toata", "lumea", NULL
  };
  char *vmediu[] = { "PATH=AM INLOCUIT-O", "VARM=VARIABILA DE MEDIU TRANSMISA", NULL };
  if (argc != 2)
  {
    fprintf(stderr, "Utilizare: %s <file-to-exec>\n", argv[0]);
    exit(EXIT_FAILURE);
  };
  printf("\nexecve_2 (proces originar), \tProcessID = %ld, ParentPID=%ld\n", (long) getpid(), (pid_t)
  getppid());
  argv_1[0] = argv[1];
  //Noul proces moşteneşte variabilele de mediu (contextul), mai puţin în cazul
  // când procesul original apelează execle() sau execve().
  execve(argv[1], argv_1, vmediu); // se va constata ca PID si PPID sunt pastrate
  /* aici se ajunge numai daca eroare execve */
  perror("functia execve esuata \n");
  exit(EXIT_FAILURE);
}
```

De reținut:

The base of each is **exec** (execute), followed by one or more letters:

- **e** – An array of pointers to environment variables is explicitly passed to the new process image.
- **l** – Command-line arguments are passed individually (a **l**ist) to the function.
- **p** – Uses the PATH environment variable to find the file named in the *file* argument to be executed.
- **v** – Command-line arguments are passed to the function as an array (**v**ector) of pointers.

Rezumat:

1. Diferența dintre apelurile din familia exec este mai mult sintactică.
2. La execl, execlp, execv, si execvp, noile procese moștenesc variabilele de mediu, restul nu.