

Programare concurentă și distribuită - Lab 1

Faculty of Mathematics and Informatics
Department of Computer Science

Conținut

Introducere

- Cerinte / Evaluare
- Întrebări

Laborator 1

- Apeluri sistem.
- Funcții de bibliotecă.
- Unelte de programare/compilatoare C în UNIX.
- Tratarea erorilor.
- Procesarea liniei de comandă.

Cerinte / Evaluable

Cerinte:

- Prezentă laborator 70% (9 prezente)
- Prezentă laborator angajați (5 prezente)

Evaluable:

- Teme laborator
- Teste laborator
- Activitate

Calcul nota laborator:

$$N_{lab} = .2 * N_{teme} + .6 * N_{testare} + .2 * N_{bonus} + .2 * Proba_practica$$



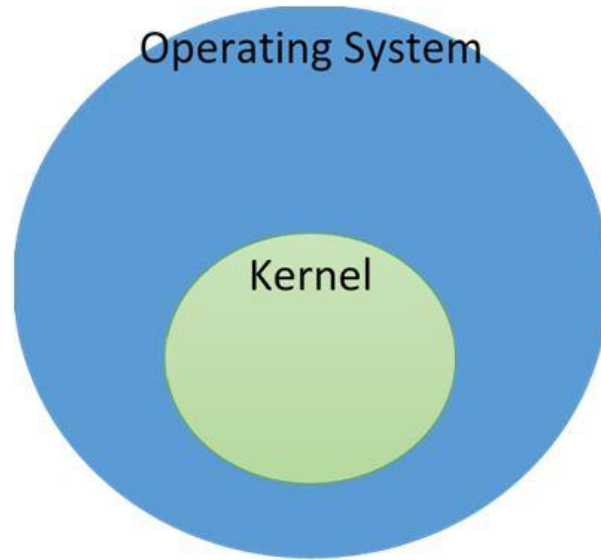
Classroom
fv7erlv



Întrebări?

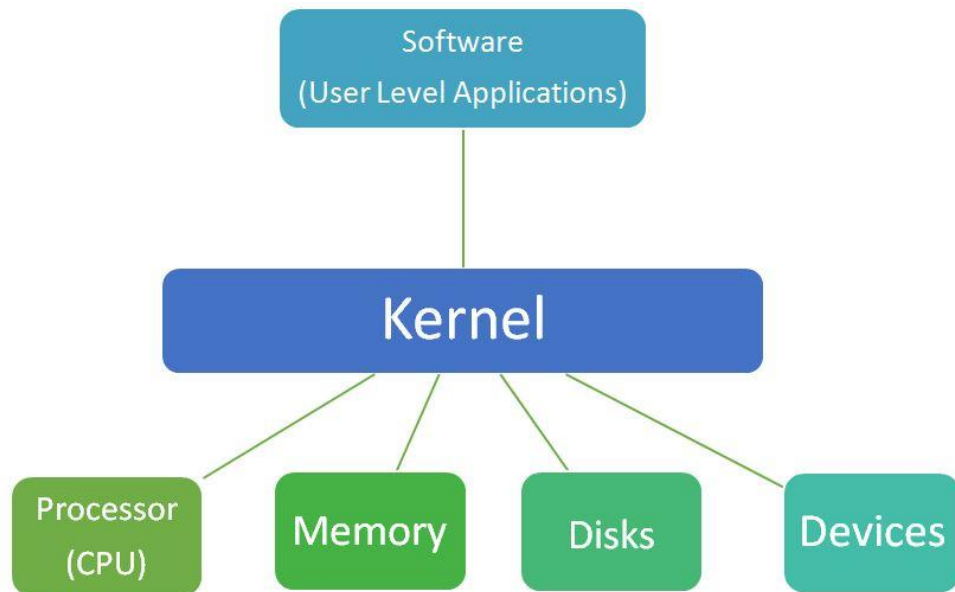


Ce este un kernel?



În engleză kernel înseamnă nucleu.

- Un OS are aplicații. (Browser, editor de text, media-player, etc..)
- Kernelul se ocupă să facă legătura cu părțile fizice.
- Kernelul ne pune la dispoziție cam orice funcție spre ajutorul nostru.
- Kernelul e deștept (cel puțin alea UNIX-based – garantat).
- Kernelul dispune de foarte multe funcții deștepte, dar mulți programatori nu știu asta (din păcate).





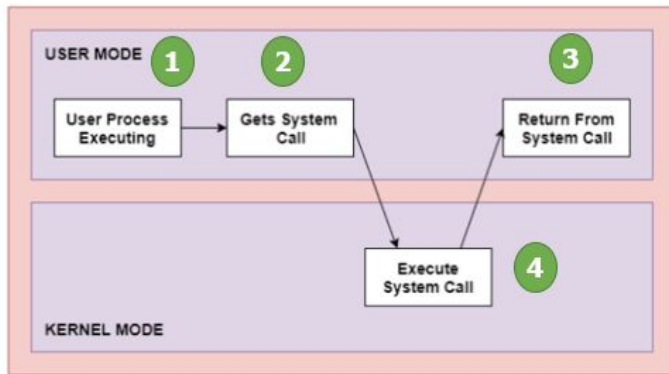
**Dacă kernelul știe să facă un lucru,
atunci programul tău nu îl poate face
mai bine sau mai eficient.**

**Concluzie - abuzați de ce știți să faceți
kernelul.**

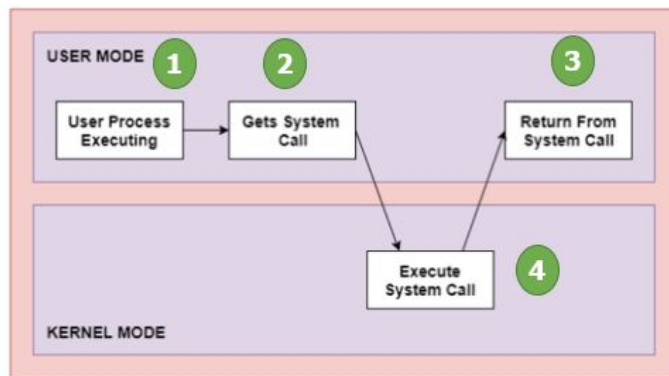


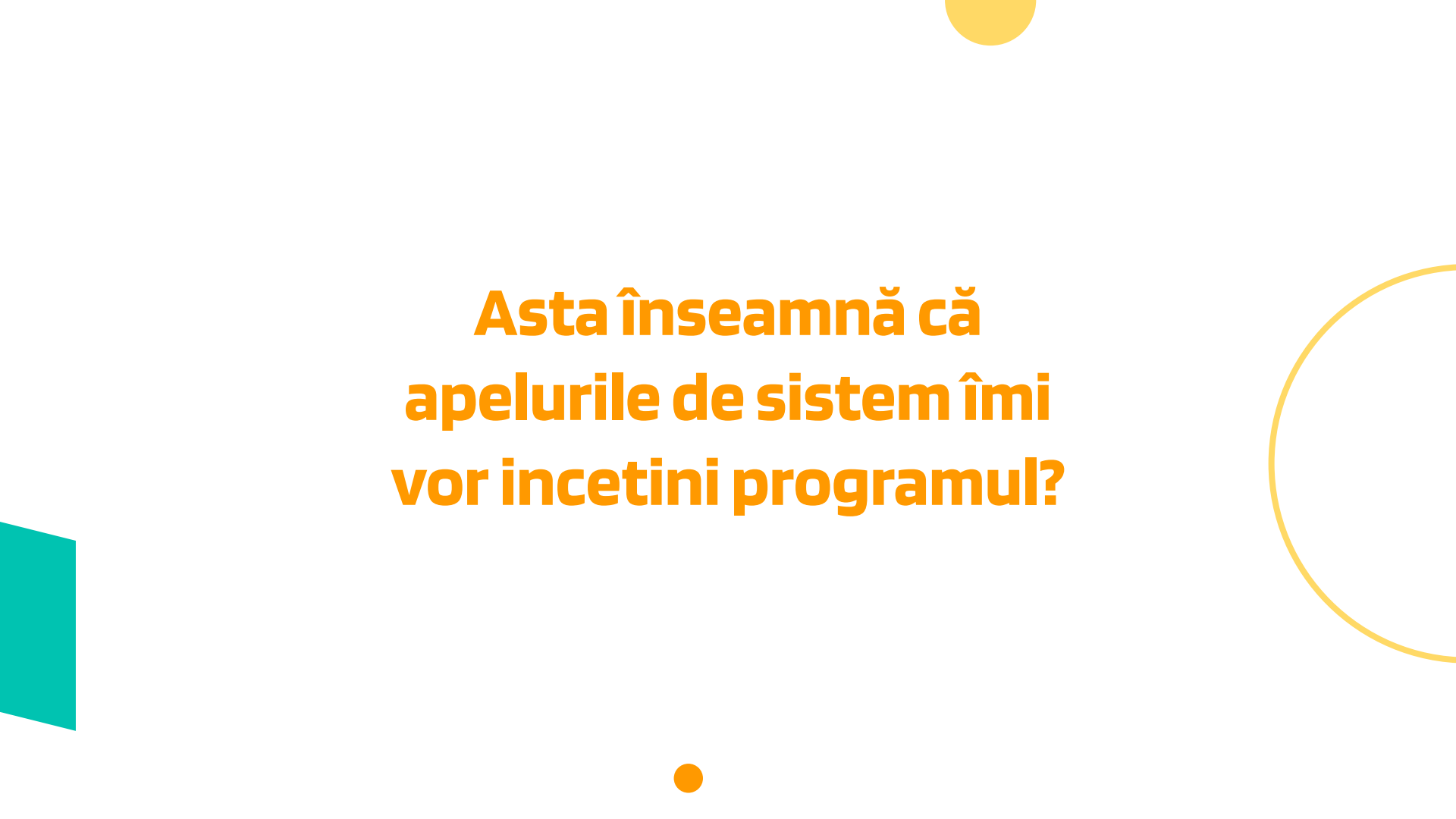
Ce este un apel de sistem?

User mode = unde programul rulează.
Când folosești un apel de sistem, intri în kernel
mode - execuția se reia când decide kernelul.

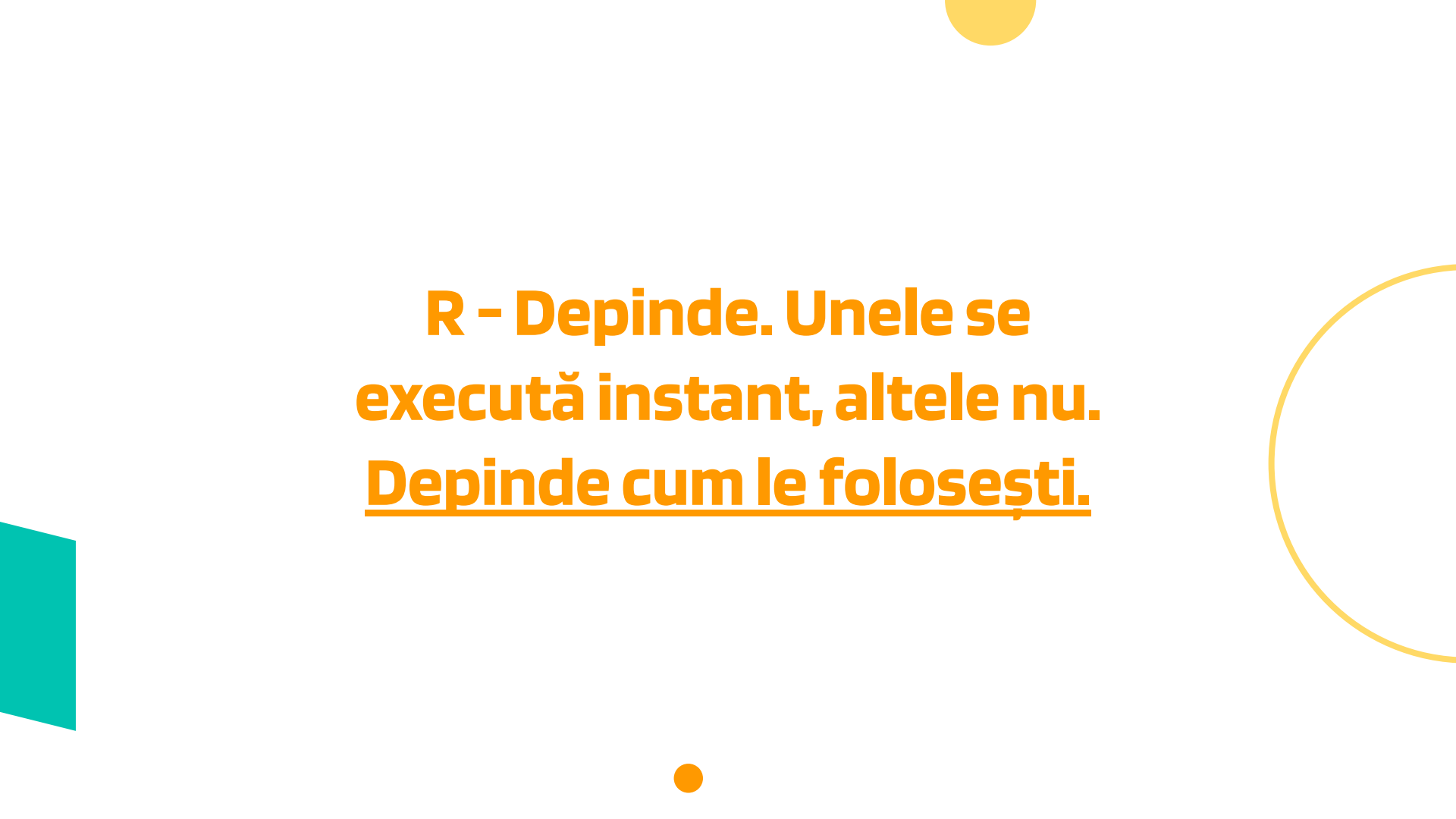


**Un apel de sistem declanșează un *context switch*.
Adică kernelul gestionează mai multe programe în
paralel. La context switch pierzi din prioritate
pentru că aștepți.**

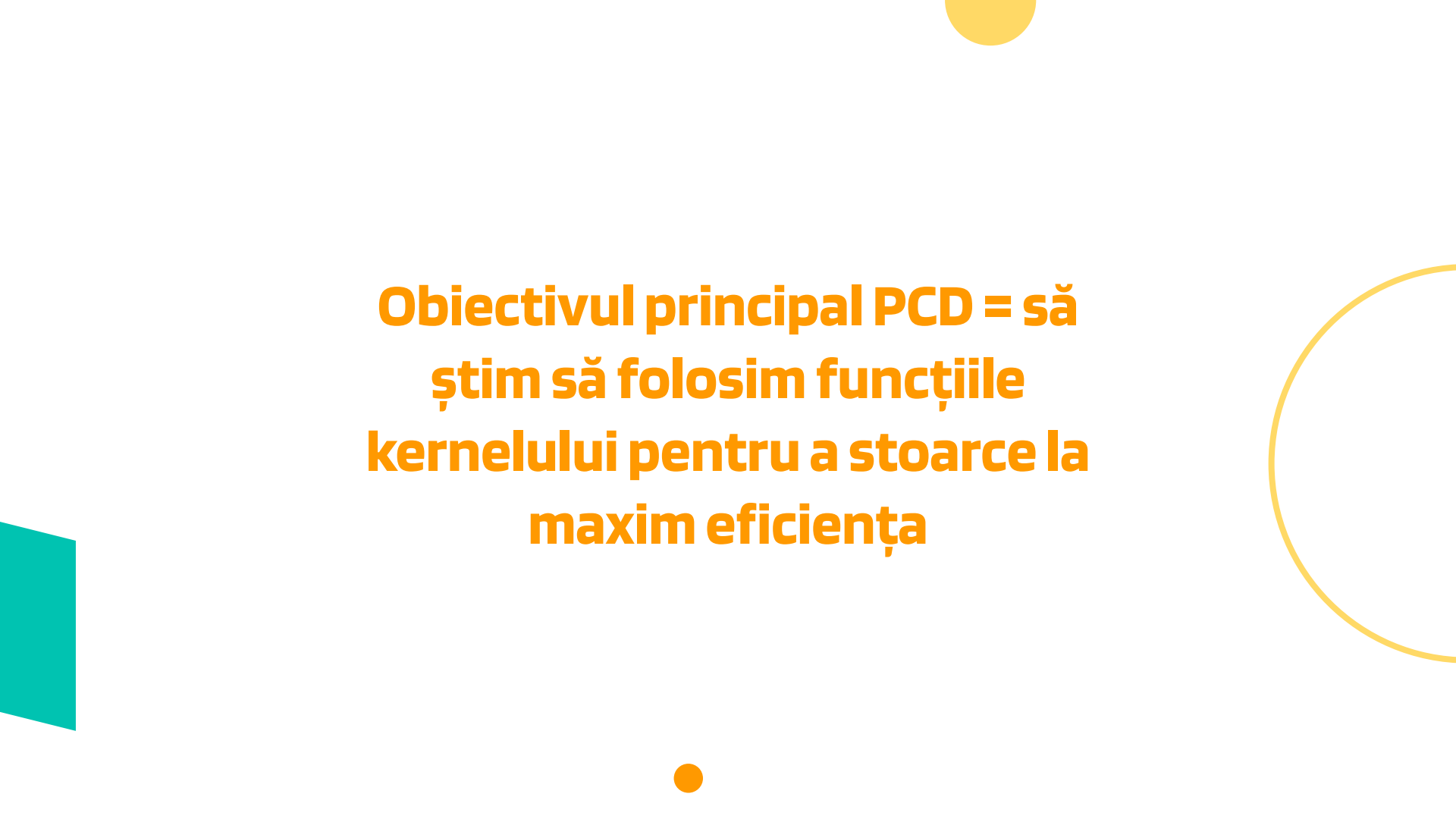




**Asta înseamnă că
apelurile de sistem îmi
vor încetini programul?**



**R - Depinde. Unele se
execută instant, altele nu.
Depinde cum le folosești.**



**Obiectivul principal PCD = să
știm să folosim funcțiile
kernelului pentru a stoarce la
maxim eficiența**



**Dacă eu nu vreau să fiu
programator de C/C++,
trebuie să știu astea?**

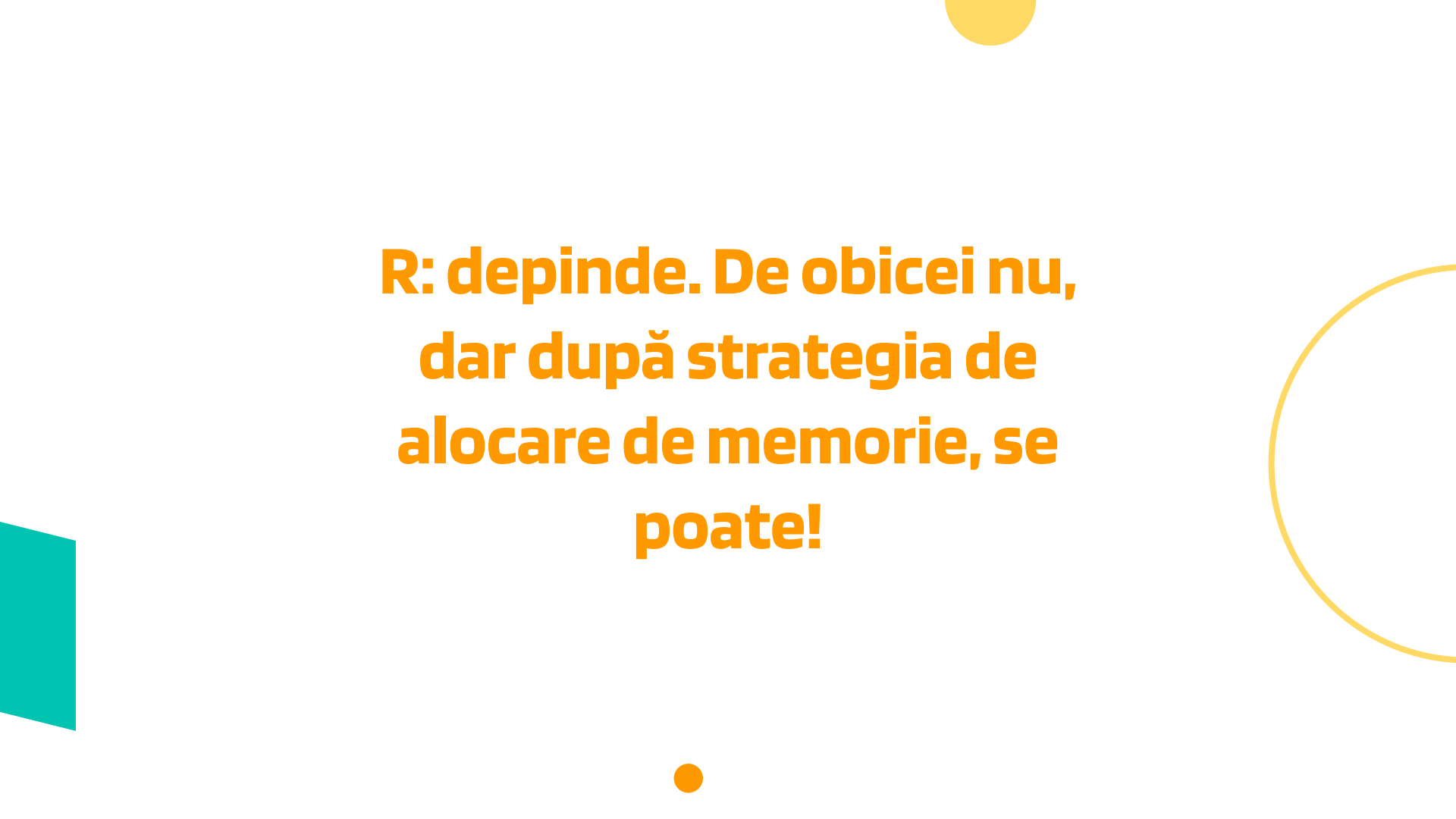


În mai multe limbaje avem:

Object x = new Object()

**Î: new() declanșează apel de
sistem?**





**R: depinde. De obicei nu,
dar după strategia de
alocare de memorie, se
poate!**

**Morala - apelurile de sistem
sunt în orice limbaj. Toate
limbajele populare (Java,
Python, Node JS, etc.) au
*bindings (legături) către
apelurile de C.***

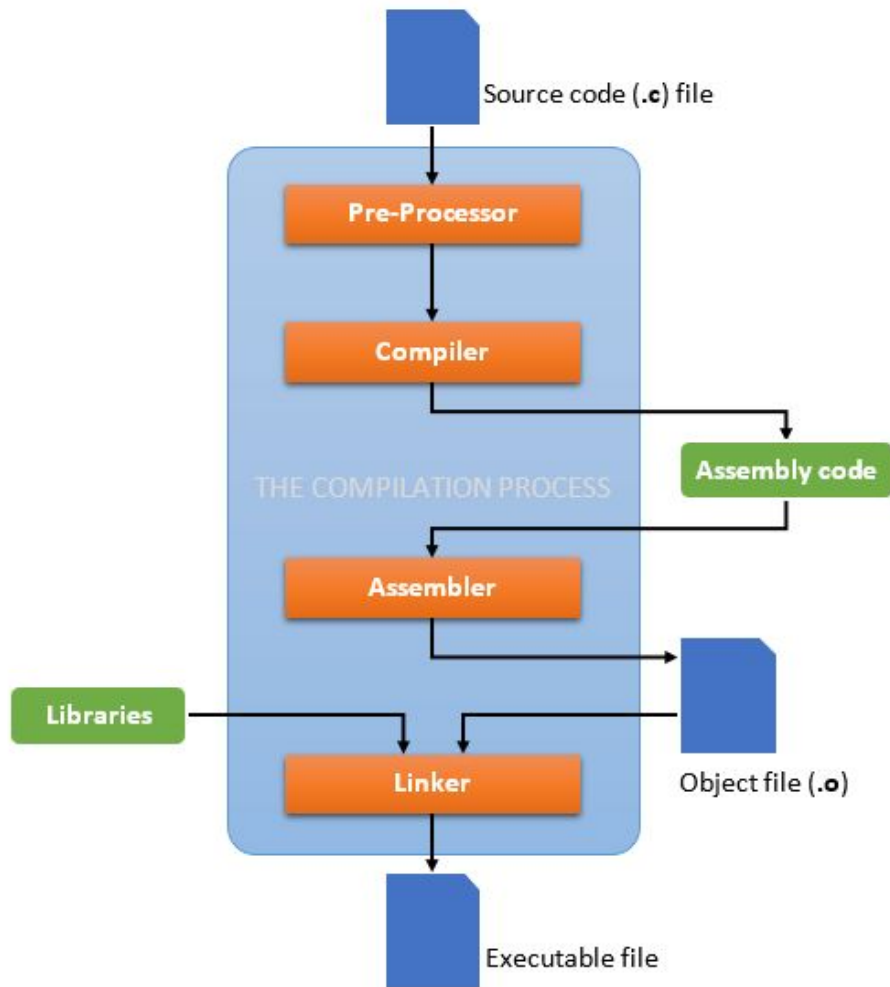
Exemple de apeluri de sistem

- `fork()`- crează un proces copil,
- `execl()`, `execle()`, `execlp()`, `execv()`, `execve()`, `execvp()`- execută un fișier
- `wait()`, `waitpid()`, `waitid()`- așteaptă ca un proces să-și schimbe starea
- `pipe()`- crează un pipe anonim
- `mkfifo()`- crează FIFO (pipe cu nume)
- `send()`, `sendto()`, `sendmsg()`- trimite un mesaj într-un socket
- `recv()`, `recvfrom()`, `recvmsg()`- citește un mesaj dintr-un socket
- `read()`- citește de la un descriptor de fișier (file descriptor)
- `write()`- scrie într-un descriptor de fișier (file descriptor)
- `getpid()`, `getppid()`- obține identificarea unui proces, etc



GCC

(GNU **Compiler** Collection)



Addition of two numbers

Assembly Language

```
.model small
.data
opr1 dw 1234h
opr2 dw 0002h
result dw 01 dup(?, '\$\'
.code
    mov ax, @data
    mov ds, ax
    mov ax, opr1
    mov bx, opr2
    clc
    add ax, bx
    mov di, offset result
    mov [di], ax
    mov ah, 09h
    mov dx, offset result
    int 21h
    mov ah, 4ch
    int 21h
end
```

C Program

```
#include <stdio.h>

int main()
{
    int a, b, sum;

    printf("\nEnter two no: ");
    scanf("%d %d", &a, &b);

    sum = a + b;

    printf("Sum : %d", sum);

    return(0);
}
```



```

/** Fisiier salut.c */
/** Starteaza de 3 ori o functie */
/** Fiecare functie forteaza o intarziere de 1,2, sau 3 sec. */
/**
    Timpul de real de executie a programului e in jur de 6 secunde
    indiferent de faptul ca calculatorul are mai multe procesoare
    sau procesorul are mai multe nuclee
*/
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
void* helsl(void *argument) {
    int *i = (int *)argument;
    int j = *i;
    fprintf(stderr, "Salut pe toata lumea %d - secunda/e\n", j);
    sleep(j); // i =1,2 sau 3 secunde la fiecare trecere
}

int main(int argc, char* argv[])
{
    int i;
    int j[3]= {1,2,3}; // valoare sleep pt. fiecare functie
    for (i=0; i <3; i++)
    {
        // startez de 3 ori functia
        helsl(&j[i]);
    }

    fprintf(stderr, "\n");
    exit(123); // cod retur pozitionat la 123
}

```



```
$ gcc -o salut salut.c <--- compilare/link-editare program
$ ./salut <--- startare program
Salut pe toata lumea 1 - secunda/e
Salut pe toata lumea 2 - secunda/e
Salut pe toata lumea 3 - secunda/e
$ echo $? <--- (123 -- cod retur)
123
$ time ./salut <--- startare prin comanda time pentru a afla real time
Salut pe toata lumea 1 - secunda/e
Salut pe toata lumea 2 - secunda/e
Salut pe toata lumea 3 - secunda/e

real 0m6.003s <--- real time cca 6 sec.
user 0m0.001s
sys 0m0.002s
```

Error Handling





```
int fd;

if ((fd = open("fis", O_RDONLY)) == -1)
    perror("nu pot sa deschid fisierul fis");
```

- În acest segment de program, **open()** returnează -1 atunci când eșuează.
- Eșecul apelului este testat și prin apelul funcției **perror()** care afișează mesajul ***nu pot sa deschid fisierul fis*** de eroare urmat de mesajul de eroare generat de apelul de sistem **open()** eșuat.
- Fd = File Descriptor. Ce este? Un id / identificator de fișier.



```
#include<string.h>
#include<errno.h>

int fd;

if (( fd = open("fis", O_RDONLY))==-1)
    fprintf(stderr, "nu pot sa deschid fisierul fis %s\n", strerror(errno));
```

- În exemplul de mai sus, eșecul apelului de sistem **open()** poziționează o variabilă globală **errno** la o anumită valoare ce reprezintă codul erorii.
- Semnificația codului de eroare este definită (prin intermediul unui macro) în **errno.h**. Apelul sistem **strerror(errno)** returnează un șir de caractere care indică natura precisă a erorii.



Argumente in linia de comandă

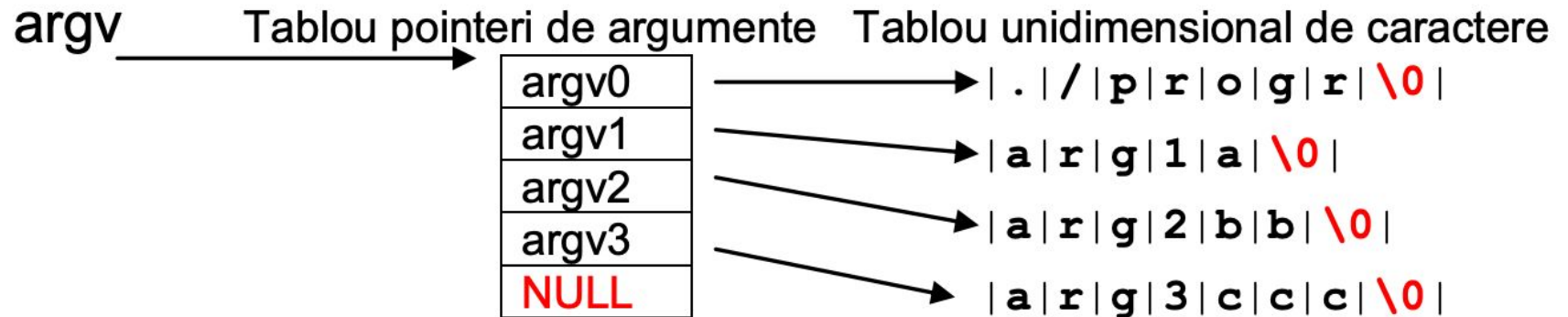


```
int main(int argc, char[][] argv)
```

- Poate fi **char****, **char[]***, **[]** si ***** sunt echivalente.
- **argv**, creat și transmis de shell programului, este un pointer către un tablou de pointeri (vector) de argumente terminat cu **NULL**. Fiecare element din vector este un pointer la rândul său către primul caracter al unui tablou unidimensional de caractere, care reprezintă valoarea argumentului de pe linia de comandă. Fiecare tablou unidimensional de caractere se termină cu '\0'. În fapt este vorba de un string (șir de caractere terminat implicit cu '\0').
- **argc** = argument count



```
$ ./progr arg1a arg2bb arg3ccc
```





```
$ ./progr arg1a arg2bb arg3ccc
```

Ce valoare are *argc*?



```
$ ./progr arg1a arg2bb arg3ccc
```

Aici, argc = 4

Exemple de programme


```

    /** Fisier salutt.c */
    /** Starteaza de 3 ori un thread */
    /** Fiecare thread forteaza o intarziere de 1,2, sau 3 sec. */
    /**
        Daca calculatorul are un procesor cu mai multe nuclee
        timpul real de executie a programului e in jur de 3 secunde.
        In cazul ca procesorul nu are mai multe nuclee atunci timpul
        real de executie a programului este in jur de 6 secunde
    */
    #include <stdbool.h>
    #include <stdio.h>
    #include <stdlib.h>
    #include <pthread.h>
    #include <unistd.h>
    void* helssl(void *argument) {
        int *i = (int *) (argument);
        int j = *i;
        fprintf(stderr, "Salut pe toata lumea %d - secunda/e\n", j);
        sleep (j); // i =1,2 sau 3 secunde la fiecare trecere
    }

    int main(int argc, char* argv[])
    {
        int i;
        pthread_t th[3]; // identificatori thread
        int j[3]= {1,2,3}; // valoare sleep pt. fiecare thread
        for (i=0; i <3; i++)
        {
            // startez de 3 ori thread-ul
            pthread_create(&th[i], NULL, helssl, &j[i]);
        }

        fprintf(stderr, "\n");
        pthread_exit(NULL); // astept terminarea thread-urilor
    }

```



```
$ gcc -o salutt salutt.c -lpthread
$ time ./salutt
Salut pe toata lumea 1 - secunda/e
Salut pe toata lumea 2 - secunda/e
Salut pe toata lumea 3 - secunda/e
real 0m3.004s <--- real time cca 3 sec.
user 0m0.000s
sys 0m0.004s
```



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

int main()
{
    // child process because return value zero
    if (fork() == 0) {
        printf("Hello from Child!\n");
    } else {
        // parent process because return value non-zero.
        printf("Hello from Parent!\n");
    }
    return 0;
}
```