Tutoriat 1 (Sisteme de Operare)
Introducere

Contents

1 Ce este un CPU?

Procesoarele sunt un termen general pentru orice componenta fizica a calculatorului care proceseaza date (CPUs, GPUs, sound processors, etc.).

Un **CPU (central processing unit)** este o componenta fizica a calculatorului, cu scopul de a face calcule si de a citi si interpreta instructiuni, controland restul componentelor. Un CPU trece printr-o serie de pasi pentru a putea executa instructiuni:

- **Fetch:** preia instructiunea din memoria sistemului.
- **Decode:** o interpreteaza.
- **Execute:** executa instructiunea, facand calcule si procesand date.
- **Write back:** stocheaza rezultatul intr-un registru sau in memoria principala.

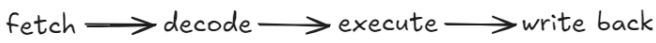


Figure 1: Functiile unui CPU

GHz (gigahertz) este o unitate de masura pentru viteza unui CPU. 1 GHz = 10^9 Hz, insemnand ca un procesor de 3.5GHz poate executa 3.5 miliarde de instructiuni intr-o secunda.

Un **CPU core (nucleu de procesare)** este o unitate de procesare in interiorul CPU-ului. Un nucleu poate sa indeplineasca o singura sarcina, iar existenta mai multor nuclee permite paralelizarea mai multor sarcini. Exista doua tipuri de CPU:

- **Single-core:** un singur nucleu de calcul. La un moment dat, se poate executa o singura operatie.
- **Multi-core (dual-core, quad-core, etc.):** cu doua sau mai multe nuclee. Se pot procesa mai multe operatii si date in paralel.

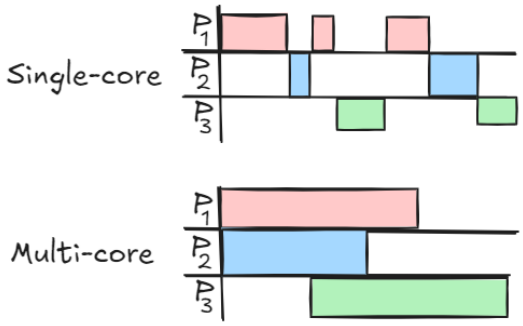


Figure 2: Single-core vs. multi-core processing

Procesoarele single-core proceseaza datele in mod **concurent:** sar de la o sarcina la alta intr-un mod aproape instant, oferind iluzia **paralelismului**. Procesoarele multi-core sunt capabile de a executa instructiuni in paralel cu adevarat, fiind mai rapide. Cand exista mai multe sarcini decat nuclee, se pune urmatoarea problema: ce sarcini ar trebui sa fie executate de catre CPU? Aceasta problema poarta numele de **CPU scheduling** si va fi studiata in detaliu la alt tutoriat.

2 Ce este un sistem de operare?

Un **sistem de operare** este un intermediar (o interfata) intre componentele fizice ale calculatorului (**hardware**) si utilizator. Un sistem de operare are in vedere gestionarea memoriei, utilizarea CPU-ului, spatiul de stocare, dispozitivele I/O, prevenirea erorilor si a utilizarii neadecvate a resurselor, etc., pentru ca utilizatorul sa poata rula aplicatii.

De obicei, un sistem de operare permite interactiunea cu utilizatorul prin intermediul unui **CLI (command-line interface**, e.g. Windows PowerShell), sau **GUI (graphical user interface)**.

Fiecare sistem de operare ofera utilizatorului un mediu de lucru diferit, cu diverse avantaje si dezavantaje. De exemplu, **Windows** este un sistem comercial, accesibil (user-friendly), bazat pe interfete grafice, in timp ce distributiile de **Linux** sunt axate in principal pe utilizarea terminalului (mai complicat). Ar fi de preferat ca un sistem de operare sa indeplineasca urmatoarele cerinte:

- **Confortul utilizatorului:** interactiunea cu sistemul ar trebui sa se produca intr-un mod accesibil.
- **Executia programelor:** furnizeaza un mediu potrivit pentru ca utilizatorul sa poata indeplini anumite sarcini.
- **Gestionarea resurselor (eficient):** se ocupa de CPU, memorie, spatiul de stocare si dispozitivele I/O. Evident, ar fi placut ca resursele sa fie utilizate in mod eficient, pentru a maximiza performanta sistemului.
- **Securitate:** trebuie asigurate integritatea si confidentialitatea datelor utilizatorului.
- **Fiabilitate:** sistemul trebuie sa continue sa opereze intr-un mod fluid si neintrerupt, in cazul aparitiei unor erori sau exceptii.

Distingem intre doua tipuri de programe:

- **System programs (programe de sistem):** software care gestioneaza hardware-ul calculatorului, cum ar fi sistemul de operare (Windows, macOS, Linux), drivers (audio drivers, touchpad drivers, etc.), compilatoare (care traduc cod high-level in cod masina), si altele.
- **User programs (programele utilizatorului):** notepad, jocuri video, aplicatii de editare video, etc.

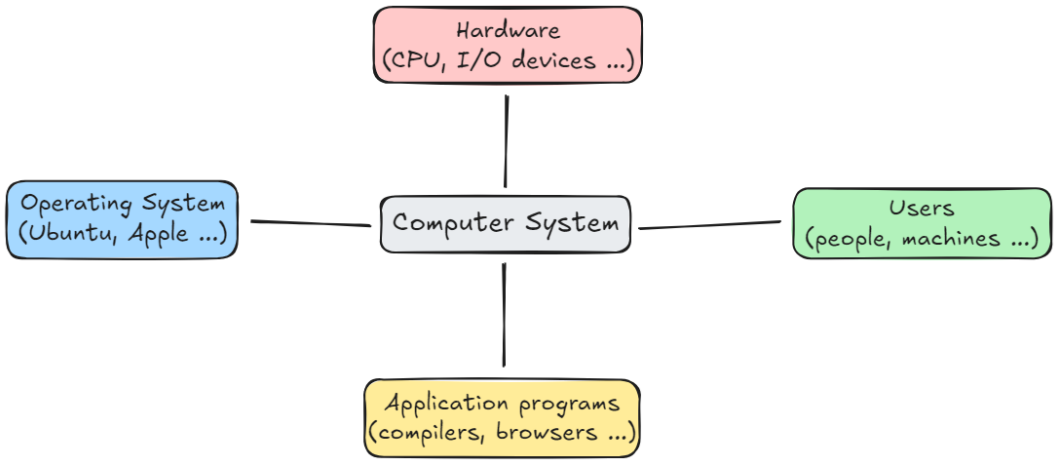


Figure 3: Structura unui calculator / sistem

Structura unui calculator poate fi impartita in patru componente:

- **Hardware:** CPU, memorie, dispozitive I/O, etc.
- **Sistemul de operare:** Ubuntu, Apple, etc.
- **Programe:** de sistem sau de utilizator.
- **Utilizatori:** pot fi persoane reale sau alte calculatoare/masini.

Kernel-ul este un program care face parte din sistemul de operare si are *control complet* asupra intregului calculator (poate gestiona hardware-ul, memoria, alocarea resurselor, procesele, apelurile de sistem si driverele). Din momentul in care porneste calculatorul, se porneste si kernel-ul (si ruleaza permanent).

Exista diverse abordari pentru construirea unui kernel. De exemplu, **structura monolitica (monolithic structure)** include toate functionalitatile kernel-ului intr-un singur fisier binar static, ce ruleaza in acelasi spatiu de

memorie. Se obtine un avantaj de performanta (totul ruleaza in acelasi spatiu), dar e dificil de implementat si de extins un asemenea sistem; de asemenea, modificarile intr-o bucata a sistemului pot avea efecte extinse asupra altor parti.

O proiectare mai modulara presupune aranjarea pe straturi (**layered approach**). Acestea sunt selectate astfel incat un strat se poate folosi numai de functiile straturilor de dedesubtul lui. Problema este definirea clara a functionalitatilor fiecarui strat; de asemenea, se obtine o performanta mai lenta, deoarece un program de utilizator care are nevoie de un serviciu furnizat de kernel trebuie sa astepte traversarea mai multor straturi.

Un **microkernel** este un program cat mai mic, ce ofera doar serviciile cele mai esentiale, cum ar fi gestionarea proceselor sau a memoriei. Restul serviciilor sunt implementate ca si programe de utilizator ce ruleaza in afara kernel-ului. Astfel, extinderea unui asemenea sistem devine o sarcina mai simpla; in plus, cum majoritatea serviciilor vor rula ca si programe de utilizator, de obicei prabusirea unui serviciu nu va afecta restul sistemului, oferind protectie si fiabilitate.

O alta posibilitate de implementare ar fi cu **loadable kernel modules (LKMs)**. Kernel-ul are niste servicii de baza, iar serviciile aditionale pot fi obtinute dinamic prin legatura cu module (cand se porneste calculatorul sau in timp ce deja ruleaza). Este o abordare mai adecvata, in comparatie cu implementarea functionalitatilor respective si recompilarea kernel-ului de fiecare data.

In concluzie, sunt putine sisteme care se tin strict de o singura structura specifica; in practica, sunt combinate mai multe structuri, rezultand intr-un **sistem hibrid**.

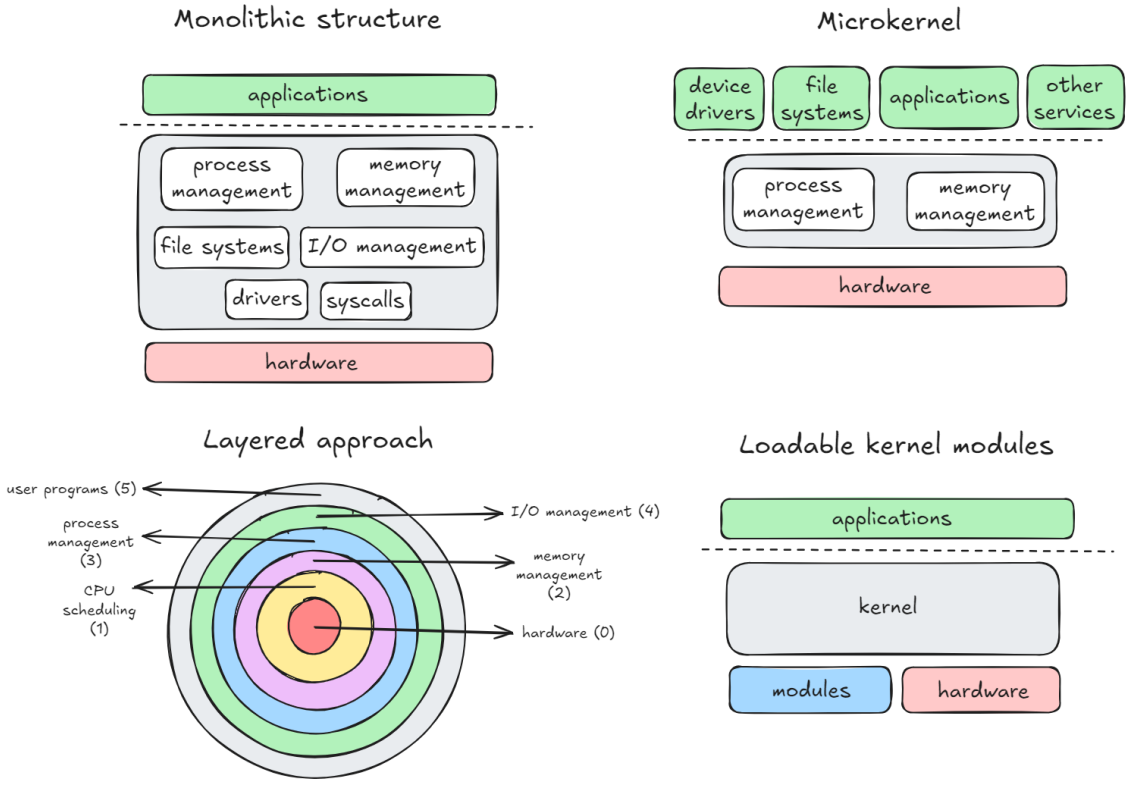


Figure 4: Diferite implementari pentru kernel

3 Linker si loader

Pentru a rula un program, sunt necesare producerea fisierului executabil asociat si incarcarea acestuia in RAM. Un **linker** este un program ce combina fisierele generate de catre compilator impreuna cu bibliotecile necesare intr-un singur fisier executabil. Un **loader** este un program ce aloca memorie pentru programe si le incarca in RAM.

Linker-ul indeplineste urmatoarele sarcini:

- **Combinarea fisierelor:** imbina fisierele compilate in format obiect (.o) cu bibliotecile necesare, pentru a forma un singur fisier.
- **Asignarea adreselor de memorie:** se ocupa de dispunerea memoriei pentru sectiunile de cod si text.
- **Rezolvarea referintelor:** de exemplu, intr-un fisier se produce un apel la o functie care este definita in alt fisier.
- **Optimizari de cod:** mai face mici optimizari ca sa reduca dimensiunea fisierului si sa fie mai rapid.

Un loader face urmatoarele lucruri:

- **Incarcarea fisierului:** il incarca in RAM.
- **Alocarea memoriei:** aloca spatiu pentru program.
- **Legare dinamica:** bibliotecile externe sau modulele necesare sunt legate de program la runtime.

4 Intreruperi si exceptii

O **intrerupere** este un semnal generat de catre un dispozitiv hardware sau o aplicatie software, pentru a notifica CPU-ul de aparitia unui eveniment.

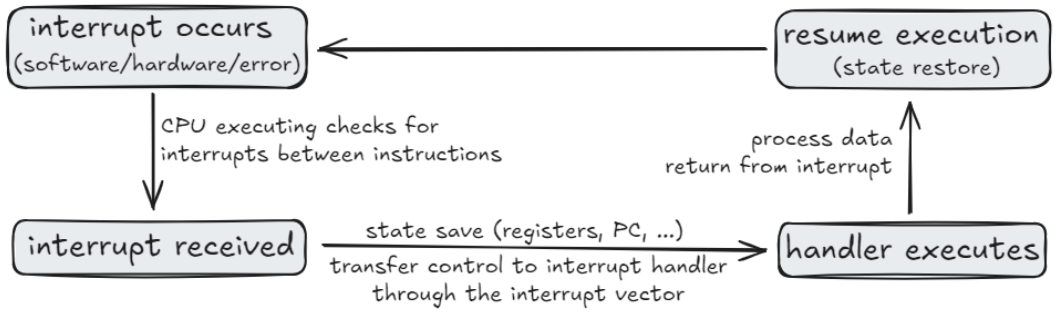


Figure 5: Gestionarea unei semnal de intrerupere

Intreruperile pot fi impartite in doua categorii:

- **Hardware:** generate de catre dispozitive externe precum tastatura, mouse, etc.
- **Software:** se mai numesc si **traps**. Sunt generate de catre aplicatii software in mod intentionat (e.g. are loc un apel de sistem) sau neintentionat (are loc o eroare, e.g. impartirea la zero).

Modul in care sunt gestionate intreruperile este urmatorul:

- Este primit semnalul, CPU-ul finalizeaza executia instructiunii curente si se realizeaza un **state save** (se salveaza valorile registrilor si **program counter-ul** - un registru special ce retine adresa urmatoarei instructiuni de executat).
- Se determina codul intreruperii; exista un **interrupt vector**, ce contine o rutina specifica pentru fiecare intrerupere in parte. Codul este folosit pentru indexare in vector.
- Se ruleaza rutina respectiva; odata ce s-a finalizat, se realizeaza un **state restore** (se restaureaza valorile vechi ale registrilor si program counter-ul), iar executia sarcinii intrerupte continua.

5 Ierarhia memoriei intr-un sistem de operare

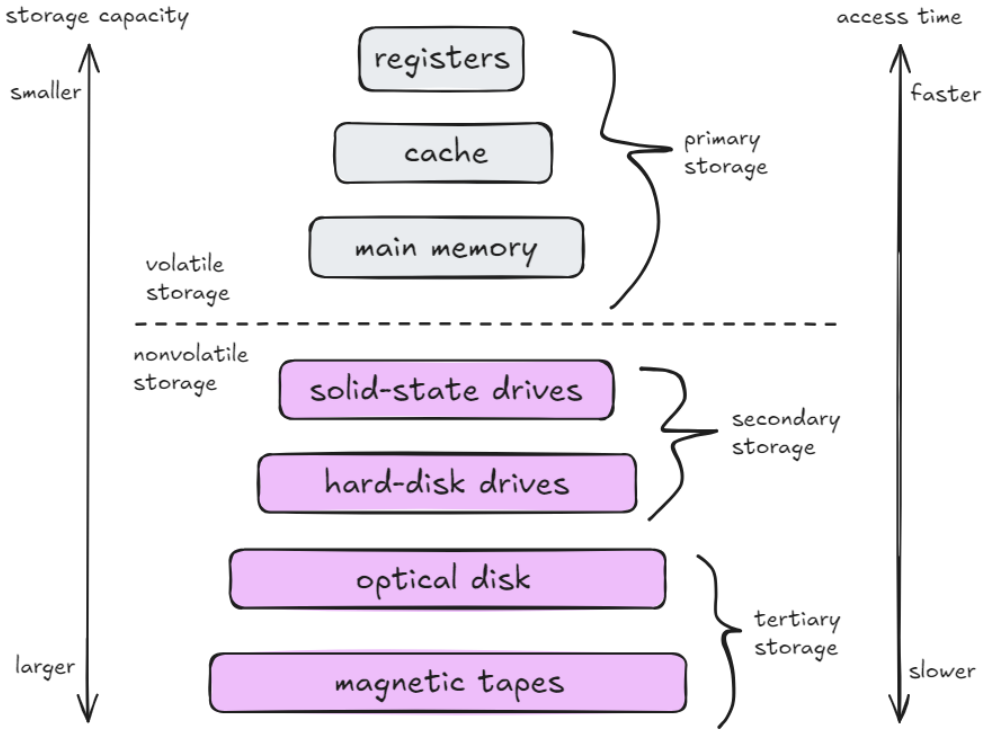


Figure 6: Ierarhia memoriei

Memoria poate fi impartita in doua categorii - **volatila** si **nonvolatila**. *Ce inseamna memorie volatila?* Cand se opreste calculatorul, toate informatiile din memoria respectiva sunt permanent pierdute; este o memorie pe termen scurt. **Memoria nonvolatila (NVM = non-volatile memory)** este un tip de memorie ce retine date pe termen lung; acestea **nu** dispar odata cu inchiderea calculatorului. Sistemul de operare in sine este pastrat in memorie nonvolatila si incarcat in RAM odata cu pornirea calculatorului.

5.1 Memorie volatila

Cu cat este mai mica dimensiunea memoriei, cu atat accesarea acesteia are loc mai rapid. **Registrii** sunt cea mai mica si rapida forma de memorie, situati in CPU si utilizati de catre acesta pentru a stoca temporar date, precum adrese de memorie si rezultatele la diverse operatii aritmetice.

Anumite date sunt accesate in mod frecvent, cum ar fi paginile web sau rezultatul la o anumita operatie aritmetica. O modalitate de optimizare a performantei unui sistem este stocarea resurselor unei pagini web sau a rezultatului unei operatii aritmetice intr-un **cache**, care este o memorie cu timpi rapizi de acces, ce contine copii ale datelor accesate in mod regulat.

Un **cache hit** are loc atunci cand se verifica cache-ul pentru o anumita informatie si aceasta este gasita; **cache miss** este opusul (se verifica, dar informatia respectiva lipseste). Performanta unui sistem incetineste cand au loc prea multe cache miss-uri.

Memoria principala este **RAM (random access memory)**, folosita pentru stocarea si accesarea rapida a datelor utilizate de catre CPU. In RAM sunt incarcate sistemul de operare, aplicatiile ce ruleaza si datele folosite de catre aplicatiile respective. *Ce inseamna random access?* Informatiile nu sunt stocate neaparat intr-o anumite ordine; datele pot fi stocate si accesate la orice locatie in orice moment.

Numarul de GB de RAM este o specificatie ce afecteaza in mod major performanta unui calculator. Cu cat ai mai mult RAM, cu atat exista mai mult spatiu pentru mai multe aplicatii (sau pentru aplicatii mai consumatoare de resurse).

RAM-ul poate fi implementat in mai multe feluri, cum ar fi **DRAM (dynamic random access memory)**, sau **SRAM (static random access memory)**.

5.2 Memorie nonvolatila

Memorie nonvolatila exista in mai multe forme, precum **ROM (read-only memory**; stocheaza **firmware** = software incorporat in memoria dispozitivelor hardware), **flash memory** (utilizata de catre **SSDs** = solid-state drives, **USBs** = universal serial bus, etc.), **magnetic storage** (pentru **HDDs** = hard-disk drives) si **EEPROM (electrically erasable programmable read-only memory**; datele pot fi sterse si rescrise).

Memoria secundara (secondary storage) este o aplicatie a conceptului de memorie non-volatila si reprezinta o categorie de dispozitive folosite pentru stocarea pe termen lung a datelor. Niste exemple de memorie secundara sunt HDDs, SSDs si USBs.

6 Apeluri de system (syscalls)

NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)
0	read	man/ cs/	0x00	unsigned int fd	char *buf	size_t count
1	write	man/ cs/	0x01	unsigned int fd	const char *buf	size_t count
2	open	man/ cs/	0x02	const char *filename	int flags	umode_t mode
3	close	man/ cs/	0x03	unsigned int fd	-	-
4	stat	man/ cs/	0x04	const char *filename	struct __old_kernel_stat *statbuf	-
5	fstat	man/ cs/	0x05	unsigned int fd	struct __old_kernel_stat *statbuf	-
6	lstat	man/ cs/	0x06	const char *filename	struct __old_kernel_stat *statbuf	-
7	poll	man/ cs/	0x07	struct pollfd *ufds	unsigned int nfds	int timeout
8	lseek	man/ cs/	0x08	unsigned int fd	off_t offset	unsigned int whence

Figure 7: Cateva apeluri de sistem de pe Linux

Procesorul unui calculator poate rula in cel putin doua moduri: **kernel mode**, unde acesta poate gestiona fara restrictii tot ce tine de resursele calculatorului (e.g. componente hardware si memorie), si **user mode**, care impune restrictii asupra anumitor operatii sensibile, pentru a imbunatati securitatea si a preveni prabusirea sistemului.

Reamintim ca programul care ruleaza permanent pe calculator si are *control absolut* asupra intregului sistem se numeste **kernel**, fiind *esenta sistemului de operare*. Astfel, cand un procesor ruleaza in *kernel mode*, acesta poate apela functiile kernel-ului, avand privilegii maxime. De regula, aplicatiile ruleaza in user mode (e.g. notepad nu are nevoie sa iti gestioneze memoria calculatorului sau hardware-ul).

Eroarea sau crash-ul unei aplicatii in user mode de obicei afecteaza numai aplicatia respectiva, in timp ce un crash in kernel mode poate duce la esecul intregului sistem. Felul prin care calculatorul distinge intre user si kernel mode este prin intermediul valorii unui bit, numit **mode bit**, furnizat de catre hardware.

Trecerea unui utilizator in kernel mode se face prin intermediul unui apel de sistem; cand apelul respectiv se incheie, mode-ul este resetat inapoi la user mode.

Ce este un apel de sistem? Un **syscall** este un apel catre o instructiune/functie speciala - o cerere catre kernel de a permite CPU-ului sa ruleze in kernel mode, pentru a putea indeplini sarcina dorita. Exista diverse servicii indeplinite de catre apelurile de sistem, cum ar fi *operatii pe fisiere* (creare, deschidere, citire, scriere), gestionarea proceselor (concept studiat in urmatorul tutoriat), gestionarea memoriei (alocare, dealocare), etc.

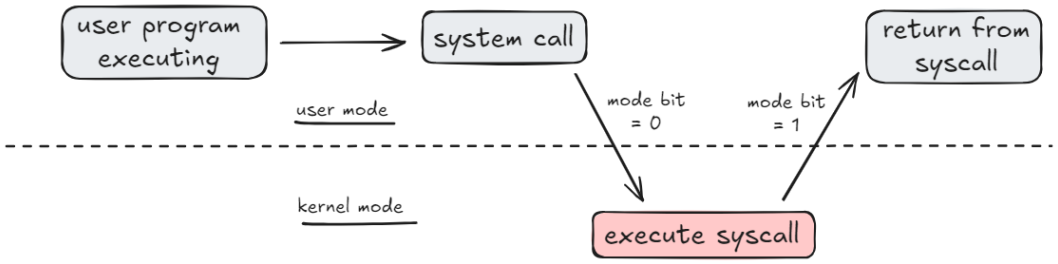
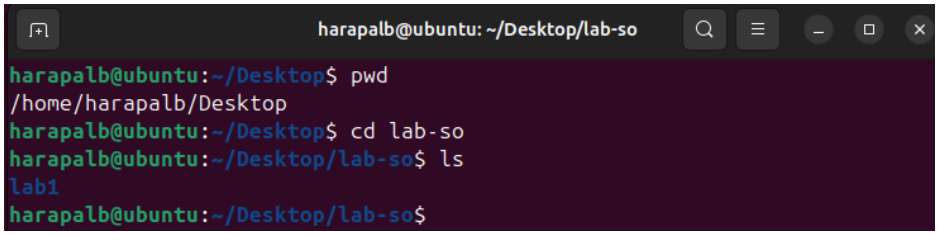


Figure 8: Tranzitia din user mode in kernel mode si inapoi

7 Introducere in utilizarea terminalului pe Linux

Terminalul de pe Linux este o interfata text, bazata pe utilizarea comenzilor pentru a interactiona cu sistemul. Este utilizat frecvent din motive de viteza, de control mai precis asupra sistemului, posibilitatea automatizarii anumitor procese si din cauza limitarilor GUI-urilor.



In terminal apar anumite simboluri inaintea fiecarei comenzi. In imaginea de mai sus, distingem urmatoarele elemente: numele utilizatorului (*harapalb*), numele calculatorului (*ubuntu*), directorul in care ne aflam ($\sim$ fiind prescurtarea de la */home/your_username*) si modul de lucru (*$* pentru user mode, *#* pentru admin/root mode). Impreuna, aceste elemente compun urmatoarea structura:

```
[username]@[hostname]:[path][mode]
```

7.1 Comenzi pentru analizarea executiei fisierelor

Consideram urmatorul program *test.c*:

```
#include <stdio.h>

int main() {
    char text[] = "un string";
    printf("%s\n", text);
    return 0;
}
```

Compilarea programului are loc astfel:

```
gcc test.c
```

Astfel, fisierul sursa *test.c* este compilat folosind **GCC (GNU Compiler Collection)**. Cum nu am specificat un nume pentru fisierul binar rezultat, acesta implicit va fi numit *a.out*. Pentru a rula executabilul respectiv, introducem urmatoarea comanda:

```
./a.out
```

Simbolul . reprezinta directorul curent, iar *./* este folosit pentru a executa un fisier specific din directorul curent. In cazul in care am fi dorit ca executabilul sa aiba alt nume, am fi putut include in comanda optiunea *-o*, ce permite redenumirea fisierului; de exemplu:

```
gcc test.c -o test
```

Acum putem rula fisierul astfel:

```
./test
```

Toate comenzile prezentate mai sus rulate in terminal ar arata astfel:

```
harapalb@ubuntu:~/Desktop/lab-so/lab1$ gcc test.c -o test
harapalb@ubuntu:~/Desktop/lab-so/lab1$ ./test
un string
harapalb@ubuntu:~/Desktop/lab-so/lab1$ gcc test.c
harapalb@ubuntu:~/Desktop/lab-so/lab1$ ./a.out
un string
harapalb@ubuntu:~/Desktop/lab-so/lab1$
```

7.1.1 Listarea apelurilor de sistem

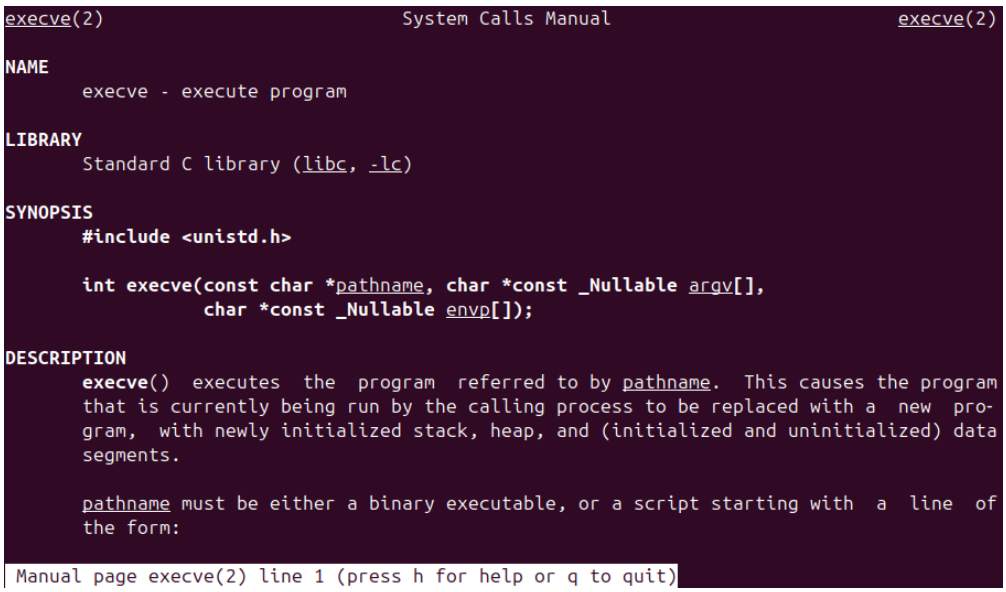
Fisierele binare executate au nevoie de anumite apeluri de sistem (syscalls) pentru a isi indeplini sarcinile. Listarea apelurilor folosite se poate realiza cu ajutorul comenzii **strace (system call trace)**; astfel, *strace ./test* executa fisierul *test* si afiseaza syscall-urile utilizate.

```
harapalb@ubuntu:~/Desktop/lab-so/lab1$ strace ./test
execve("./test", ["./test"], 0x7ffe91e047e0 /* 49 vars */) = 0
brk(NULL)                               = 0x59c411fc0000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x722b70a9a000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=59911, ...}) = 0
mmap(NULL, 59911, PROT_READ, MAP_PRIVATE, 3, 0) = 0x722b70a8b000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x722b70800000
mmap(0x722b70828000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x
28000) = 0x722b70828000
```

Randurile din output-ul de mai sus urmeaza structura:

```
syscall(arguments...) = return_value
```

Cateva din syscall-urile de mai sus care vor fi studiate ar fi *execve*, *fstat* si *mmap*. Apelurile de sistem identificate pot fi ulterior analizate cu ajutorul comenzii *man*. De exemplu, daca dorim sa aflam ce face *execve*, rulam **man 2 execve** in terminal si obtinem:



7.1.2 Bibliotecile utilizate

Biblioteca *glibc (GNU C Library)* furnizeaza multe functii standard, precum *printf* si *malloc*. Un program ce utilizeaza asemenea functii va avea nevoie de aceasta biblioteca (si posibil altele). Inspectarea bibliotecilor utilizate de catre un program se poate face cu ajutorul comenzii *ldd (list dynamic dependencies)*:

```
harapalb@ubuntu:~/Desktop/lab-so/lab1$ ldd ./test
        linux-vdso.so.1 (0x00007fff8e785000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fac0dc00000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fac0df04000)
```

Pe fiecare rand este afisat numele unei biblioteci, iar valoarea hexazecimala din paranteza reprezinta adresa de memorie virtuala asociata bibliotecii respective in spatiul de adresa al procesului (aceste concepte vor fi studiate in detaliu mai tarziu). Prima biblioteca este **Linux vDSO (virtual dynamic shared object)**, utilizata pentru a face apelurile de sistem mai rapide. A doua este **glibc**. Ultima biblioteca este un linker si loader dinamic, ce incarca bibliotecile necesare programului si le rezolva dependentele.