Fonaments dels Computadors

L'Entorn i l'Execució de Programes (2/2)

Grau en Intel·ligència Artificial

Xavier Martorell

Xavi Verdú

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC)

Creative Commons License

Aquest document utilitza Creative Commons Attribution 3.0 Unported License

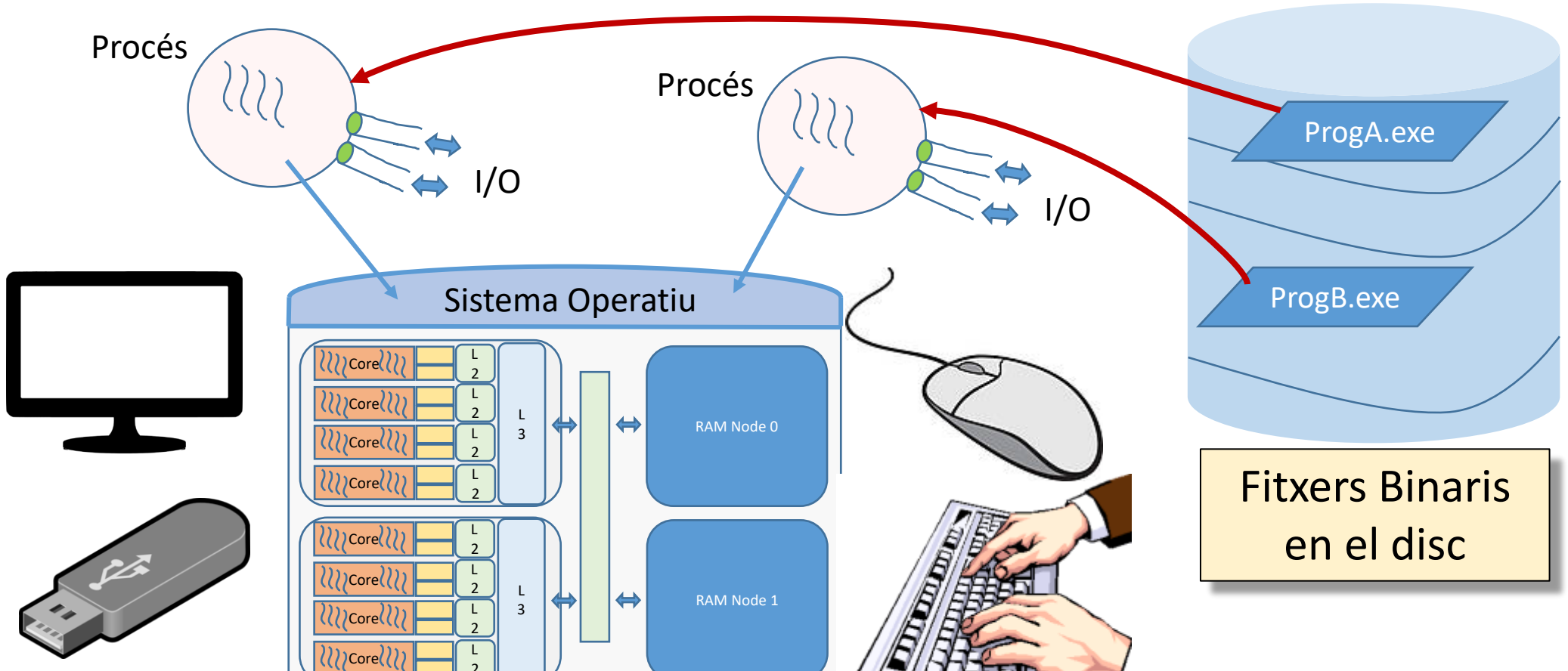
Els detalls d'aquesta licencia es poden trovar a
https://creativecommons.org/licenses/by-nc-nd/4.0

Index

- Generació d’executables
- **Components Bàsics del SO**
- **Impacte del SO en el rendiment i en el consum d’energia**

Què és un Sistema Operatiu?

- El SO és un Sw que gestiona els recursos Hw i Sw

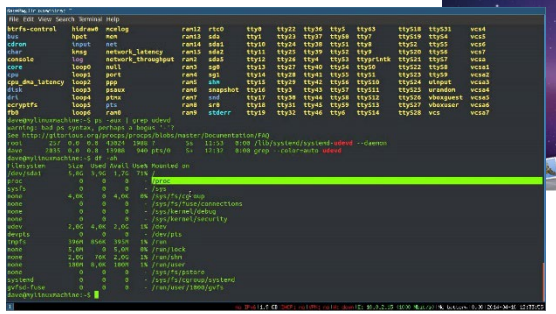


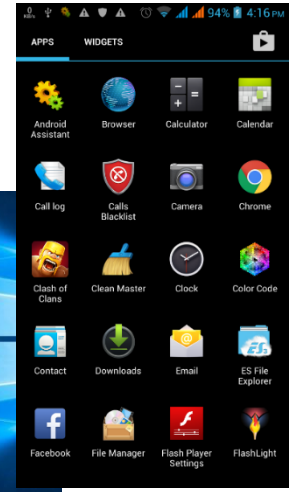
Principis de disseny del SO

- El SO juga un paper clau per l'interacció de l'usuari
  - Ofereix un entorn **usable**
    - Abstraeix a l'usuari de les particularitats de diferents "sistemes" i Hw
  - Ofereix un entorn d'execució **segur/robust/protegit**
    - Segur des del punt de vista d'accedir correctament al Hw
    - Protegit des del punt de vista de l'interacció de l'usuari
  - Ofereix un entorn d'execució **eficient**
    - Gestió del Hw de manera precisa
    - Permet molts usuaris/programes compartir recursos, garantint la seva correcte utilització

Gestió del Sistema

- El SO és l'intermediari entre aplicacions i Hw, diferenciant dues parts
  - **Internals del SO:** defineix estructures de dades per gestionar les aplicacions I el Hw, així com algorismes per decidir com fer-ho
    - El **Kernel** és la part fonamental (explicat en properes transparencies…)
  - **API del SO:** ofereix un conjunt de funcions per demanar **serveis del sistema**
    - **Crides a Sistema** (explicat en properes transparencies…)
- Diferentes visions gràfiques
  - Simplifiquen l'ús de l'entorn





Etapes genèriques en la gestió del SO

Arrancar

- Quan el sistema arrenca, el codi del SO es carrega en memòria
- S'inicialitcen configuracions bàsiques del Hw i d'accés al SO (e.g. interrupcions)
- Arrenquen mecanismes d'accés, interacció i gestió: daemons, login, shell, etc

Utilització

- Sessions d'usuari, execució d'aplicacions, serveis…
- Suport al desenvolupament de codi…

Apagar

- Quan s'apaga el sistema, el SO finalitza els programes i serveis que estan en funcionament
- Guarda informació persistent, para dispositius, etc

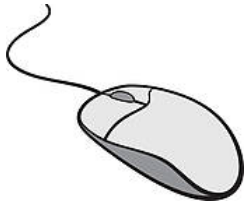
Accés a funcions del SO

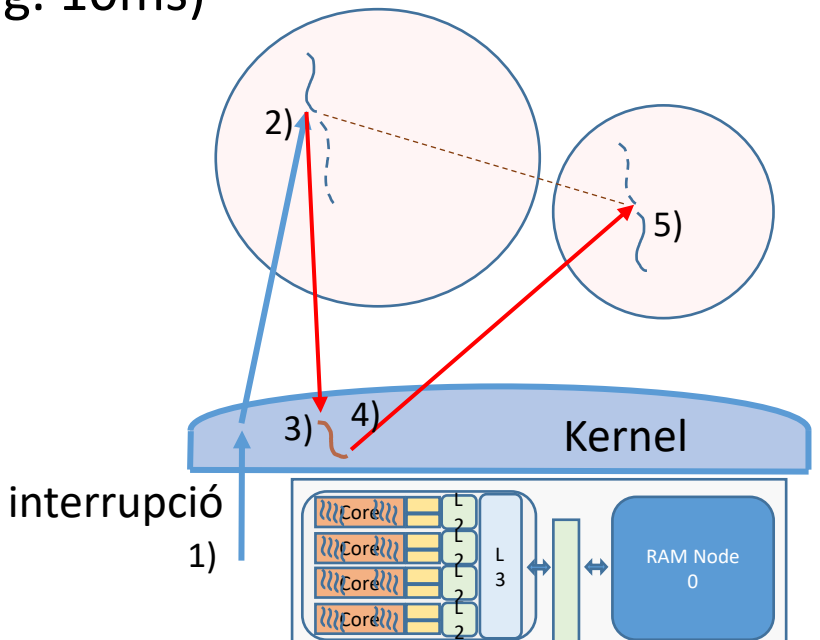
- **Modes d'execució**
  - Suport del hw per garantir seguretat
    - La CPU ha de ser capaç de diferenciar quan està executant instruccions que venen del codi de l'usuari (no-privilegiat) o del codi del SO (privilegiat)
  - Dos o més nivells de privilegi
    - Mode usuari vs. Privilegiat (a.k.a. kernel o sistema)
    - Exemple: 0 (més privilegiat) – 1 – 2 – 3 (menys privilegiat)

- El **Kernel** és la part crítica (el nucli) del SO. S'executa en el mode amb més privilegis perquè ha d'accedir a parts del SO
  - e.g., Linux kernel, Windows kernel, MacOS kernel

Quan s'executa el codi del Kernel? (I de III)

- Quan succeeix una **interrupció**:
  - Les interrupcions són generades per dispositius Hw
    - Des del punt de vista d'execució d'instruccions:
      - Esdevenimentent involuntari i asíncron
  - Interrupcions de rellotge: hi han varis rellotges en el sistema
    - **Tick del SO**: configurat durant l'arrancada pel SO per determinar la freqüència d'executar codi de gestió (e.g. 10ms)
  - Storage
  - Network
  - Entrada/Sortida
    - USB
    - Teclat
    - Ratolí

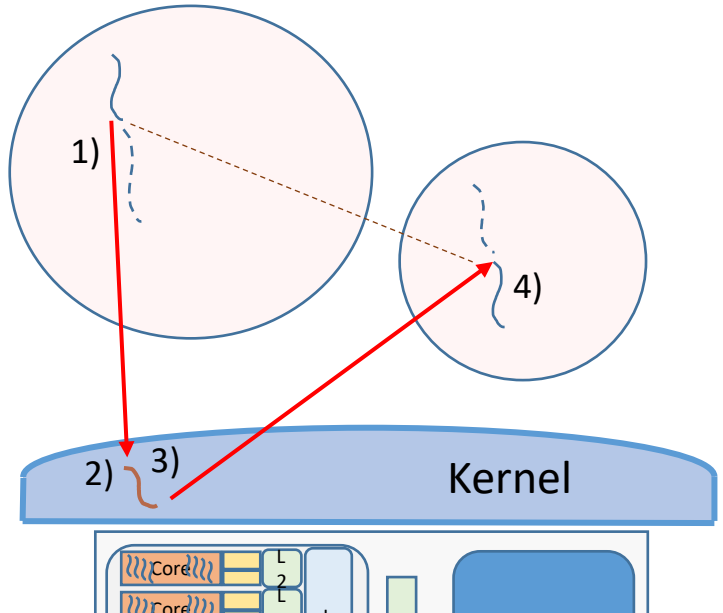




Quan s'executa el codi del Kernel? (II de III)

- Quan succeeix una **excepció**:
  - Les excepcions són generades per la CPU quan hi ha algun problema durant l'execució d'instruccions
    - Des del punt de vista d'execució d'instruccions:
      - Esdeveniment involuntari, però síncron

excepció



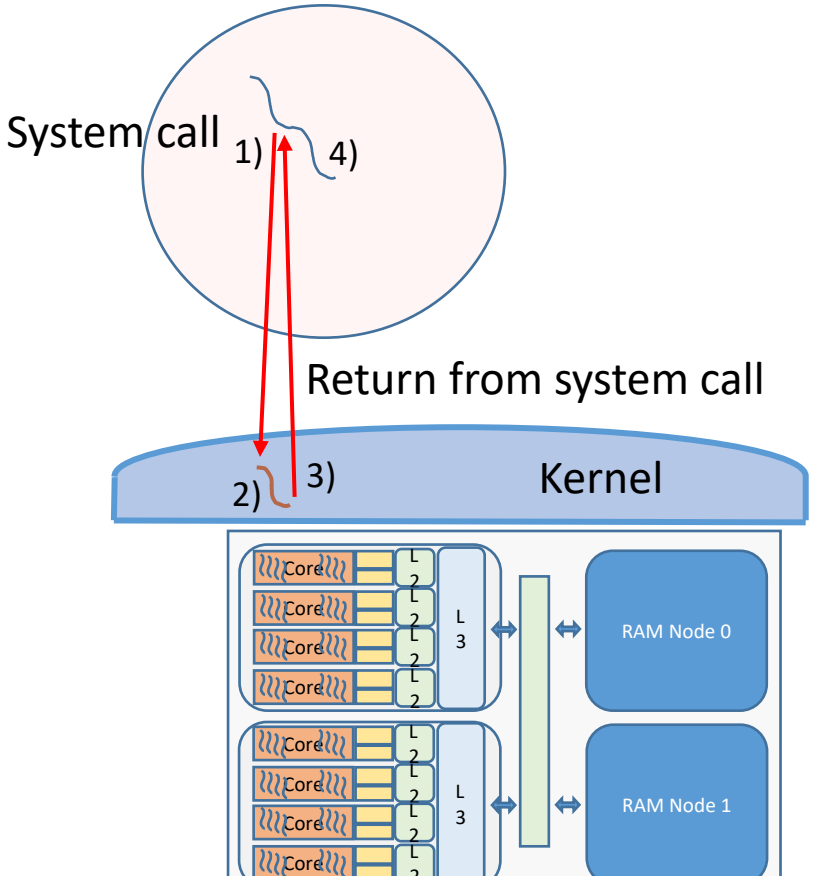
Floating point
- Division by 0
- Overflow
- Underflow
- Inexact

Bad address (segmentation fault/bus error)

Page fault
Breakpoint
Invalid instruction
Alignment check
...

Quan s'executa el codi del Kernel? (III de III)

- Quan un programa demana serveis al **kernel** mitjançant una interficie que es diu **crida al sistema (system call)**

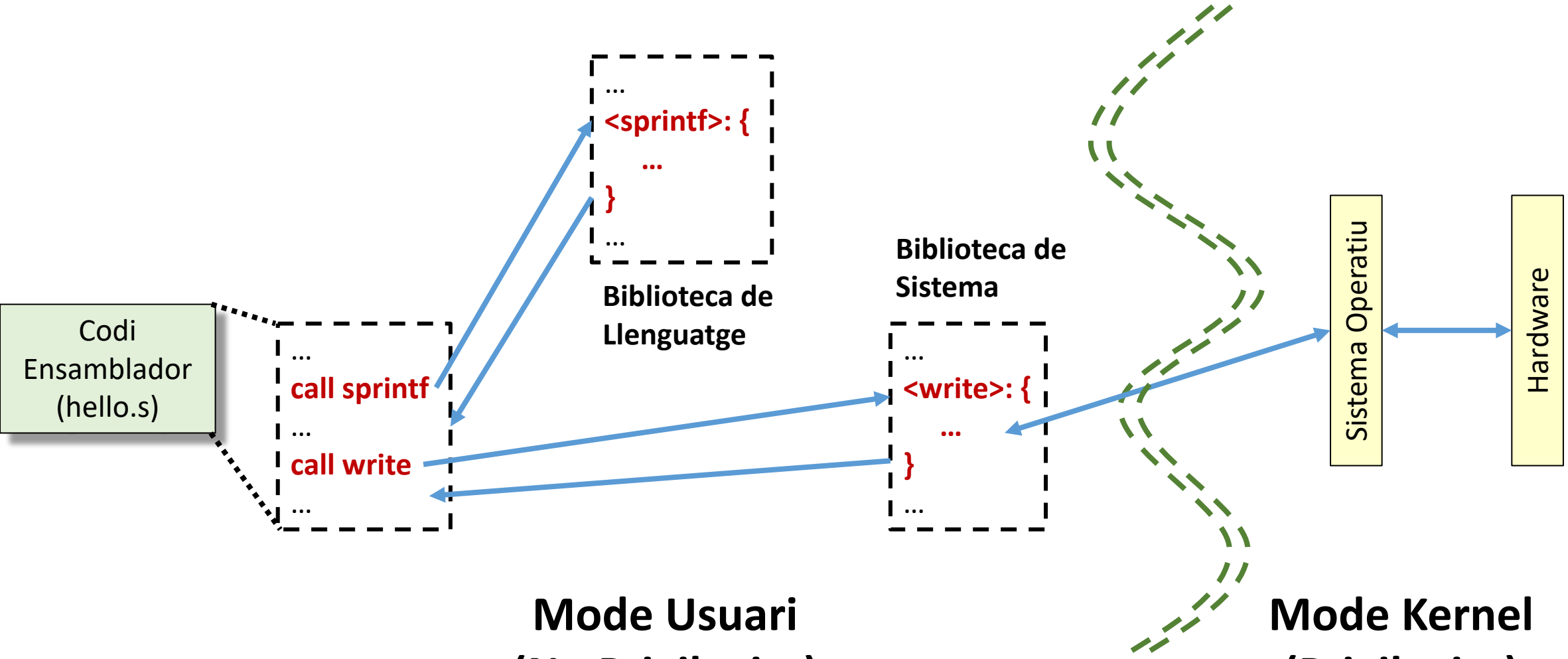


open, close, read, write, ioctl, fnctl, stat, fstat...
mount, umount...
fork, exec, exit, clone...
socket, bind, listen, accept, connect...
...

OpenFile, ReadFile, WriteFile...
CreateProcess, CreateProcessEx...
...

... up to tens of syscalls... (or hundreds...)

Mode d'execució no-privilegiat vs privilegiat



Crides al Sistema

- Una crida a sistema té més requisits que una “simple” crida a funció
  - Requisits
    - El codi del kernel s’ha d’executar en mode privilegiat
    - Per seguretat, el “salt” implicit en l’instrucció per cridar a la funció i el canvi del mode d’execució s’ha d fer amb una única instrucció de baix nivell
    - L’adreça de memòria de les rutines que implementen els serveis del SO podria canviar entre versions de kernel
  - A tenir en compte…
    - Canvis en el mode d’execució implica gestions i costos adicionals
      - Per exemple, gestió de la MMU per l’accés a memòria

Biblioteca de Sistema

- Per ocultar tots els detalls al usuari, el sistema proporciona una biblioteca que serà enllaçada amb els codis d'usuari
    - Això ho fa automàticament el compilador (e.g. gcc / g++)

- La biblioteca de sistema permet la transició des de l'API de les crides a sistema pel llenguatge específic (C, C++, etc.) al codi en ensamblador aplicant tots els requisits corresponents
    - Per C i C++, les crides a sistema estàn incloses en la biblioteca de C (libc)
        - Normalment en **/lib/x86_64-linux-gnu/libc.so** (shared)
          /usr/lib/x86_64-linux-gnu/libc.a (static)
    - Per Python, les crides a Sistema estàn disponibles en els mòduls "os" i "sys"

Un exemple similar vist anteriorment...

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

int global = 4;

int main(int argc, char **argv){
        char buf[512];
        int num;

        num = sprintf(buf, "Hello World!\n”);
        write(1, buf, num);

        return 0;
}
```

Linux i386 (32bits) vs Linux x64 (64bits)

Linux i386

```
...
movl  $4, %eax      ; use the write syscall
movl  $1, %ebx      ; write to stdout
movl  $msg, %ecx ; use string "Hello World!\n"
movl  $13, %edx   ; write 13 characters
int     $0x80        ; make syscall
...
```

Linux x64

```
...
movq $1, %rax     ; use the write syscall
movq $1, %rdi     ; write to stdout
movq $msg, %rsi ; use string "Hello World!\n“
movq $13, %rdx   ; write 13 characters
syscall                ; make syscall
...
```

Conceptes Bàsics de Gestió de Processos

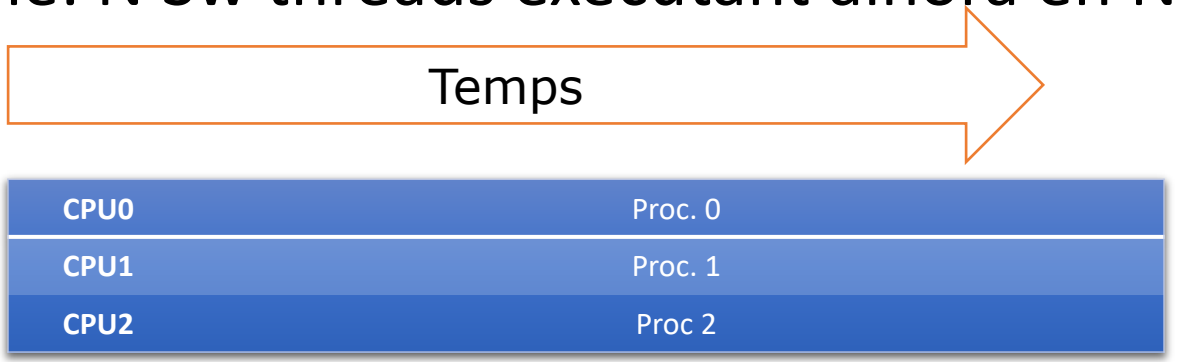
- **Programa**: fitxer executable amb permisos d'execució
- **Procés**: programa que s'ha llençat a executar
  - També és l'entitat a la qui se li assigna recursos
    - Per exemple: memòria, dispositius d'E/S, Hw Threads
- **Sw Thread**: fil d'execució que representa la secuencia d'instruccions a executar per un Hw thread
  - Un procés té com a mínim un thread
- **PCB**: Process Control Block
  - Estructura de dades que s'utilitza per gestionar l'execució d'un procés
  - Cada vegada que es crea un procés s'ha de crear un nou PCB
- **PID**: Process ID
  - Identificador únic en TOT el sistema que identifica cada PCB
- **TID**: Thread ID
  - Identificador únic en TOT el sistema que identifica cada Sw thread

Entorn Multi-Procés

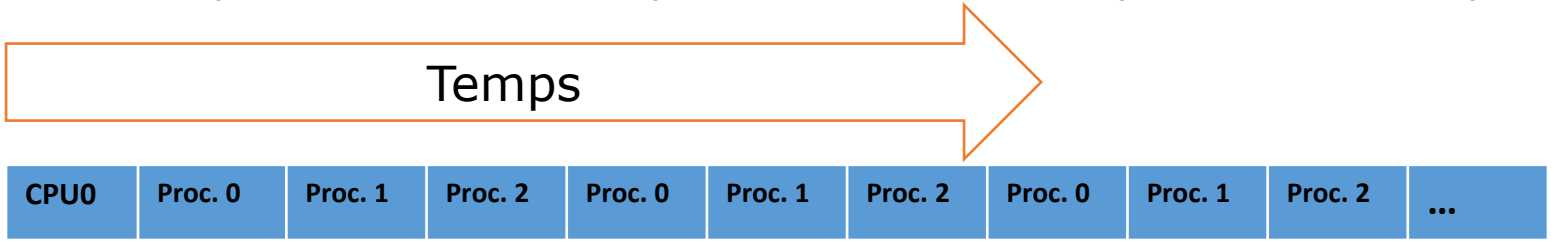
- Normalment hi han molts processos vius alhora en un SO convencional
  - Cada procés té un o més Sw Threads
- Els processos alternen l'utilització de la CPU amb l'ús d'altres recursos
  - En un entorn multi-programat, el SO gestiona com es comparteixen els recursos entre els processos
- En un sistema de propòsit general, el kernel alterna els processos que utilitzen la CPU (és a dir, els Sw threads que executen instruccions en els Hw threads)
  - **Canvi de context**: canvi del procés que executa instruccions sense perdre l'estat de l'execució
    - Es guarden dades bàsiques (estat de l'execució) per poder restaurar posteriorment l'execució
      - Per exemple, valor dels registres, quina és la següent instrucció a ser executada
  - **Política de planificació**: algorisme per decidir com alternar els processos
    - L'objectiu de les polítiques poden ser diferents: maximitzar rendiment, ser just, etc.

Paral·lelisme vs Concurrència

- Paral·lelisme: N Sw threads executant alhora en N Hw threads



- Concurrència: N Sw threads podrien ser potencialment executats en paral·lel, però no hi han sufficients recursos Hw per fer-ho
  - El SO selecciona quins Sw threads poden executar i quins han d'esperar



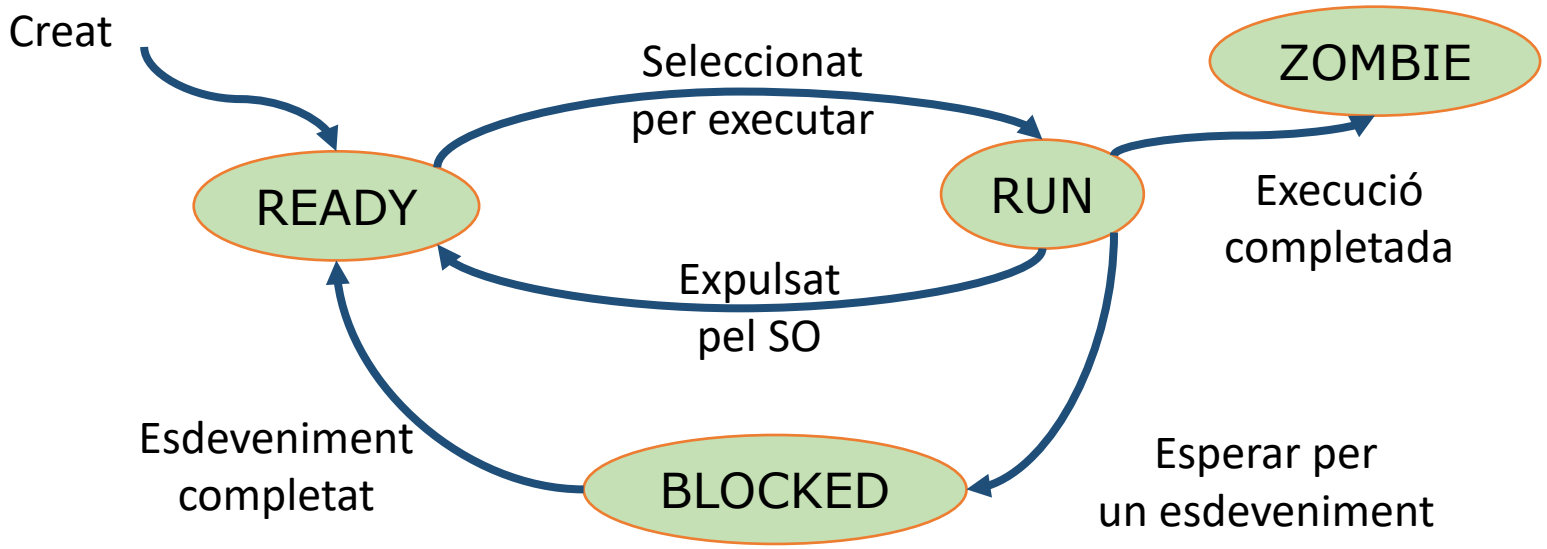
Gestió de Processos

- Estats:
  - **Ready**: el Sw Thread és candidat a ser seleccionat per poder executar instruccions
  - **Run**: el Sw Thread està executant instruccions en un Hw Thread
  - **Blocked**: el Sw Thread està a l'espera d'un event i per tant no és candidat a ser seleccionat per poder executar instruccions
  - **Zombie**: el Sw Thread ha finalitzat l'execució del programa

- Inici de l'execució
  - Es crea un PCB (PID i TID corresponents), s'assignen els recursos Hw necessaris i passa a estat "Ready"
    - Memòria, dispositius d'E/S, estadístiques, etc
  - Es carrega en memòria les instruccions i dades necessàries per executar el programa
- Finalització de l'execució
  - Alliberació de recursos en estat Zombie
    - S'alliberen els recursos Hw assignats
    - Es manté el PCB i uns pocs camps dins d'aquesta estructura
      - PID, TID, valor de finalització (voluntària o involuntària) i alguns pocs camps addicionals

Gestió de Processos

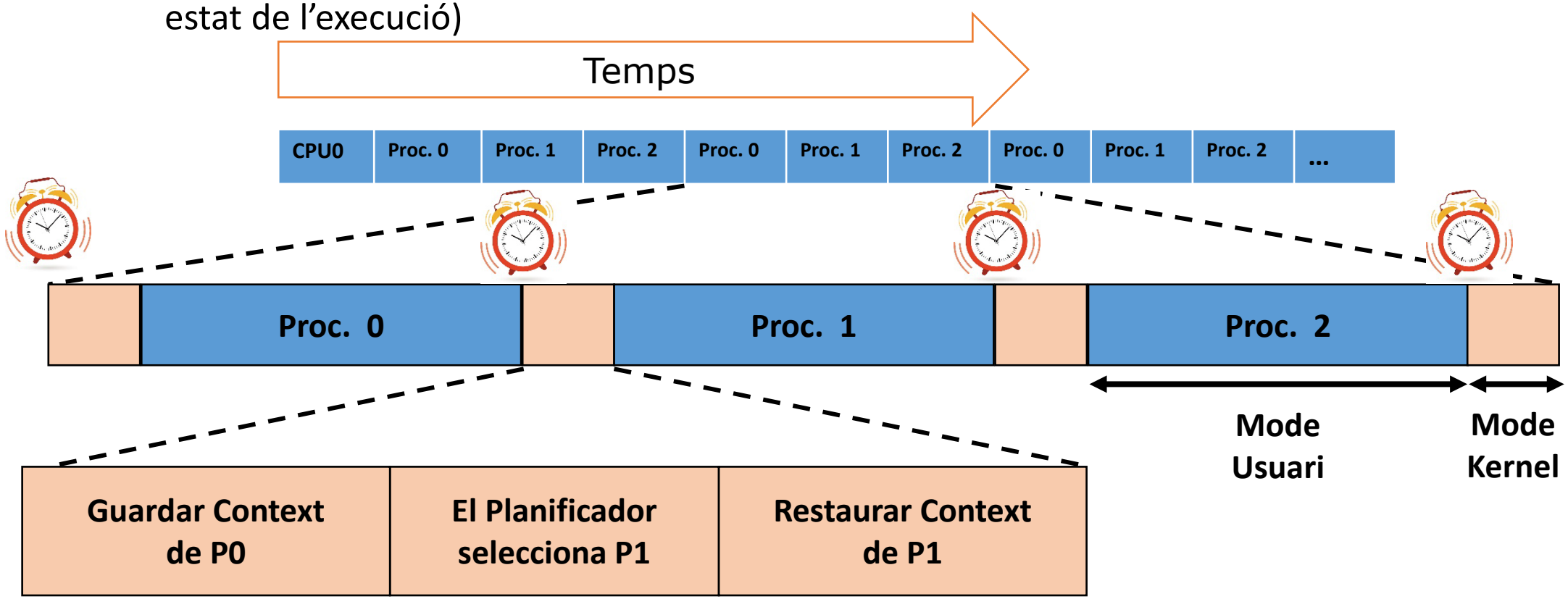
- Graf d'estats de execució de Sw Threads (o processos)



- Aquest és un graf d'estats genèric, però representa la base de la majoria dels kernels, encara que…
  - …cada kernel defineix el seu propi graf amb diferentes modificacions

Impacte del canvi de context en el rendiment

- **Canvi de context**: canvi del procés que executa instruccions
  - Implica un sobrecost degut a l'execució de codi kernel per gestionar el canvi (guardar/restaurar estat de l'execució)



Planificadors

- Els planificadors són critics pel rendiment del sistema

- **Curt** termini: en cada tick del SO (vist prèviament)
  - Quin és el següent Sw thread en executar en la CPU?

- **Mitjà** termini: quan el SO detecta que recursos Hw s'esgoten (e.g. memòria)
  - Quins processos són candidats a alliberar **temporalment** els recursos que s'esgoten (memòria) per permetre que altres els utilitzin

- **Llarg** termini (*opcional*): cada començament/final d'un procés
  - Quin és el nombre màxim de processos apropiat per executar en el sistema
    - Controla el nivell de multiprogramació del sistema

Exemple de Planificador a Curt Termini

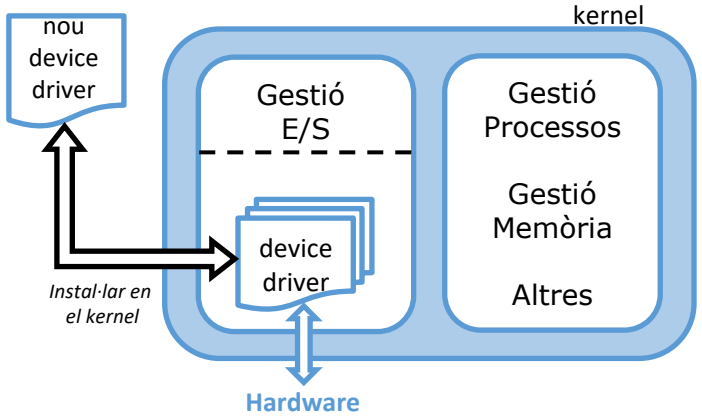
- **CFD: Completely Fair Scheduling**
  - Algorisme utilitzat actualment en la majoria de les distribucions de Linux
- Objectiu:
  - El temps d'utilització de la CPU ha de ser similar en tots els processos
    - Altres algorismes penalitzen processos intensius en E/S
  - Evitar que un usuari que executa molts processos acapari la CPU
- Detalls:
  - El temps max d'ús consecutiu de la CPU (també conegut com **quantum**) és variable
  - Sistema de prioritats variables
    - Més prioritat quan més lluny està de la mitjana d'ús de CPU de tots els processos
  - Creació i gestió de grups de processos (criteri configurable pel administrador)
    - Per comptabilitzar el consum de CPU per grup de processos

Gestió de Entrada/Sortida

- E/S és una de les activitats clau dels processos
  - L'E/S és la capacitat d'enviar/rebre dades des de/cap a un procés
    - L'acció és sempre realitzada des del punt de vista del procés
  - Això vol dir que també inclou comunicació entre processos
    - Per exemple, "sockets" per comunicar processos mitjançant la xarxa
- Accedir i gestionar dispositius són operacions "privilegiades"
  - Necessita tenir una gestió segura per garantir el correcte ús entre usuaris
  - Root / administrador és qui pot gestionar-ho a nivell d'administrador
- Per poder realitzar operacions d'E/S és necessari crides a sistema
  - Són independents de les característiques del dispositiu
- **IMPORTANT:** una operació d'E/S pot bloquejar un procés fins que es completi
  - Estat "BLOCKED" i per tant no utilitza CPU
  - Hi ha diverses alternatives per minimitzar impacte
    - E.g. Operacions asíncrones (o no bloquejants), transferències amb buffer, planificació eficient, ...

Device Drivers

- Els programadors no poden desenvolupar codi per tots els diferents dispositius que existeixen
- Els fabricants han de proporcionar el conjunt de codi de baix nivell (codi compilat) que necessita el dispositiu per realitzar la funcionalitat
  - El codi + dades per accedir al dispositiu es conneix com **Device Driver**



- Per afegir un nou dispositiu
  1. Opció 1: recompilar el kernel
  2. Opció 2: **sense** recompilar el kernel
     - El SO ha d'oferir un mecanisme per afegir dinàmicament codi/dades al kernel de manera segura
       - *Dynamic Kernel Module Support*, *Plug & Play*

Device Drivers: controlador de dispositiu

- Conjunt de rutines que gestionen un dispositiu i permet a un programa interactuar amb ell
  - Segueix l'interfície definida per cada SO per implementar tasques genèriques
    - Obrir, llegir, escriure, tancar…
  - Implementa tasques específiques del dispositiu
  - Conté codi de baix nivell per gestionar interrupcions i altres elements del dispositiu
  - Són fitxers binaris

Independència de Dispositiu

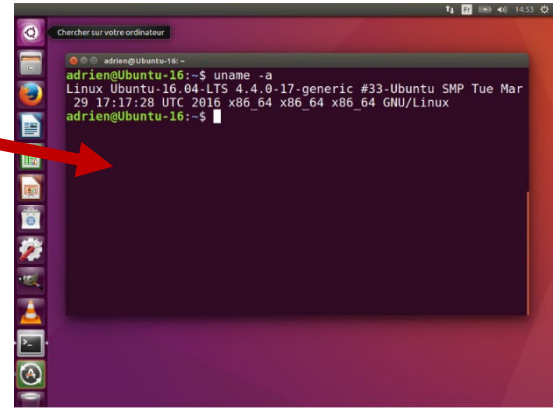
- Els dispositius presenten una gran varietat de característiques i necessitats
  - Per exemple, velocitat, tipus de dispositiu, tipus de transferència, etc.

- Classificació interna
  - UNIX/Linux: **character/block**, → per exemple, teclat (char) / disc (block)
    major (quin tipus de dispositiu?) → és un número enter >= 0
    minor (quina instància és?) → és un número enter >= 0

- Per què l'independència del dispositiu és important?
  - **Operacions uniformes d'E/S** – open, close, read, write...
  - **Dispositius virtuals** – només utilitza un identificador (file descriptor)
  - **Redireccionament** – canviar el dispositiu que s'utilitza sense canviar el codi (vist en el Lab)

Suport del SO pel subsistema d'E/S

- Qualsevol dispositiu d'E/S necesita ser accessible
  - En UNIX/Linux els dispositius es representen en el Sistema de Fitxers
    - E.g.: un terminal pot ser: "/dev/pts/0"

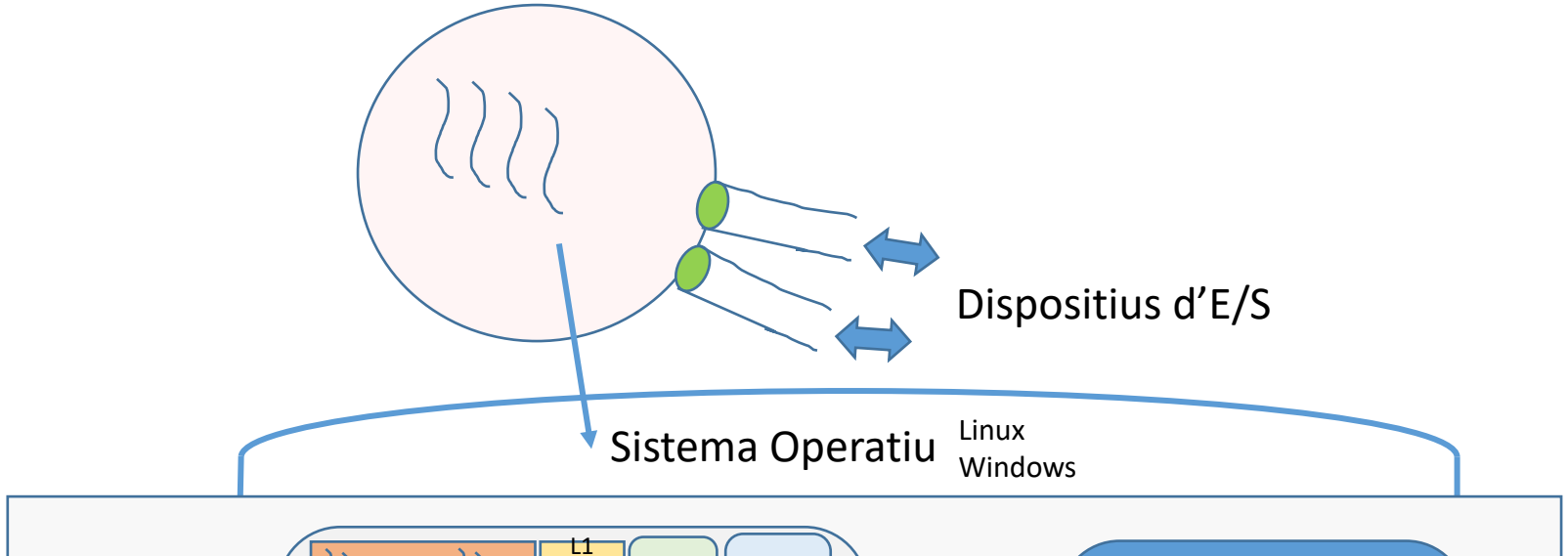


- El SO introdueix varies capes d'abstracció…
  - …per simplificar la gestió d'E/S
    - Ha d'oferir crides a sistema independents dels internals dels dispositius

- El SO gestiona, per cada procés, una taula privada de dispositius que utilitza. Pel procés són dispositius abstractes (o "virtuals")
  - **File descriptor table**: una taula de punters a un altra estructura global del SO

Taula de descriptors de fitxers

- Cada procés té una taula privada
  - És el punt d'accés per part del codi d'usuari per utilitzar dispositius
    - Totalment independent de les característiques del dispositiu
  - Cada dispositiu està representat pel número de l'entrada que se li ha assignat
    - Quan es demana començar a utilitzar un dispositiu s'assigna la primera entrada disponible
    - Els tres primers (0→STDIN; 1→STDOUT; 2→STDERR) per defecte estan assignats (per defecte a la terminal)

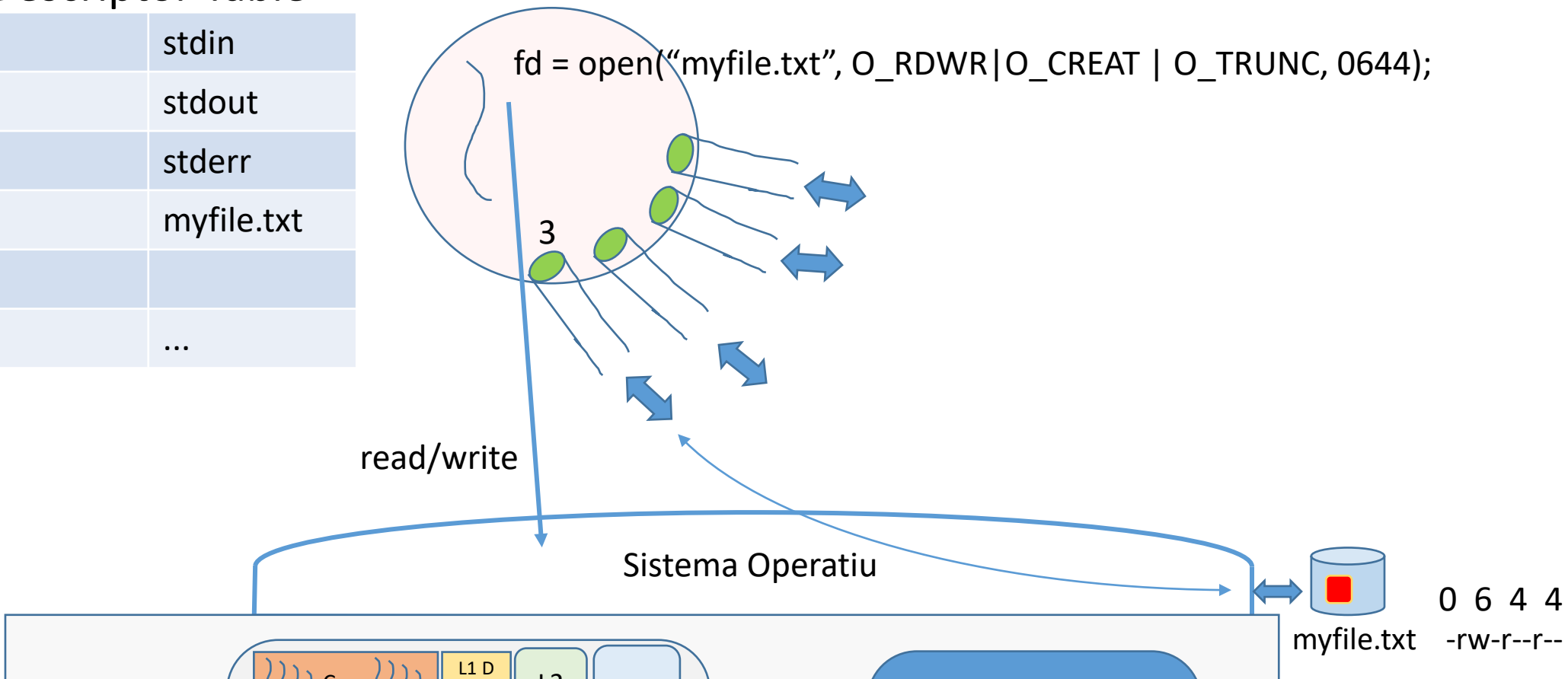


Taula de descriptors de fitxers

En Linux es pot consultar en /proc/<PID>/fd

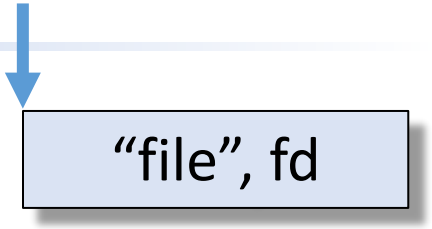
File Descriptor Table

0	stdin
1	stdout
2	stderr
3	myfile.txt
4	
...	...



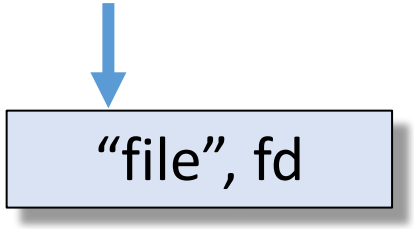
Crides a sistema bàsiques de E/S

Per defecte ens posicionem al principi del fitxer quan es fa l'open



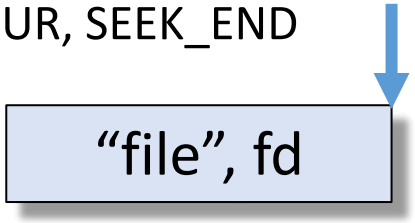
- fd = open(path, flags[, permissions]);
  - Flags bàsics:
    - O_RDONLY O_WRONLY O_RDWR
  - Combinacions de flags:
    - O_WRONLY|O_CREAT|O_TRUNC
  - Permissions:
    - Control d'accés al fitxer que es crea (0644 -> rw-r—r--)

Internament el SO actualitza l'offset (posició) en la que està després de r/w



- bytes = read(fd, @data, bytes);
- bytes = write(fd, @data, bytes);

- new_offset = lseek(fd, offset, whence);
  - Desplaçarse dins del fitxer sense accessos a disc
  - Whence: SEEK_SET, SEEK_CUR, SEEK_END



Es pot anar directament a punts remots del fitxer sense accedir a disc

- close(fd);

Suport de la biblioteca C

- Ofereix serveis d'E/S d'alt nivell
  - **FILE** és una estructura que engloba més que el "file descriptor"
  - Afegeix altres característiques per facilitar l'ús
    - Especialment **BUFFERING**
- Cada funció invoca a la crida a sistema corresponent
  - E.g. "fopen" crida a "open"
- Llegir i escriure...
  - ...està preparat per transferir un nombre d'elements d'una mida (*elemsize* bytes)
- Lseek funciona de la mateixa manera

```
FILE * f;
size_t size;  int res;

f = fopen(path, mode);
     // mode = "r"    read-only
     // mode = "w"  truncate, write-only
     // mode = "a"   append
     // mode = "r+"  read-write
size = fread (data, elemsize, numelems, f);
if (size < 0) perror ("fread");

size = fwrite (data, elemsize, numelems, f);

res = fseek(f, newoffset, whence);
     // whence = SEEK_SET, SEEK_CUR, SEEK_END

res = fclose(f);
```

Suport de la biblioteca C

- Hi ha altres funcions addicionals per fer E/S pensades per text
  - printf, escriure per la sortida estàndard **stdout**
  - fprintf, utilitzar una sortida diferent
  - sprintf, escriure en un array de chars

```
FILE *f;
f = fopen("myfile", "r+");
if (f == NULL) {
    fprintf (stderr, "fopen: %s\n",strerror(errno));
    exit(1);
}
fprintf (f, "my message");
```

  - scanf, llegir una string de format concret des de l'entrada estàndard **stdin**
  - fscanf, llegir una string de format concret des de una entrada diferent
  - sscanf, llegir una string de format concret des d'una string

En S8-FC_GIA/check-multiply.XXX-wise.c (session 8) s'utilitza fscanf i sscanf:

```
char line[4096];    // assuming lines will be smaller than 4K
int i; double d; char str[11];

fgets (line, sizeof(line), f);
n = sscanf(line, "%d %lf %10s", &i, &d, str);
     // will understand lines like   10   4.5   hola
     // and assign i=10, d=4.5, str="hola"
```

Impacte en el Rendiment

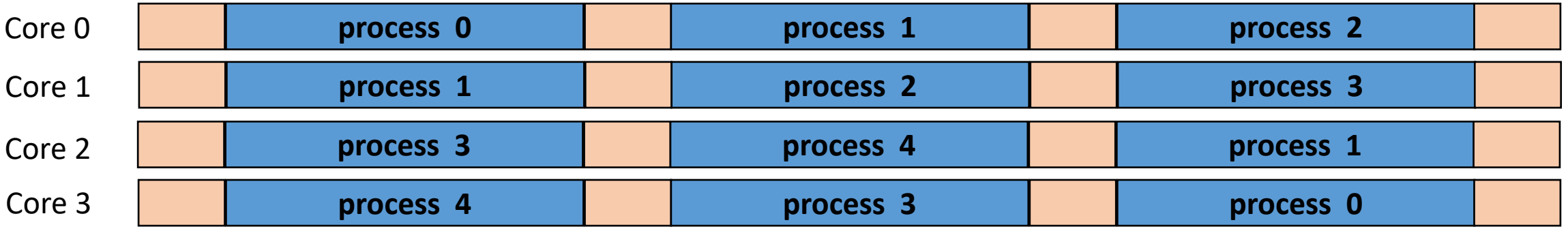
- Una operació d'E/S pot **bloquejar** un procés (operació síncrona)
  - Si les dades es poden disposar al moment, torna immediatament
  - Si les dades no estan disponibles, el procés passa d'estat "Running" a "BLOCKED" i per tant no utilitza CPU fins que es completi
- Una operació d'E/S pot ser **no bloquejant**
  - Gestió d'identificadors de l'operació asíncrona
  - Retorna immediatament tant si l'operació s'ha completat com si no
- Les rutines d'E/S són crides a sistema
  - Tenen l'overhead implícit per passar a mode privilegiat (executar codi del kernel)
  - Utilitzar buffers per reduir el nombre de crides a sistema

Impacte en el rendiment i l'energia

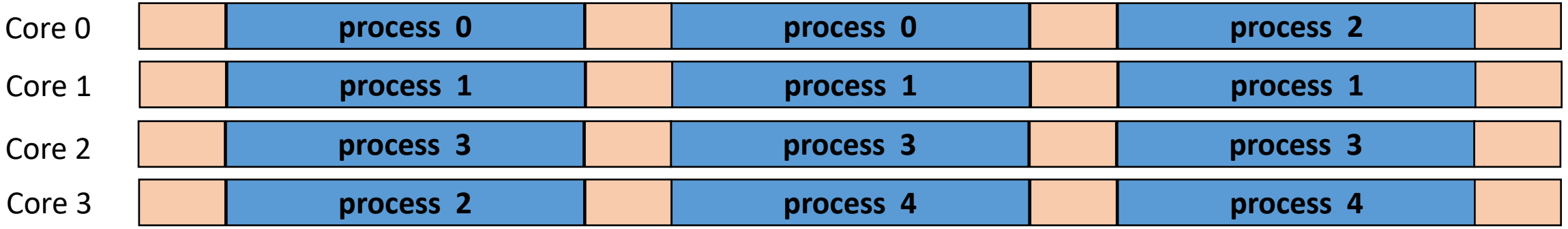
- El SO ha de tenir en compte
  - Obtenir el rendiment necessari dels recursos Hw...
  - ... preservant el consum energètic
- Tècniques per reduir consum energètic a nivell de...
  - Arquitectura
    - Menor freqüència i voltatge del processador
    - Incorporar optimitzacions com ara operacions vectorials
  - Compilador
    - Generar codi amb instruccions que consumeixin menys energia, per exemple instruccions vectorials, simplificar codi de funcions
  - SO
    - Millorar el planificador per aconseguir millor localitat en la jerarquia de memòria
    - Saber gestionar els canvis de context entre processos que fan E/S vs computació

Impacte en el rendiment i l'energia

- El planificador intenta mantenir la localitat dels processos als cores



vs.



Impacte en el rendiment i l'energia

- La planificació de processos acostuma a ser caòtica, i Linux és un exemple…

Starting

```
Stat  PU  PID    Prog
R+     0 181181 matmul
R+     2 181175 matmul
R+     3 181176 matmul
R+     4 181180 matmul
R+    20 181179 matmul
R+    21 181177 matmul
R+    26 181182 matmul
R+    27 181178 matmul
```

after 1 second

```
Stat  PU  PID    Prog
R+     0 181175 matmul
R+     1 181179 matmul
R+     2 181181 matmul
R+    28 181177 matmul
R+    29 181178 matmul
R+    30 181176 matmul
R+    32 181182 matmul
R+    34 181180 matmul
```

after 10 seconds

```
Stat  PU  PID    Prog
R+     0 181180 matmul
R+     1 181179 matmul
R+     2 181181 matmul
R+     3 181177 matmul
R+    29 181178 matmul
R+    30 181176 matmul
R+    32 181182 matmul
R+    35 181175 matmul
```

- És un tema de recerca ampli en SOs

$ ps –e –o stat,psr,pid,comm | grep "^R" | sort –n –k2

Impacte en el rendiment i l'energia

- Moure processos entre cores genera moltes invalidacions en la jerarquia de memòria (L1, L2, L3)

Stat	PU	PID	Prog
R+	0	181178	matmul
R+	1	181179	matmul
R+	2	181181	matmul
R+	3	181177	matmul
R+	30	181176	matmul
R+	31	181180	matmul
R+	32	181182	matmul
R+	35	181175	matmul

Stat	PU	PID	Prog
R+	0	181178	matmul
R+	**1**	**181181**	**matmul**
R+	31	181180	matmul
R+	**33**	**181179**	**matmul**
R+	**34**	**181182**	**matmul**
R+	35	181175	matmul
R+	**44**	**181176**	**matmul**
R+	**47**	**181177**	**matmul**

- Normalment, quant més càrrega, més quantitat de moviments

Impacte en el rendiment i l'energia

- Eines del SO per controlar el consum d'energia
  - Cpufrequtils: permet controlar la política i freqüència del processador
    - Política de rendiment - freqüència alta
    - Política de minimitzar consum - baixa freqüència
  - Exemple:
    - Executant matmul ...

```
$ cpufreq-info
...
Analyzing CPU 0:
  driver: intel_pstate
  CPUs which run at the same frequency: 0
  hardware limits: 400 MHz – 2.80 GHz
  available cpufreq governors: performance, powersave
  current policy: "performance" may decide the speed
  current CPU frequency is 2.70 GHz
```

Sense executar "res" ...

```
$ cpufreq-info
...
Analyzing CPU 0:
  driver: intel_pstate
  CPUs which run at the same frequency: 0
  hardware limits: 400 MHz – 2.80 GHz
  available cpufreq governors: performance, powersave
  current policy: "powersave" may decide the speed
  current CPU frequency is 792 MHz
```

Següents passos / cursos

- Paral·lelisme amb memòria compartida
  - Multi-threading – Pthreads
  - OpenMP
- Sistemes distribuïts
  - Networking
  - Message Passing Interface (MPI)
  - Distributed applications
- Computació d'Alt Rendiment

Fonaments dels Computadors ✓

Paral·lelisme i Sistemes Distribuïts

Computació d'Altes Prestacions

Bibliografia

- Computer Systems – A Programmer's perspective (3rd Edition)
  - Randal E. Bryant, David R. O'Hallaron, Person Education Limited, 2016
    - https://discovery.upc.edu/permalink/34CSUC_UPC/11q3oqt/alma991004062589706711
  - Chapter 7: Linking
  - Chapter 8, sections 8.2: Processes, 8.4: Process Control, 8.7: Tools for manipulating processes
  - Chapter 10: System Level I/O

- GCC Compiler Tutorial
  - http://gcc.gnu.org/

- Python Tutorial
  - https://docs.python.org/3/tutorial/