

# API de procese

shell  
ANSI C  
POSIX  
Win32  
Python  
Java

### Lucrul cu procese in ANSI C

```
fork() // crea proces  
wait() // asteapta procesul sa termine  
waitpid() // asteapta un anumit proces  
system() // executa un program in noua stare de proces
```

### Lucrul cu procese in POSIX

```
fork() // crea proces  
wait() // asteapta procesul sa termine  
waitpid() // asteapta un anumit proces  
system() // executa un program in noua stare de proces
```

### Lucrul cu procese in POSIX (2)

```
fork() // crea proces  
wait() // asteapta procesul sa termine  
waitpid() // asteapta un anumit proces  
system() // executa un program in noua stare de proces
```

### Lucrul cu procese in Win32

```
ProcessID_t ProcessID; // ID-ul procesului  
HANDLE hProcess; // handle-ul procesului  
HANDLE hThread; // handle-ul thread-ului  
HANDLE hThread2; // handle-ul thread-ului 2
```

### Lucrul cu procese in Python

```
subprocess.Popen() // crea proces  
subprocess.Popen().wait() // asteapta procesul sa termine  
subprocess.Popen().communicate() // asteapta procesul sa termine si returneaza output-ul
```

### Lucrul cu procese in Java

```
ProcessBuilder // crea proces  
ProcessBuilder.start() // executa procesul  
ProcessBuilder.waitFor() // asteapta procesul sa termine
```

### Lucrul cu procese in shell

```
fork() // crea proces  
wait() // asteapta procesul sa termine  
waitpid() // asteapta un anumit proces  
system() // executa un program in noua stare de proces
```

### Descriptorii de resurse

File descriptor // identificator de resurse  
open() // creeaza descriptor de resurse  
close() // inchide descriptor de resurse

### Ierarhia de procese

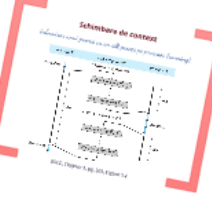
Process // proces  
Thread // thread  
Process Group // grup de procese  
Process ID // ID-ul procesului

### Multitasking

Context switch // schimbare de context  
Scheduler // program de planificare  
Priority // prioritate

### Tranzitii intre stari ale unui proces

Running // procesul este in executie  
Ready // procesul este gata de executie  
Waiting // procesul este in asteptare  
Terminated // procesul a terminat



### Semnale

Signal // mesaj de comunicare  
kill() // trimite un semnal  
killpg() // trimite un semnal unui grup de procese  
waitpid() // asteapta un proces si returneaza semnalul primit

### Resursele unui proces

File descriptor // identificator de resurse  
Memory // memorie  
Stack // stiva  
Heap // heap

## Atributele unui proces

Process Control Block (PCB)  
Resursele unui proces  
Ierarhia de procese  
Starea unui proces



### Paralelism la nivel de SO

Parallelism // executie simultana  
Concurrency // executie aparenta simultana  
Multiprocessing // executie simultana pe mai multe procesoare

## Planificarea proceselor

Paralelism la executie  
Tranzitii intre stari  
Schimbare de context  
Planificatorul de procese

### Planificarea proceselor

Scheduler // program de planificare  
Priority // prioritate  
Time slice // interval de timp

### Probleme in folosirea semnalelor

Signal // mesaj de comunicare  
Race condition // situatie in care doua procese acceseaza aceeasi resursa  
Deadlock // situatie in care doua procese asteapta unul altuia

### Process Control Block (PCB)

PCB // structura de date care descrie un proces  
PCB // structura de date care descrie un proces  
PCB // structura de date care descrie un proces

## Comunicarea interproces

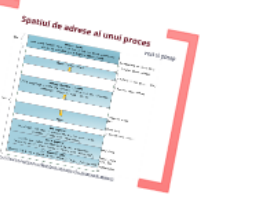
Semnale  
Pipe-uri  
Memorie partajata  
Message passing  
Socketi

### Program vs. Proces

Program // cod surs  
Process // executie  
Process // executie  
Process // executie

### De ce procese?

Parallelism // executie simultana  
Concurrency // executie aparenta simultana  
Multiprocessing // executie simultana pe mai multe procesoare



### Fork si exec in Unix

fork() // crea proces  
exec() // executa program

### Incheierea unui proces

wait() // asteapta procesul sa termine  
waitpid() // asteapta un anumit proces sa termine  
system() // executa un program in noua stare de proces

### Terminarea unui proces

wait() // asteapta procesul sa termine  
waitpid() // asteapta un anumit proces sa termine  
system() // executa un program in noua stare de proces

### Pipe-uri

Pipe // canal de comunicare  
pipe() // creeaza pipe  
read() // citeste din pipe  
write() // scrie in pipe

### Ce este un proces?

Process // executie  
Process // executie  
Process // executie

## Procese

Ce este un proces?  
Program si procese  
De ce procese?  
Spatiul de adrese al unui proces  
Tipuri de procese

### Tipuri de procese

Process // executie  
Thread // thread  
Process Group // grup de procese

### Crearea unui proces

fork() // crea proces  
exec() // executa program

## Operatii cu procese

Crearea unui proces  
Incheierea unui proces  
Terminarea unui proces  
Asteptarea unui proces

### Asteptarea unui proces

wait() // asteapta procesul sa termine  
waitpid() // asteapta un anumit proces sa termine  
system() // executa un program in noua stare de proces

### Memorie partajata

Shared memory // memorie partajata  
shmget() // creeaza memorie partajata  
shmat() // adauga proces la memorie partajata

### Cuprins

Procese  
Stari ale proceselor  
Planificarea proceselor  
Operatii cu procese  
Comunicatia interproces

### Suport de curs

OSCE  
Capitolul 1 - Procese  
Capitolul 2 - Processes and Threads  
Capitolul 3 - Process Management  
Capitolul 4 - Advanced Process Management  
Capitolul 5 - Signals  
Capitolul 6 - Process Management  
Capitolul 7 - Interprocess Communication

# SO Curs 3

### Cuvinte cheie

proces  
program  
ierarhia de procese  
PCB  
atribute de proces  
starea unui proces  
descriptorii de resurse  
multitasking  
context switch  
planificator

### Message Passing

Message passing // comunicare intre procese  
Pipe // canal de comunicare  
Socket // punct de comunicare

### Socketi

Socket // punct de comunicare  
Socket // punct de comunicare  
Socket // punct de comunicare



# API de procese

# SO Curs 3

## Procese

### Atributelor unui proces

Process Control Block (PCB)  
Resursele unui proces  
Ierarhia de procese  
Starea unui proces

### Planificarea proceselor

Paralelism la executie  
Transitii intre stari  
Schimbare de context  
Planificatorul de procese

### Procese

Ce este un proces?  
Program si procese  
De ce procese?  
Spatiul de adrese al unui proces  
Tipuri de procese

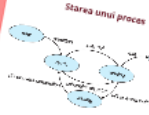
### Operatii cu procese

Crearea unui proces  
Incheierea unui proces  
Terminarea unui proces  
Asteptarea unui proces

### Comunicarea interproces

IPC (Inter-Process Communication)  
Semnale  
Pipe-uri  
Memorie partajata  
Message passing  
Sockets

### Ierarhia de procese



### Multitasking

Procedurile de executie sunt de fapt procese care sunt executate in paralel.  
Un proces poate fi executat intr-un anumit timp de procesare.  
Un proces poate fi executat intr-un anumit timp de procesare.  
Un proces poate fi executat intr-un anumit timp de procesare.

### Paralelism la nivel de SO

Un proces poate fi executat intr-un anumit timp de procesare.  
Un proces poate fi executat intr-un anumit timp de procesare.  
Un proces poate fi executat intr-un anumit timp de procesare.

### Planificarea proceselor



### Terminarea unui proces

Procesul este terminat din cauza:  
- erorii de program  
- finalizarii  
- expirarii termenului de executie  
- intervenției operatorului

### Asteptarea unui proces

Procesul poate deveni inactiv din cauza:  
- lipsei resurselor necesare  
- lipsei datelor necesare  
- lipsei resurselor necesare  
- lipsei datelor necesare

### Procese zombii si orfane

Un proces zombii este un proces care nu este mai executat, dar care nu a fost terminat.  
Un proces orfan este un proces care nu are niciun proces parinte.

### Pipe-uri

Un canal de comunicare unidirecțional între două procese.  
Este folosit pentru comunicarea datelor între procese.

### Memorie partajata

Un mecanism de comunicare între două procese care permite accesul comun la o zonă de memorie.

### Sockets

Un mecanism de comunicare între două procese care permite comunicarea de date între procese.

**Lucrul cu procese in ANSI C**

```
fork() // crea proces nou
wait() // asteptarea procesului
```

**Lucrul cu procese in POSIX**

```
fork() // crea proces nou
waitpid() // asteptarea procesului
```

**Lucrul cu procese in POSIX (2)**

```
fork() // crea proces nou
waitpid() // asteptarea procesului
```

**Lucrul cu procese in Win32**

```
Process() // crea proces nou
WaitForSingleObject() // asteptarea procesului
```

**Lucrul cu procese in Python**

```
subprocess.Popen() // crea proces nou
subprocess.wait() // asteptarea procesului
```

**Lucrul cu procese in Java**

```
ProcessBuilder // crea proces nou
ProcessBuilder.waitFor() // asteptarea procesului
```

**Lucrul cu procese in shell**

```
system() // executa comanda in shell
systemv() // executa comanda in shell
```

**Descriptorii de resurse**

Un descriptor de resurse este un număr care reprezintă o resursă deschisă.

**Ierarhia de procese**

Un proces poate crea un alt proces, care poate crea un alt proces, și așa mai departe.

**Multitasking**

Execuția simultană a mai multor procese.

**Transitii intre starea unui proces**

Procesul poate fi in starea Ready, Running sau Waiting.

**Schimbare de context**

Procesul este mutat dintr-o stare in alta.

**Semnale**

Un mijloc de comunicare între două procese.

**Probleme in folosirea semnalelor**

Probleme de sincronizare și ordine de executie.

**Resursele unui proces**

Procesul are acces la resursele sistemului și ale procesului parinte.

**De ce procese?**

Modul de abstractizare a activitatii și organizarea resurselor.

**Starea unui proces**

Diagrama care arata starea unui proces in timpul executiei.

**Paralelism la nivel de SO**

Execuția simultană a mai multor procese la nivel de sistem de operare.

**Planificarea proceselor**

Algoritmi pentru alegerea procesului care urmeaza sa fie executat.

**Comunicarea interproces**

Moduri diferite de comunicare între două procese.

**Process Control Block**

Structura de date care descrie un proces in sistemul de operare.

**De ce procese?**

Beneficiile proceselor și modul de organizare a activitatii.

**Spatiul de adrese al unui proces**

Diagrama care arata structura adresei memoriei pentru un proces.

**Fork si exec in Unix**

Modurile de creare a proceselor noi in sistemul de operare Unix.

**Incheierea unui proces**

Modurile de terminare a unui proces in sistemul de operare.

**Pipe-uri**

Canale de comunicare unidirecționale între procese.

**Program vs. Proces**

Diferența dintre un program sursă și un proces executabil.

**De ce procese?**

Importanța proceselor în sistemul de operare și beneficiile acestora.

**Tipuri de procese**

Categoriile diferite de procese existente in sistemul de operare.

**Crearea unui proces**

Modurile de creare a proceselor noi in sistemul de operare.

**Operatii cu procese**

Modurile de control al proceselor in sistemul de operare.

**Memorie partajata**

Mechanismul de comunicare între două procese care permite accesul comun la o zonă de memorie.

**Ce este un proces?**

Definiția și caracteristicile unui proces in sistemul de operare.

**De ce procese?**

Beneficiile proceselor și modul de organizare a activitatii.

**Tipuri de procese**

Categoriile diferite de procese existente in sistemul de operare.

**Crearea unui proces**

Modurile de creare a proceselor noi in sistemul de operare.

**Operatii cu procese**

Modurile de control al proceselor in sistemul de operare.

**Memorie partajata**

Mechanismul de comunicare între două procese care permite accesul comun la o zonă de memorie.

**Ce este un proces?**

Definiția și caracteristicile unui proces in sistemul de operare.

**Cuprins**

- Procese
- Stari ale proceselor
- Planificarea proceselor
- Operatii cu procese
- Comunicatia interproces

**Suport de curs**

- OSCE
- Capitolul 1 - Procese
- MCS
- Capitolul 2 - Processes and Threads
- Secțiunea 1
- LSP
- Capitolul 3 - Process Management
- Capitolul 4 - Advanced Process Management
- Capitolul 5 - Signals
- WSP
- Capitolul 6 - Process Management
- Capitolul 11 - Interprocess Communication

**Tipuri de procese**

Categoriile diferite de procese existente in sistemul de operare.

**Crearea unui proces**

Modurile de creare a proceselor noi in sistemul de operare.

**Operatii cu procese**

Modurile de control al proceselor in sistemul de operare.

**Memorie partajata**

Mechanismul de comunicare între două procese care permite accesul comun la o zonă de memorie.



# Suport de curs

## OSCE

- Capitolul 3 – Process

## MOS

- Capitolul 2 – Processes and Threads
  - Sectiunea 1

## LSP

- Capitolul 5 – Process Management
- Capitolul 6 - Advanced Process Management
- Capitolul 9 – Signals

## WSP

- Capitolul 6 – Process Management
- Capitolul 11 – Interprocess Communication

# Cuprins

*Procese*

*Stari ale proceselor*

*Planificarea proceselor*

*Operatii cu procese*

*Comunicatia interproces*

# Procese

*Ce este un proces?*

*Programe si procese*

*De ce procese?*

*Spatiul de adrese al unui proces*

*Tipuri de procese*

# Ce este un proces?

Un program aflat in executie  
Instanta a unui program

Abstractizarea actiunii in sistemul de operare

Abstractizare peste procesor

Ce se face si cu ce resurse

# Program vs. Proces

pasiv  
executabil  
date, cod  
cale in sistemul de fisiere  
imaginea unui proces

activ  
memorie & CPU  
date, cod, heap, stiva  
PID (Process ID)  
instanta unui program  
resurse  
spatiu de adresa

Care este asocierea program - proces?

# De ce procese?

Mod de abstractizare a actiunii

Unitatea de lucru in sistemul de operare

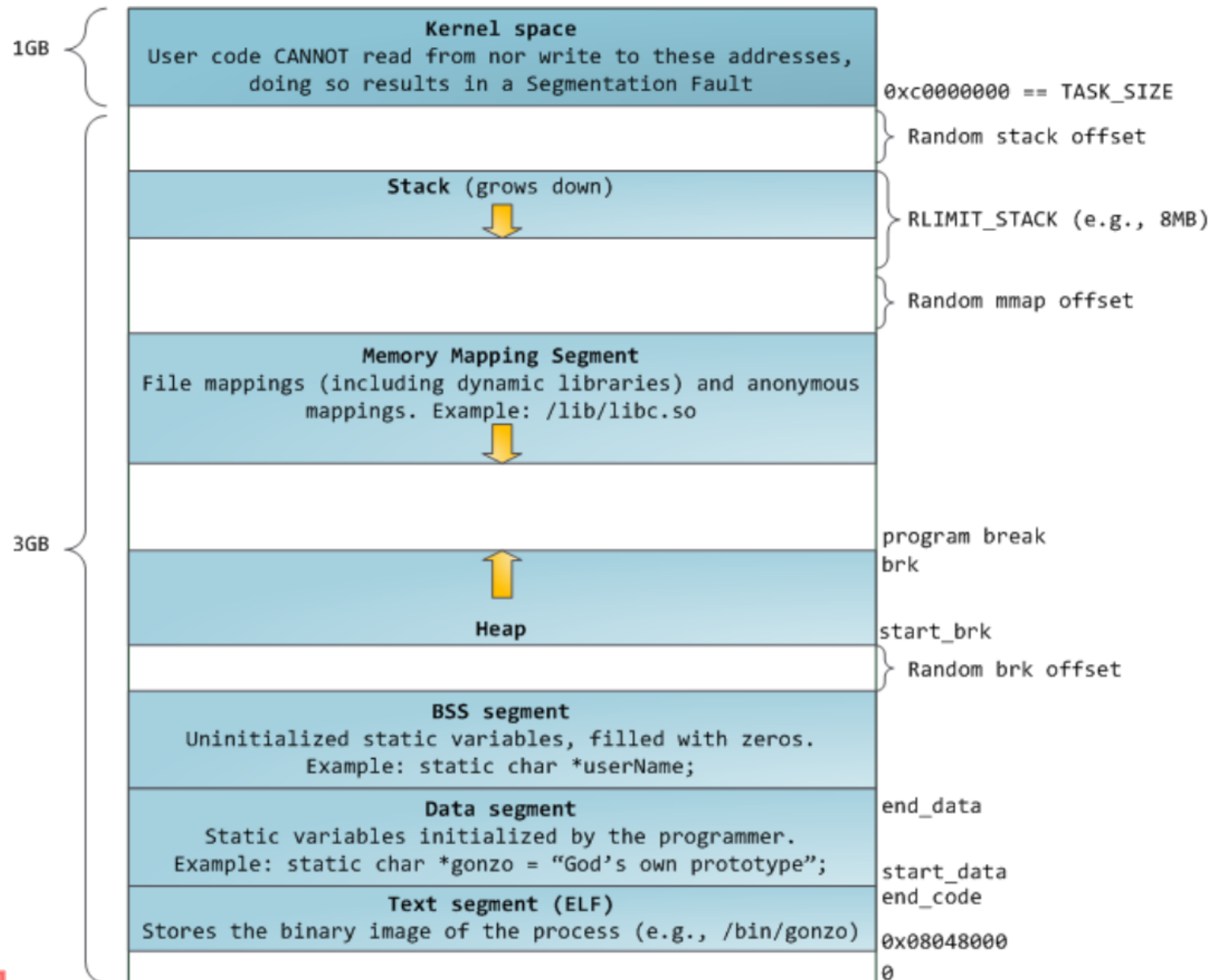
Agregarea resurselor folosite

Abstractizare peste CPU si memorie



# Spatiul de adrese al unui proces

vezi si pmap



<http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory>

# Tipuri de procese

interactive (foreground)

- interactioneaza cu utilizatorul

neinteractive (background, batch)

- nu interactioneaza cu utilizatorul
- servicii, daemoni

I/O intensive (I/O bound)

CPU intensive (CPU bound)

# Atributele unui proces

*Process Control Block (PCB)*

*Resursele unui proces*

*Ierarhia de procese*

*Starea unui proces*

# Process Control Block

*PCB*

Structura ce descrie procesul si attributele acestuia

- struct `task_struct` in nucleul Linux
- `EPROCESS/KPROCESS` in nucleul Windows

PID

spatiul de adrese (zonele de memorie)

tabela de descriptori de fisier

masca de semnale

informatii de securitate (user id, group id, capabilitati)

referinte la alte procese

informatii de monitorizare si stare

# Resursele unui proces

resursele fizice abstractizate in resurse logice

- disc - fisiere
- retea - socketi

operatii cu resurse: deschidere, inchidere, citire, scriere

resursele au un identificator (descriptor) si o stare

private sau partajate cu alte procese

# Descriptorii de resurse

file descriptori, handle-uri, tabele

unici la nivelul unui proces (namespace)

- descriptorul 4 din procesul P1 refera altceva in procesul P2
- poate fi valid intr-un proces dar nevalid in altul

# Ierarhia de procese

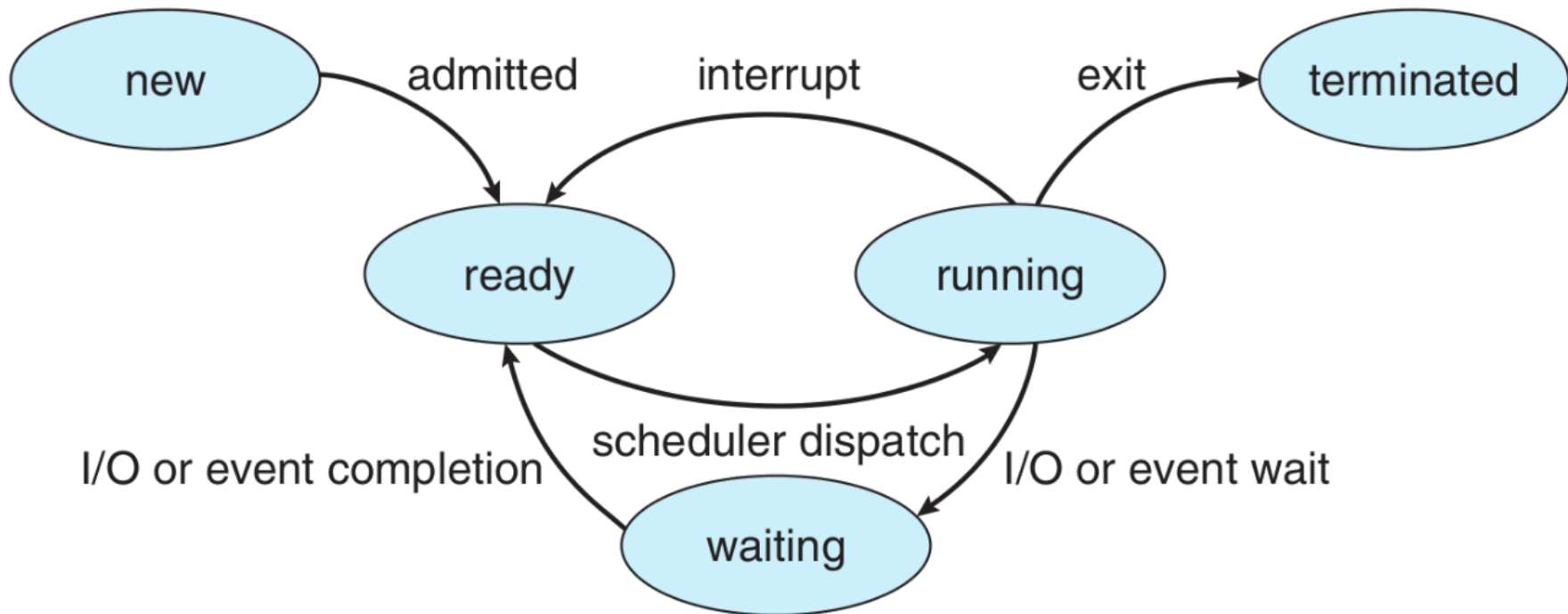
un proces are un proces parinte  
un proces are mai multe procese copil

in Unix, init este la radacina ierarhiei de procese

un proces are un PID si un PPID (parent process ID)

pstree, ps -H, Process Explorer

# Starea unui proces



OSCE, Chapter 3, pg. 101, Figure 3.2



# Planificarea proceselor

*Paralelism la executie*  
*Tranzitii intre stari*  
*Schimbare de context*  
*Planificatorul de procese*

# Paralelism la nivel de SO

mai multe procese la nivelul SO  
SO mediaza accesul proceselor la procesor

cu adevarat simultan ruleaza N procese

- N = numar de procesoare

pseudoparalelism: procesele sunt schimbate rapid

paralelism efectiv: sisteme multiprocesor, multicore

# Multitasking

procesele sunt schimbate rapid de SO

- cuanta de timp de rulare (time slice)
- la expirarea cuantei, procesul este inlocuit de pe procesor

permite prezenta simultana a mai multor utilizatori

nu este garantata executia completa a unei secvente

- un proces va fi intrerupt la expirarea cuantei

expirarea cuantei declanseaza inlocuirea (schimbare de context)

# Tranzitii intre starile unui proces

running -> ready

- expirarea cuantei (time slice)
- proces prioritar

ready -> running

- CPU liber
- primul selectat de planificator

running -> waiting

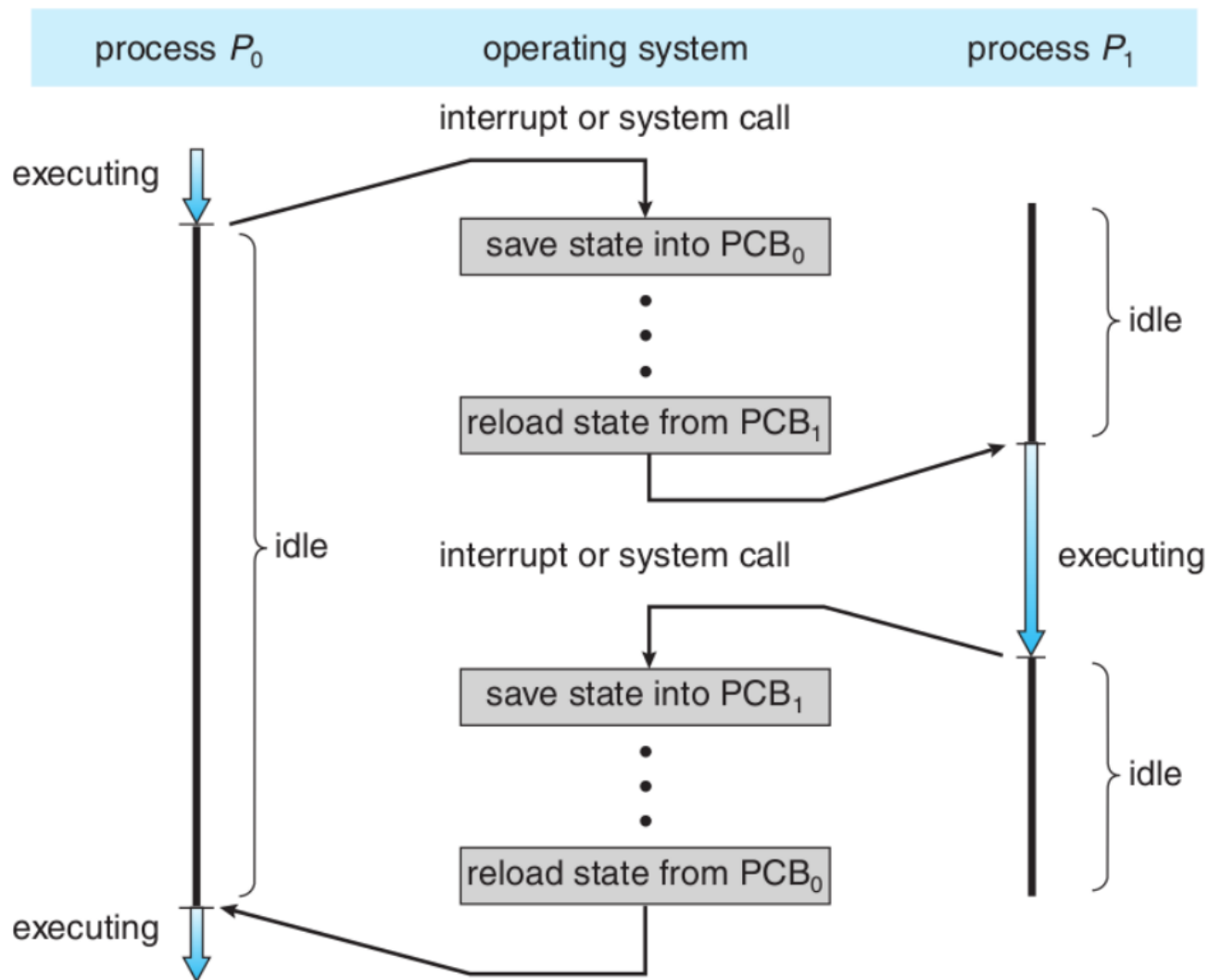
- asteptat dupa eveniment, dispozitiv, lock

waiting -> ready

- rezolvat eveniment de asteptare

# Schimbare de context

*inlocuirea unui proces cu un alt proces pe procesor (running)*



# Planificarea proceselor

*scheduling, scheduler*

subsistem al nucleului SO (process scheduling, process management)

planificarea proceselor pe procesoare

- alegerea celui mai potrivit proces din coada ready
- declansat de o schimbare de context

cele mai potrivite procese

- prioritati
- euristici specifice
- algoritmi de planificare

criterii de planificare

- eficienta, performanta (throughput)
- echitate (fairness)

*mai multe in cursul 4 - Planificarea executiei*

# Operatii cu procese

*Crearea unui proces*  
*Incheierea unui proces*  
*Terminarea unui proces*  
*Asteptarea unui proces*

# Crearea unui proces

dintr-un proces existent

un proces existent (parinte) creeaza un nou proces (copil)

procesul nou are la baza un program/executabil  
executabilului i se asociaza resurse dinamice si devine proces  
creat cu ajutorul loader-ului (ld-linux.so pe Linux)

input: executabil si argumente

output: un nou proces, resurse noi, un nou PID

Cine este procesul parinte la rulara comenzii ls?



# fork si exec in Unix

fork creeaza un proces identic cu primul (o clona)

- procesul parinte si copil sunt identice
- in general au multe resurse partajate
- input: nimic
- output: un nou proces

exec "transforma" procesul copil pornind de la un executabil

- resursele sunt acum unice fiecarui proces
- procesul "transformat" are un alt spatiu de adresa
- acum se invoca loader-ul
- PID-ul procesului nu se schimba
- input: executabil cu argumente
- output: actualizarea resurselor procesului

# Incheierea unui proces

procesul isi incheie executia

- decizie a procesului
- ajunge la finalul programului (finele lui main)
- se apeleaza `exit()`

input: valoare de retur

output: proces incheiat, nu mai exista in sistem

# Terminarea unui proces

procesul este terminat de alta entitate

- nucleul SO
- un alt proces
  - primirea unui semnal (Unix)
  - primirea unei exceptii (Windows)

CTRL+C, End Task, Segmentation Fault, Division by Zero etc.

- transmiterea unui semnal sau a unei exceptii

input: PID, cauza

output: proces inexistent in sistem

# Asteptarea unui proces

procesele isi sincronizeaza executia

- un proces asteapta incheierea altui proces
- "waiting for another process"

dupa incheierea unui proces, exista informatii remanente

- modul in care s-a incheiat procesul (valoarea de retur)

functiile de asteptare ofera informatiile remanente

input: PID-ul procesului asteptat

output: informatii remanente dupa incheierea procesului

# Procese zombie si orfane

un proces copil este asteptat de un proces parinte

probleme:

- parintele moare
- copilul moare inainte sa il astepte parintele

proces zombie

- proces care a murit inainte sa fie asteptat
- ramane in starea zombie pana la asteptare (wait)
- este nevoie de informatie reziduala (valoarea de retur)

proces orfan

- proces caruia i-a murit parintele
- procesul este adoptat de init

# API de procese

*shell*  
*ANSI C*  
*POSIX*  
*Win32*  
*Python*  
*Java*

# Lucrul cu procese in shell

creare: ls, mv, top, ifconfig

incheiere: dupa incheierea comenzii

asteptare: implicit, shell-ul se blocheaza

- operatorul & (run in background)

terminare: kill sau CTRL+C, CTRL+Z, CTRL+\

Care este relatia de rudenie intre shell si procesele create?  
Ce face comanda "exec ls"?

# Lucrul cu procese in ANSI C

```
system("ps -u student");
```

```
FILE *f = popen("ps -u student", "r");  
pclose(f);
```

- nu e ANSI, e POSIX, dar are suport si pe Windows



# Lucrul cu procese in POSIX

## *fork si exec*

```
pid = fork();
switch (pid) {
    case -1: /* fork failed */
        perror("fork");
        exit(EXIT_FAILURE);
    case 0: /* procesul copil */
        execlp("/usr/bin/ps", "ps", "-u", "student", NULL);
    default: /* procesul printe */
        printf("Created process with pid %d\n", pid);
}
```

fork e apelat o data si se intoarce de doua ori

- intoarce 0 in copil
- PID-ul procesului copil in parinte
- Cum afla procesul copil PID-ul parintelui?
- Ce face procesul parinte cu PID-ul copilului?

## Lucrul cu procese in POSIX (2)

```
exit(EXIT_SUCCESS);
```

```
kill($PID, SIGTERM);
```

```
pid = wait(&status);
```

```
pid = waitpid(-1, &status, 0); /* echivalent */
```

# Lucrul cu procese in Win32

```
PROCESS_INFORMATION pi;  
CreateProcess(NULL, "notepad", NULL, NULL, ..., &pi);
```

- nu exista fork si exec (separat) pe Windows

```
exit(0);
```

```
TerminateProcess(pi.hProcess, 123);
```

```
WaitForSingleObject(pi.hProcess, INFINITE);  
GetExitCodeProcess(pi.hProcess, &retValue);
```

# Lucrul cu procese in Python

*subprocess module*

```
subprocess.call(["ps", "-u", "student"])
```

```
p = subprocess.Popen(["ps", "-u", "student"],  
    shell=True, bufsize=bufsize,  
    stdin=PIPE, stdout=PIPE, stderr=PIPE, close_fds=True)  
(child_stdin,  
 child_stdout,  
 child_stderr) = (p.stdin, p.stdout, p.stderr)
```

# Lucrul cu procese in Java

## *ProcessBuilder*

```
ProcessBuilder builder = new ProcessBuilder("ps", "-u", "student");
```

```
Process p = builder.start();
```

```
InputStream is = p.getInputStream();
```

```
OutputStream os = p.getOutputStream();
```

echivalent cu subprocess (Python) si popen()

# Comunicarea interproces

IPC (Inter-Process Communication)

*Semnale*

*Pipe-uri*

*Memorie partajata*

*Message passing*

*Socketi*

# Semnale

doar pe Unix; pe Windows - exceptii

notificarea unei situatii neprevazute

un semnal este transmis de un proces sau de kernel altui proces

identificat printr-un numar  
kill -l

handler de tratare a semnalului

- de multe ori, in mod implicit, handler-ul este "omoara procesul"

# Probleme in folosirea semnalelor

semnalele vin asincron fata de executia procesului

- handlerele de semnal sunt rulate asincron
- procesul poate fi intrerupt in puncte aleatoare
- conditii de cursa (race conditions)

in semnale nu se pot apela functii reentrante

- malloc
- definitii pentru reentrante in cursul 8 - Thread-uri

apelurile blocante (read, write) pot fi intrerupte

- trebuie reluat apelul
- `errno == EINTR`



# Pipe-uri

comunicare simpla intre procese inrudite  
pipe = buffer la nivelul nucleului

operatii de open, read, write, close

un pipe = 2 file descriptori (unul de citire, unul de scriere)

- un proces scrie un proces citeste

implementarea operatorului | (pipe)

pipe-uri cu nume (FIFO)

- intrare in sistemul de fisiere
- comunicare intre procese neinrudite

# Memorie partajata

bucati din spatiul de adrese al mai multor procese refera acelasi spatiu de memorie fizica (RAM)

accesul se face pe bază de pointeri (adrese)

- nu e nevoie de apeluri de sistem pentru acces

avantaj: rapida

dezavantaj: necesita sincronizare

# Message Passing

formatarea datelor ca mesaje cu sursa si destinatie  
multe biblioteci (in diverse limbaje) pentru message passing

util in medii distribuite

livrare directa procesului  
livrare indirecta intr-un mailslot

operatii de tipul send/receive  
• sincrone sau asincrone

# Socketi

comunicare in retea (BSD sockets)  
comunicare locala (Unix sockets)

socket API: socket, connect, listen, send, recv

comunicare de tip "teava": byte stream

# Cuvinte cheie

proces

program

ierarhia de procese

PCB

atribute de proces

starea unui proces

descriptori de resurse

multitasking

context switch

planificator

crearea unui proces

terminarea unui proces

asteptarea unui proces

fork

exec

CreateProcess

IPC

semnale

pipe-uri