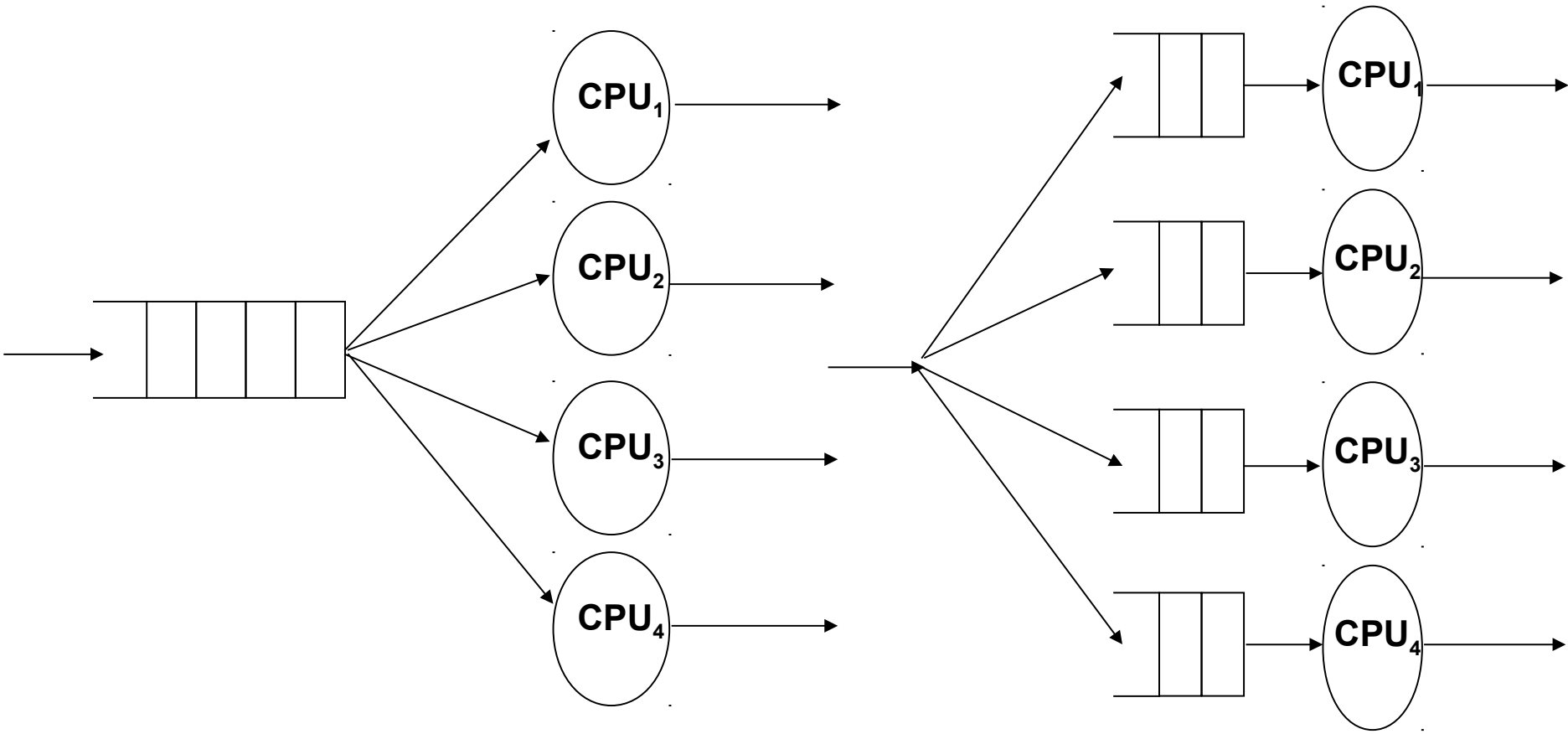
Sisteme de Operare

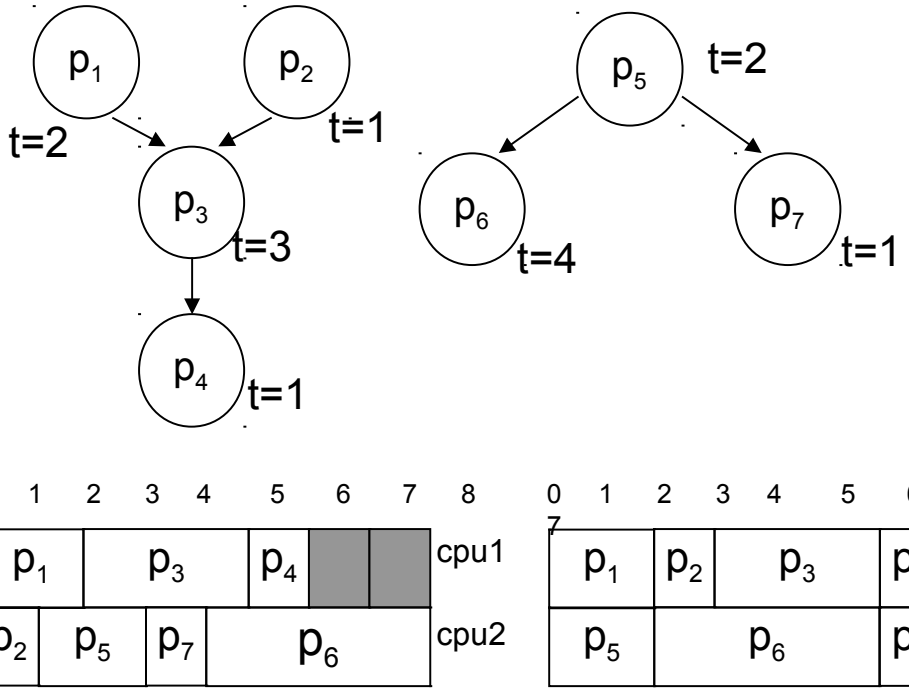
- Planificarea în sistemele multiprocesor
- Planificarea în sistemele de timp real
- Studii de caz
  - UNIX, Linux, Windows

Planificarea în sistemele multiprocesor



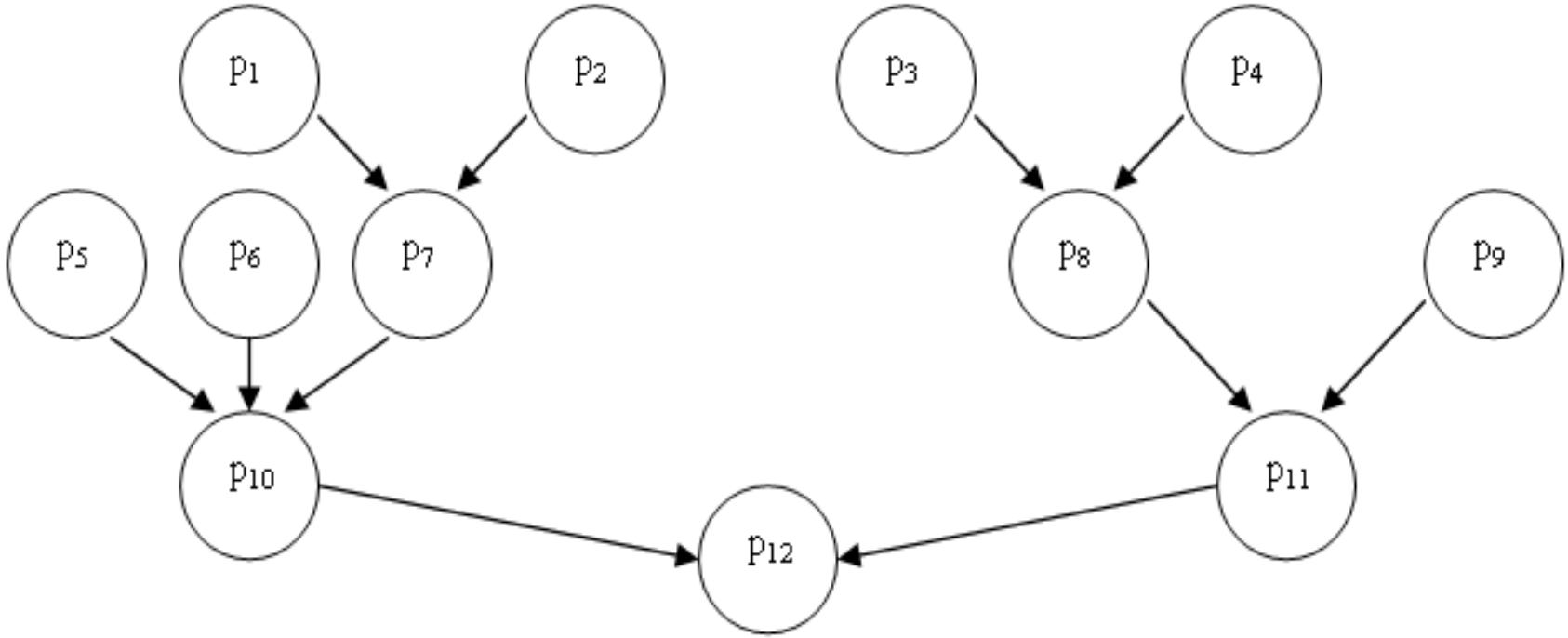
Echilibrarea încărcării

- În cazul planificării multiprocesor procesele din lista gata de execuţie formează un graf de precedenţă.
- Obiectivul planificării multiprocesor este optimizarea planificării pentru **n≥2 CPU** pentru **procese cu grafuri de precedenţă cu timpi de execuţie cunoscuţi**.



Graful de precedență al proceselor
Algoritm de planificare

- **Pas 1**: se aşteaptă ca un procesor să devină idle;
- **Pas 2**: se defineşte mulţimea **R** a proceselor pentru care fiecare predecesor şi-a terminat execuţia;
- **Pas 3**: se alege un **R'** $\subset$ **R** şi se găsesc **p** $\in$ **R'** astfel încât **length(p)** este maxim;
- **Pas 4**: se alege un **p** $\in$ **R'** din cei găsiţi anterior şi se atribuie procesorului idle;
- **Pas 5**: dacă nu mai sunt procese de planificat algoritmul s-a terminat; altfel goto **Pas 1**.



Planificarea pentru 3 procesoare este:

	0	1	2	3	4	5
cpu1	p_3	p_2	p_8	p_{10}	p_{12}	
cpu2	p_4	p_5	p_6	p_{11}		
cpu3	p_1	p_9	p_7			

- La momentul 0 :

  procesele p_1, p_2, p_3, p_4 sunt fără predecesori şi au length=3;

  procesele p_5, p_6, p_9 au length=2.

Planificarea proceselor în sistemele de timp real

- Din punct de vedere al importanţei deadline-ului sistemele de timp real se împart în:
  - **hard - real time**: task-urile trebuie terminate într-un anumit interval de timp.
  - **soft - real time**: sunt mai puţin restrictive şi impun prioritatea proceselor critice asupra celorlalte procese din sistem

Planificarea în sistemele de timp real - hard-real time

- În general când un proces este lansat în execuţie este specificat şi necesarul de timp pentru realizarea operaţiilor de I/O.
- Planificatorul poate accepta procesul, garantând că se va termina la timp sau îl poate respinge.
- Acest lucru se mai numeşte şi rezervarea resurselor:
  - planificatorul ştie exact cât va dura execuţia fiecărei funcţii sistem.
  - sunt construite pentru sisteme dedicate şi nu au funcţionalitatea unui sistem de uz general.

Planificarea în sistemele de timp real - soft-real time

- Adăugarea unei astfel de funcţionalităţi unui sistem de tip time-sharing poate duce la:
  - o alocare defectuoasă a resurselor,
  - întârzieri mari în rularea proceselor
  - apariţia fenomenului de "înfometare" pentru unele procese.
  - Implementarea acestei funcţionalităţi presupune o mare atenţie în proiectarea planificatorului.
    - sistemul trebuie să aibă o planificare bazată pe priorităţi
    - procesele de timp real trebuie să aibă prioritatea cea mai mare, prioritate care nu trebuie să scadă în timp.
    - lansarea proceselor trebuie să fie rapidă.

Preemptarea apelurilor sistem

- introducerea punctelor de preemptare în apelurile sistem de lungă durată
  - trebuie verificat dacă sunt procese cu prioritate mai mare care trebuie rulate.
  - pot fi plasate numai în locaţii sigure ale kernelului (unde nu se fac modificări ale structurilor de date ale kernelului).
- întreg kernelul să poată fi preemptat:
  - trebuie asigurat faptul că structurile de date ale kernelului sunt protejate prin utilizarea mecanismelor de sincronizare.
  - toate datele kernelului sunt protejate împotriva modificării lor de către procesele cu prioritate ridicată.

Clase de algoritmi de planificare

- Static table-driven
  - se încearcă realizarea întregii planificări (se determină când trebuie să fie executat un task).
- Static priority-driven preemptive
  - poate fi folosită planificarea bazată pe priorităţi.
- Dynamic planning-based
  - o soluţie ar fi crearea unei planificări care să conţină task-urile planificate anterior precum şi cele nou sosite.
- Dynamic best effort
  - când apare un proces nou, sistemul îi atribuie o prioritate bazată pe caracteristicile procesului.

Planificarea în funcție de deadline

- trebuie cunoscute diverse informaţii despre task:
  - ready time
  - starting deadline – timpul în care un proces trebuie lansat
  - timpul în care se termină procesul
  - timpul de rulare
  - cererile de resurse
  - prioritatea
  - structura procesului.

Studii de caz
UNIX

- Algoritmii de planificare sunt proiectaţi pentru a da un timp de răspuns bun proceselor utilizatorilor într-un sistem de tip time-sharing.
- Pentru SVR4, schemele de planificare includ şi componente ale sistemelor de timp real.
- Sistemul de operare UNIX implementează o listă multinivel cu feedback
  - În cozile de prioritate este folosit algoritmul round-robin (RR) cu o cuantă de 1 secundă.
  - Dacă un proces nu se termină sau nu se blochează într-o secundă este preemptat.
  - Prioritatea este în funcţie de tipul procesului şi informaţiile din perioadele de rulare anterioare.

Studii de caz UNIX

$$CPU_j = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i-1)}{2} + nice_j$$

- **$CPU_j(i)$** = utilizarea procesorului de către procesul **j** în intervalul de timp **i**.
- **$P_j(i)$** = prioritatea procesului **j** la începutul intervalului **i**. Valorile mici implică o prioritate mai mare
- **$Base_j$** = prioritate de bază a procesului **j**.
- **$nice_j$** = valoare la dispoziţia utilizatorului.

Studii de caz
UNIX

- Prioritatea fiecărui proces este recalculată la fiecare secundă sau de fiecare dată când apare un proces nou.
- Scopul lui **$Base_j$** este împărţirea proceselor la cozile de prioritate.
- **$CPU_j(i)$** şi **$nice_j$** sunt folosite pentru a preveni migrarea proceselor dintr-o coadă în alte cozi de prioritate.
- Aceste **cozi de prioritate** sunt folosite pentru a optimiza accesul la dispozitivele de tip bloc (disk) şi permit sistemului de operare să răspundă rapid la apelurile sistem.

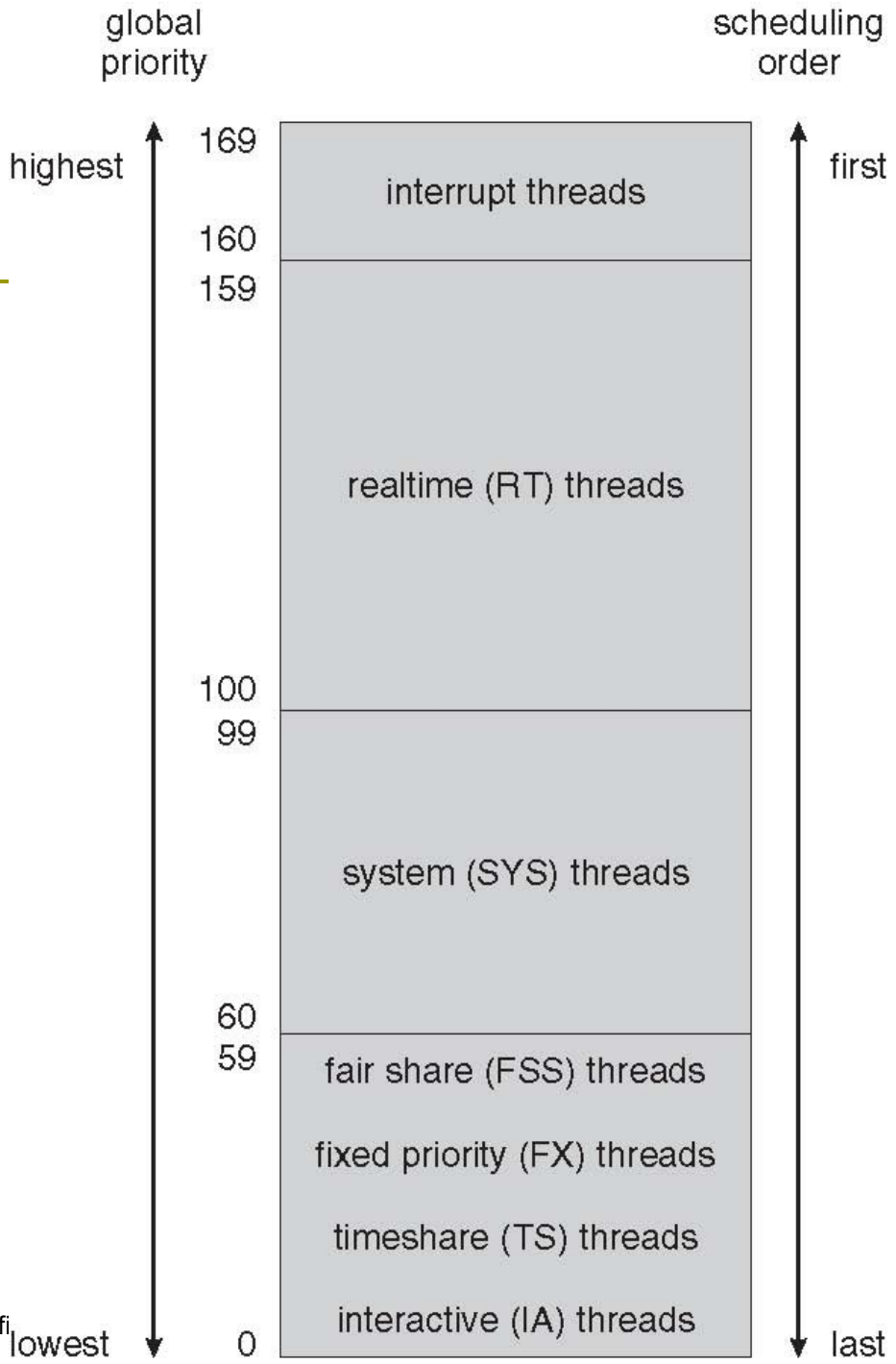
Studii de caz
UNIX

- În ordinea descrescătoare a priorităţii aceste **cozi de prioritate** sunt:
  - swap-ul;
  - block I/O device control;
  - lucrul cu fişiere;
  - character I/O device control;
  - procesele utilizator.
- Această ierarhie permite o utilizare eficientă a sistemului de I/O.
- În cadrul ultimei liste, a proceselor utilizator, utilizarea informaţiilor despre execuţiile anterioare penalizează procesele care consumă foarte mult timp de lucru cu sistemele de I/O.
- Împreună cu algoritmul RR, această strategie de planificare poate satisface cerinţele unui sistem de tip time-sharing.

Studii de caz
UNIX

- Există un set de **160 niveluri de prioritate** împărţite în trei clase:
  - **real time (159 ÷ 100)**:
    - Procesele cu aceste niveluri de prioritate sunt selectate pentru execuţie înaintea kernelului sau a oricărui proces.
    - pot folosi punctele de preemptare pentru a planifica procesele kernelului şi procesele utilizator.
  - **kernel (99 ÷ 60)**:
    - Aceste procese sunt selectate pentru rulare înaintea proceselor de tip time-sharing, dar trebuie să acorde prioritate proceselor de timp real.
  - **time-shared (59 ÷ 0)**:
    - Sunt procesele cu prioritatea cea mai mică destinate proceselor utilizator, altele decât cele de timp real.
- Sunt introduse punctele de preemptare în kernel fapt ce permite întreruperea kernelului dacă datele sunt în siguranţă, iar sincronizarea accesului la resurse se face cu ajutorul semafoarelor.

Studii de caz UNIX - Solaris

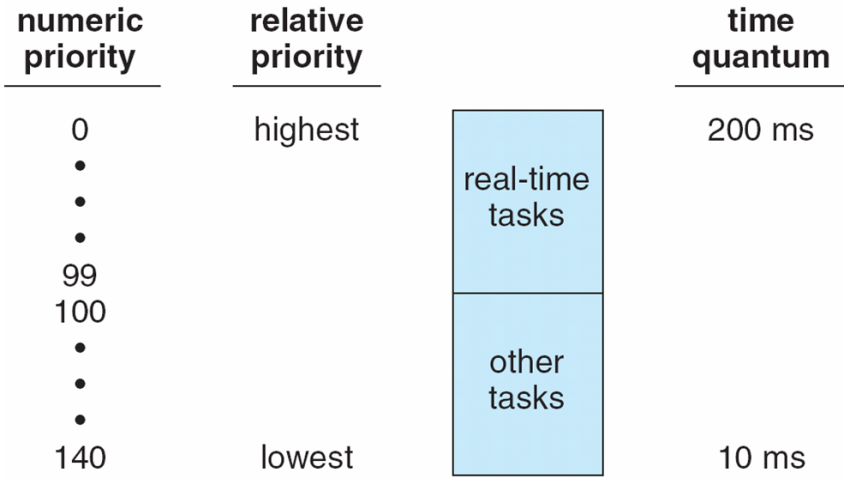


Studii de caz
Linux

- Kernel 1.2 – planificatorul utilizează o coadă circulară şi o politică de planificare round-robin
- Kernel 2.2 – se introduc clasele de planificare
  - Politici diferite pentru următoarele tipuri de task-uri: real-time, non-preemptible şi non-real-time
  - Suport pentru symmetric multiprocessing (SMP).
- Kernel 2.4 - **O(n) scheduler**
- Kernle 2.6 -2.6.23 - **O(1) scheduler**
  - Timp de planificare constant indiferent de numărul de task-uri
- Kernel post-2.6.23 - **CFS (Completely Fair Scheduler)**
  - complexitatea algoritmului **O(log N)**
    - Alegerea unui task se face în timp constant
    - Reinserarea dureaza O(log N)

Studii de caz Linux

- O(1) scheduler
  - Două clase de priorităţi:
    - real-time (valoarea nice: 0-99)
      - cuanta de 200 ms
    - time-sharing (valoarea nice: 100-140)
      - cuanta de 10 ms
- Kernel pre-2.6
  - foloseste o lista multinivel cu feedback pentru implementarea cozii de procese în starea ready
- Kernel post-2.6.23
  - Foloseste red-black tree pentru implementarea cozii de procese în starea ready



Studii de caz
Linux

- clase de planificare:
  - SCHED_FIFO (FIFO real-time)
    - nu este întrerupt decât dacă a apărut un proces cu prioritate mai mare, procesul se blochează sau eliberează singur procesorul
    - dacă este întrerupt, procesul este pus în coada de aşteptare asociată priorităţii lui.
    - Când un proces devine ready şi are o prioritate mai mare decât procesul care se execută, procesul curent este preemptat şi este executat cel cu prioritatea mai mare. Dacă sunt mai multe procese cu prioritate mare, este ales procesul care a aşteptat cel mai mult.
- SCHED_RR (round-robin real-time)
  - la sfârşitul fiecărei cuante un alt proces cu o prioritate mai mare sau egală este planificat
- SCHED_OTHER (non-real-time)
  - folosit atunci când există procese gata de execuţie şi nu intră în categoria proceselor real-time

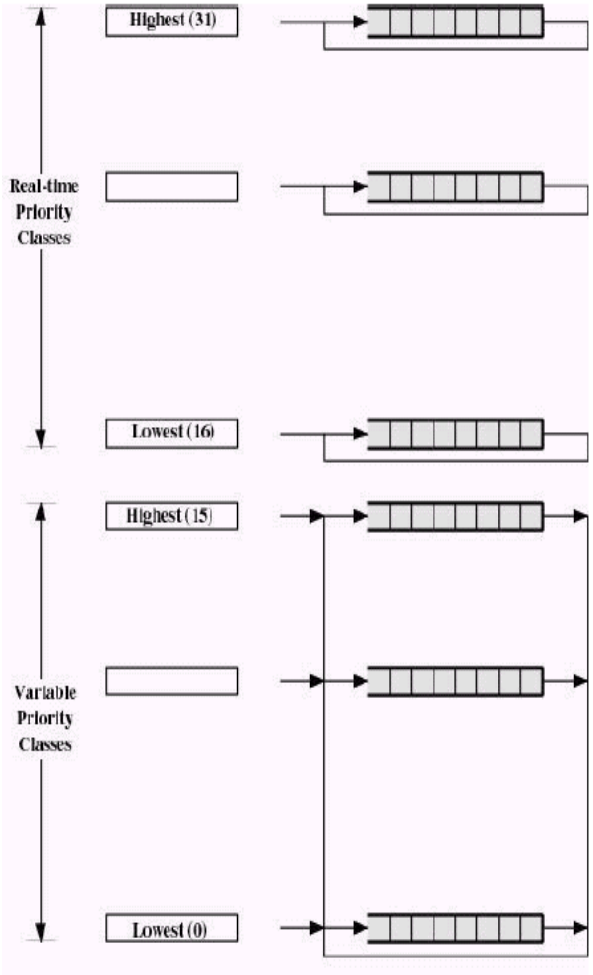
Studii de caz
Linux

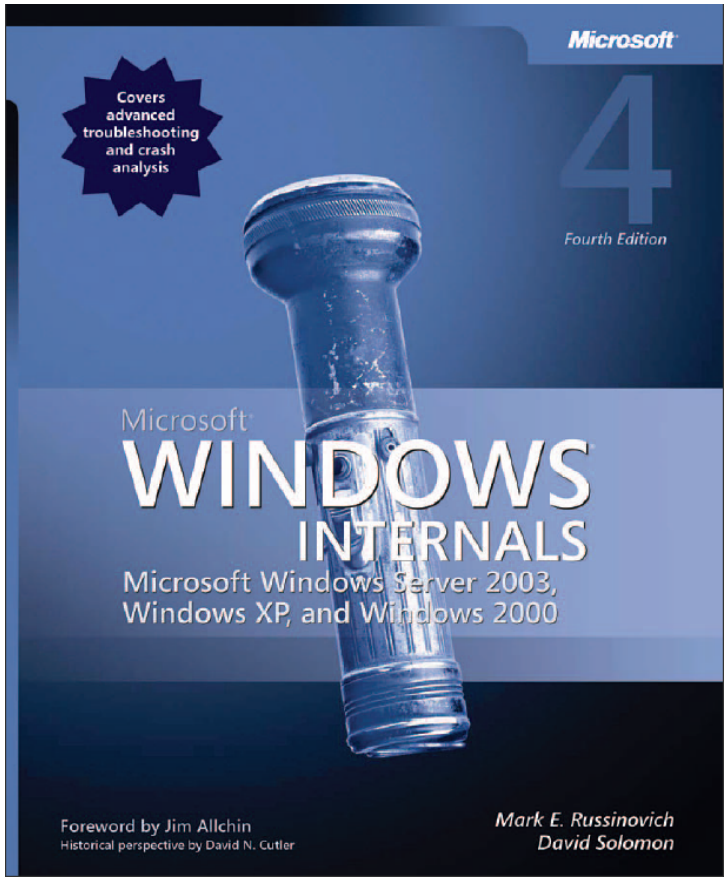
- Understanding the Linux Kernel
  - http://oreilly.com/catalog/linuxkernel/chapter/ch10.html
- Inside the Linux scheduler
  - http://www.ibm.com/developerworks/linux/library/l-scheduler/
- Inside the Linux 2.6 Completely Fair Scheduler
  - http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/
- http://en.wikipedia.org/wiki/Completely_Fair_Scheduler
- Linux 2.6.8.1 CPU Scheduler Paper
  - http://joshaas.net/linux/linux_cpu_scheduler.pdf
  - http://en.wikipedia.org/wiki/Scheduling_%28computing%29

Studii de caz Windows2000 si XP

- două clase de priorităţi fiecare cu 16 niveluri de prioritate, deci în total 32 de niveluri de prioritate:
  - Real time (31-16) : toate procesele au prioritate fixă, care nu se modifică în timpul rulării
  - Variable (15 – 1): prioritatea procesului se poate modifica în timpul rulării şi creşte pe măsură ce procesul este blocat şi scade când este executat în cuanta sa de timp.
- Un thread ce ruleaza cu prioritatea 0 este folosit pentru managementul memoriei
- În cadrul fiecărui nivel de prioritate este folosit algoritmul Round-robin.

Studii de caz Windows2000 si XP





- Cozile de priorităţi pentru lansarea proceselor la Windows 2000 şi XP