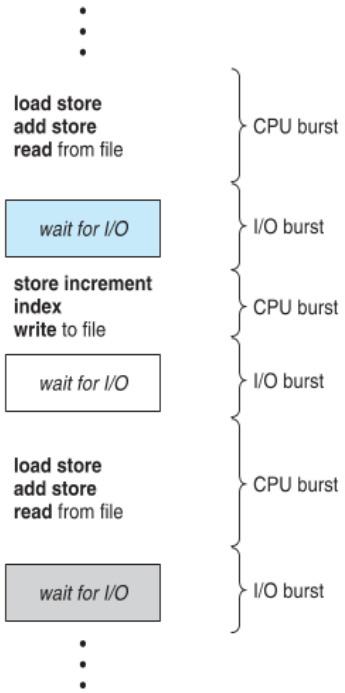
Tutoriat 7

CPU Scheduling

1. Concepte

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern



CPU Burst

It is the amount of time a process uses the CPU until it starts waiting for some input or is interrupted by some other process.

I/O Burst or Input Output burst

It is the amount of time a process waits for input-output before needing CPU time.

Preemptive => un proces se poate întrerupe oricând
Non-preemptive => un proces nu se poate scoate de pe procesor până nu termină

In Preemptive scheduling, dacă un proces cu prioritate mai mare decat procesul curent vrea sa se execute, procesul cu prioritate mai mica se întrerupe din execuție și pe procesor intra noul proces.
In Non-preemptive scheduling, odată ce un proces a intrat pe procesor rămâne acolo până când își termină execuția sau trece în starea de așteptare.

2. Criterii de scheduling

- CPU utilization - keep the CPU as busy as possible (pt optimizare trebuie să -> max)
- Throughput - # of processes that complete their execution per time unit (-> max)
- Turnaround time - amount of time to execute a particular process (-> min)
- Waiting time - amount of time a process has been waiting in the ready queue (-> min)
- Response time - amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment) (-> min) (E mai mult pt UI de ex)

3. Algoritmul FCFS (First-Come, First-Served)

First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:

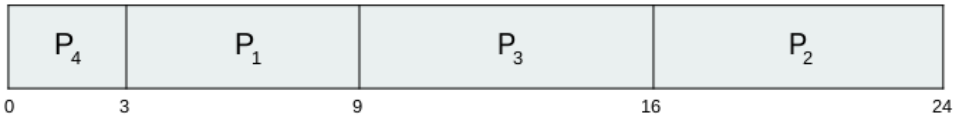
- Waiting time for P_1 = 6; P_2 = 0; P_3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

4. Algoritmul SJF (Shortest-Job-First)

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Upside: **Timpul mediu de așteptare este minim.**
Downside: Dificultatea determinării lungimii următorului CPU request.

Exercitiu: Scrieți valoarea de average waiting time a execuției proceselor de mai jos pe un singur procesor aplicand algoritmul SJF:

Process	Burst time
P1	14
P2	3
P3	9
P4	5

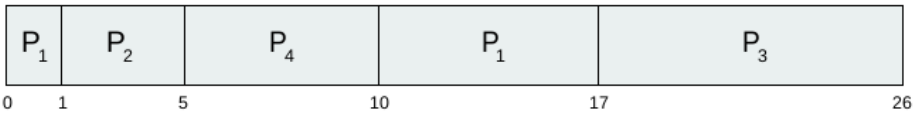
Raspuns:
P1 -> 3(P2) + 5(P4) + 9(P3) = 17
P2 -> 0
P3 -> 3 + 5 = 8
P4 -> 3
=> Average waiting time = (17 + 0 + 8 + 3) / 4 = 7

5. Shortest-remaining-time-first (SJF + preemption)

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

OBJ

Intra pe procesor procesul care are cel mai mic timp ramas.
Dacă la un moment dat intra în coada un proces cu timp mai mic, atunci e bagat pe procesor.

6. Priority Scheduling

Priority scheduling - fiecare proces are o prioritate. De exemplu, [-20, -1] pentru kernel si [0, 20] pentru userland, unde valoarea cea mai mica reprezinta prioritatea cea mai mare.
Problema cu aceasta abordare este faptul că poate apărea starvation deoarece este posibil ca procesele cu prioritate mica sa nu intre la execuție.
Soluția este aging (i.e. creșterea priorității în timp).

*Exista funcția de sistem nice care schimba prioritatea unui proces.

7. Algoritmul Round Robin

Fiecare proces se executa un timp **q,** dupa care (daca nu a terminat) e intrerupt, scos de pe procesor și pus la capătul cozii.

Daca **q** e prea mic, se face context switch foarte des ceea ce nu e bine.
Daca **q** e prea mare, algoritmul devine un FIFO.

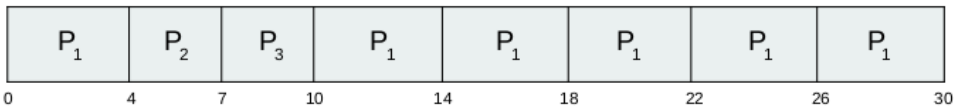
q este in general in intervalul [10ms, 100ms]
q ar trebui sa fie mare în comparatie cu timpul necesar context switch-ului
(care este < 10 usec)

Observatii:
Are un response time mai bun decat SJF.
În general are timp mediu de **turnaround** mai mare decat SJF.

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



8. Multilevel queue

Multilevel Queue - mai multe cozi care nu au acelasi algoritm. De exemplu avem 3 cozi: prima cu un RR cu un q1, a doua cu un RR cu alt q2, si a treia cu FCFS si un proces e "promovat" în coada următoare dacă nu a și-a terminat execuția