**Chapter 6**

CPU ütemezés – 9. előadás

Alapvetések

* Gépidő és I/O égetés miatt jött a Multitasking ötlete
* Csakis multiprogramozottsággal érhető el optimális CPU kihasználtság
  + Ára van 🡪 Folyamatok közti váltás ideje
* Első 8 miliszekundomban van a folyamatok legintenzívebb része
  + Utána csak vár, nem effektív
  + Megoldás 🡪 8 ms 🡪 Másik folyamat 🡪 8 ms 🡪 Másik folyamat
  + Lényegében ez a CPU ütemezés

Folyamatok 4 állapota

* Waiting
* Ready to run
* Running
* Terminated

Ütemező

* Rövidtávú ütemezőröl beszélünk leginkább
  + Választ a ready queue-ban lévő folyamatok közül, és ad neki gépidőt
    - A ready queue sokféle sorrendben lehet
* Mit tehet az ütemező?
  + 1. Running 🡪 Waiting
  + 2. Running 🡪Ready to run
  + 3. Waiting 🡪 Ready tor un
  + 4. Running 🡪 Terminated
* Nem preemptív ütemezőnél csak az 1. és 4. állapot van
* Preemptív ütemezésnél mind a 4, mivel át kell gondolni a
  + Közös adathozzáférést
  + Preemption kernel módban
  + Megszakítások fontos OS folyamatok közben
* Ma minden ütemező preemptív
  + Kapsz gépidőt, bármikor elvehető
  + Windows 95-nál váltott a Microsoft preemptívre
  + Nagyon fontos a reszponzivitás 🡪 A felhasználó türelmetlen
* Kell valami ami elvégzi a váltást két folyamat között, a context switchet
  + Elmentés, másik betöltése
  + Dispatcher/Diszpécser

Dispatcher

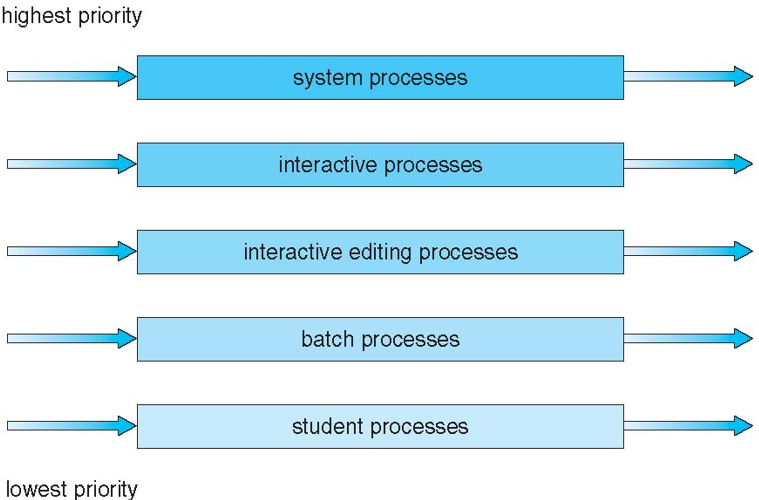
* Lehetővé teszi a CPU-nak a
  + Kontextus váltást
  + User mode váltást
  + User programban a megfelelő helyre ugrást, hogy újraindíthassa a programot
* Előző folyamat mentése, másik betöltése időbe kerül 🡪 Dispatch latency
  + A multiprogramozottság ára

Ütemező szempontjai

* Processzor kihasználtsága 🡪 Maximalizálni
* Kibocsátóképesség /Throughput 🡪 Maximalizálni
  + Mennyi ciklust végez el, mekkora az időablak
* Idő 🡪 Minimalizálni
  + Megfordulási 🡪 Egy adott folyamat KIVÉGZÉSE
  + Várakozási idő 🡪 Mennyit vár egy folyamat a ready queue-ban
  + Válasz idő 🡪 A kéréstől az első válaszig eltelt idő, nem feltétlenül output
* Mind egyszerre nem lehet tökéletes

Ütemező típusok

* First Come First Served – FCFS
  + Sorban, szekvenciálisan ahogy beérkeznek 🡪 Feltorlódás
  + Konvoj effektus 🡪 Egy rövid, gyors folyamat beragadva egy hosszú folyamat mögé
  + Magas várakozási idő
  + Lehet jó is, de nem igazán
* Shortest Job First
  + Legkevesebb időt igénybevevő feladatok először
  + Minimum átlag várakozási idő
  + Konvoj hatás lehetősége
  + Becsülgeti a következő CPU burst hosszát, aszerint osztja be
  + Preemptív változat 🡪 Shortest remaining time first
    - We likey
* Priority Scheduling
  + Magas prio előre, alacsony hátra
  + Jó lehet, de probléma 🡪 Starvation
  + Starvation 🡪 Lehet az alacsony priojú feladatok soha nem futnak le
    - Megoldás 🡪 Aging 🡪 Idő elteltével a ready queue-ban lévő folyamatok prioja növekszik
* Round Robin
  + PC-knél ez a befutó
  + Bevezet egy időablakot 🡪 Időquantum 🡪 10 ms < Quantum < 100 ms
    - Context Switch nagyjából 1 ms, tehát marad 9 ms minimum gépidő
    - Az 1 ms a multitasking ára
  + Amint lejár a quantum fogja a következő folyamatot

Multilevel queue

* Két ready queue-re bomlik
  + Háttérfolyamatok
  + Előtérben lévő folyamatok
  + Egy adott folyamat csak 1-ben lehet,   
    ott permanensen
* Előtér
  + Ineraktív alapvetően
  + Round Robin
  + 80% gépidő
* Háttér
  + Szolgáltatások, rendszerfolyamatok
  + Nem interaktálnak a userrel
  + Bármi lehet FCFS/SJF
  + 20 % gépidő

Multilevel feedback queue

* 3 külön queue
  + Q0 🡪 8ms
  + Q1 🡪 16 ms
  + Q2 🡪 FCFS
* Minden folyamat bekerül először Q0-ba, ha nem végez 8 ms alatt, megy tovább Q1-be, ha nem végez az az ottani +16 ms alatt sem, akkor megy Q2-be ahol sima FCFS van
* Ha elég egy folyamatnak Q1, akkor oda is tér majd vissza

Eddig csak folyamatütemezésről beszéltünk

Szálütemezés 🡪 Nice to have, elég a folyamat

* Same but different mint a folyamatkezelés
* User és Kernel level threadek

Multiple Processor Scheduling

* Több processzornál komplexebb az ütemezés
* CPU-k közötti mozgatás 🡪 Nem effektív, gépidőt vesz el
* Processzor affinitiás 🡪 Mennyire „ragaszkodjon” egy folyamat a processzorhoz, amin fut
  + Általában folyamatoknál ez az érték nagy , hogy ne tegye át másik procira a folyamatot, ezzel időt veszítve
  + Soft affinitás, hard affinitás
  + Folyamat egy procin belül szeretne maradni 🡪 nem mindig megoldható
* Processing típusok
  + Homogén processzorok🡪 Egy multiprocesszoron belül
  + Assymetric multiprocessing
    - Master CPU-k és Slave CPU-k
  + Symmetric Multiprocessing – SMP
    - CPU-k egyenrangúak
* Load Balancing
  + SMP esetén összes processzornak kell „adni feladatot”, hogy hatékony legyen
  + Load Balancing 🡪 Megpróbálja elosztani a terhet CPU-k között
  + Pull migration
    - Ha vannak vegetatív magok/processzorok, áthúzza a terhelt prociról
    - Békés
  + Push migration
    - Terhelt processzorról letolja a terhelést
    - Nem a legbékésebb

Multicore Processzorok

* Egy fizikai chipen belül több processzor
* Nem olyan hatékony 2 mag mint 2 processzor
  + Elveszted a plusz Cache-t
  + Az ALU-t 🡪 Aritmetikai és Logikai egység, kinda fontos ngl
* Mindenképpen javulás egy maghoz képest, viszont közel nem lineáris a javulás

Valósidejű Processzorütemezés

* X időn belül kell valami response/feedback a folyamattól
  + Nincs olyan hogy szimplán „lefagy”
  + Ha nem készül el időn belül, hibának veszi, elkezdi kiértékelni, javítani
    - Hibakezelő
* PC-kre egyáltalán nem jellemző