Előadás_#03.

1. Ütemezés

   Tekintsük át, hogy eddig minek a kapcsán merült fel ütemezés. Tulajdonképpen minden olyan lépés, ami állapot átmeneti változást eredményez, egyben jó eséllyel ütemezési feladatot is jelent. Minden esetben ütemezésre van szükség, ha egy (hardveres vagy szoftveres) erőforrásra több folyamat is benyújtotta igényét. Az ütemezés az ütemezendő események sorrendjének adott algoritmus szerinti meghatározását jelenti, vagyis azt, hogy melyik folyamat kapja meg az erőforrást. Az ütemezők neve vagy közvetlenül az ellátott funkcióra, vagy az ütemezés gyakoriságára utal.

   - Rövid távú ütemező (CPU ütemező)

     A végrehajtás alatt álló folyamatok között képes váltani. Ütemezése minden CPU löket után lefut, tehát gyorsnak kell lennie, ezért a rövidtávú ütemező a kernel része, állandóan a memóriában tartózkodik. Egy processzor mag esetében nem kérdés, hogy mindent ezzel a rendelkezésre álló egy maggal kell megoldani, de több processzor illetve több processzormag esetén az egyik lehetséges megoldás, hogy minden mag a saját ütemezéseiért is felelős, illetve létezik olyan megoldás, amikor az egyik mag csak az össze mag ütemezését végzi, így a többi mag hatékonyabban tud a folyamatokkal foglalkozni.

     Az ütemezési algoritmus lehet:

     - Preemptív

       Ebben az estben a kiosztott futási jog elvehető az éppen futó folyamattól, aminek hatására a folyamat futásra kész állapotba kerül. Azért ebbe az állapotba, mert csak a CPU, mint erőforrás hiányzik neki. (A Windows NT alapú rendszerek preemptív ütemezést használnak, amit később részletesen tárgyalunk.)

     - Nem preemptív

       Ebben az esetben a kiosztott futási jog nem vehető el. Az éppen futó folyamat tehát vagy normál módon befejeződik, vagy önmaga mond le a futási jogáról. Ütemezés tehát futás közben csak akkor következik be, amikor a folyamat maga erre utasítást ad. (A UNIX például kernel módban nem preemptív, de user módban preemptív – ezt később részletesen tárgyaljuk.)

Az ütemezést kiváltó esemény lehet, amikor:

- a futó folyamat befejeződik,
- a futó folyamat várakozni kényszerül,
- a futó folyamat önként lemond a futási jogáról, vagy bármely okból az ütemező elveszi tőle a futási jogot,
- egy folyamat felébred és futásra késszé válik,
- illetve órajel alapú ütemezés esetén az időszelet letelte.

Homogén rendszerek esetén jellemző, hogy több CPU mag is ugyanazt a várakozási sort használja. Ebben az esetben az első szabaddá váló CPU mag kapja meg a soron következő folyamatot.
Heterogén rendszerek esetében (amikor több különböző architektúrájú és utasításkészletű CPU illetve mag együttműködéséről van szó) a folyamatok csak a nekik megfelelő utasításkészletű CPU-n tudnak futni.
Közös várakozási sorok esetében a következő megoldások alkalmazhatók:

- Szimmetrikus ütemezés
  Minden CPU mag külön ütemezőt futtat. A közös várakozási sorhoz való hozzáférést, azaz annak megosztott használatát hibamentesen kell biztosítani, például kölcsönös kizárás segítségével.
- Aszimmetrikus ütemezés
  Egyetlen kiválasztott CPU mag futtatja az ütemezőt, végzi a kiválasztást.

A CPU ütemezés algoritmusainak hatékonyságát több összehasonlító paraméterrel is megvizsgálhatjuk:

- CPU kihasználtság (CPU Utilization)
  Azt mutatja meg, hogy a CPU az idő mekkora százalékában foglalkozik a folyamatok utasításainak végrehajtásával. A CPU-nak nyilván időt kell szakítania magára az ütemezésre, rendszeradminisztrációra, illetve az is előfordulhat, hogy nincs futásra kész folyamat, azaz a CPU tétlen (Idle).
- Átbocsájtó képesség (Throughput)
  Azt mutatja meg, hogy adott időegység alatt az operációs rendszer mennyi programot (folyamat egységet) volt képes lefuttatni.
- Körülfordulási idő (Turnaround Time)
  Az egy munkára fordított teljes időt mutatja meg, azaz, hogy a munka mennyi idő alatt fejeződött be. Ez tehát a futási idő és az összes egyéb (várakozással, adminisztrációval, stb.) eltelt idő összessége.
- Várakozási idő (Waiting Time)

Azt mutatja meg, hogy egy munka összesen mennyi időt töltött várakozással. Több összetevője van, mint a futás megkezdése előtti várakozás (lásd hosszú távú ütemező), a futásra kész állapotban, valamint a felfüggesztett állapotokban töltött idő.

- Válaszidő (Response Time)
  Azt mutatja meg, hogy az operációs rendszer milyen gyorsan képes a felhasználói beavatkozásokra, igényekre reagálni.

- Közép távú ütemező
  Amikor az operatív memória szűk keresztmetszetté válik, azaz gyakorlatilag megtelik, bizonyos folyamatok háttértárra mozgatásával szabad memóriára tehetünk szert. Ennek a memória felszabadításnak és visszarendezésnek az ütemezését végzi a középtávú ütemező.
- Hosszú távú ütemező
  Az új folyamatok indításáról dönt, a folyamat indítást ütemezi. Követendő cél a CPU igényes és az I/O igényes folyamatok arányának megtartása. Relatíve ritkán fut, tehát nem kell szupergyorsnak lennie. Egy folyamat létrejötte után, megkapja az azonosítóját, valamint a legfontosabb erőforrását, a hozzá rendelt memóriaterületet. A folyamat a hosszú távú ütemező várakozási sorából akkor léphet át a rövid távú ütemező várakozási sorába, ha már minden szükséges perifériával rendelkezik, kivéve a CPU-t.
- Periféria ütemezők
  Egy-egy periféria kiszolgálási sorrendjéről dönt, általában érkezési sorrend (FIFO pl. gépelés), vagy a prioritás (pl. diszk művelet) alapján. Egyes esetekben előfordulhat a fordított sorrendű (LIFO/FILO pl. LCFS / Last Come First Served szervezés esetén) kiszolgálás is.

2. Ütemezési algoritmusok
   Az ütemezés többféle algoritmussal valósítható meg. Az egyes ütemezők számára más-más megoldás tekinthető optimálisnak. Az összes olyan követelménynek, amit egy "ideálisan működő" operációs rendszerrel szemben lehet támasztani, már csak a kérdésekben rejlő ellentmondások miatt sem lehet egyszerre megfelelni. A cél nem is ez, hanem a "megfelelő célra, megfelelő algoritmust" elv szem előtt tartás. A leggyakrabban megfogalmazott követelmények operációs rendszertől függetlenek, általánosságban vonatkoznak az ütemezésre, illetve annak közvetlen következményeire:

- Minden folyamat kapjon minél előbb CPU időt.
- A folyamatokat a prioritásuknak megfelelően legyenek kezelve, de részesítse előnyben a kihasználatlan erőforrásokat, illetve a fontos (olyan, amelynek a működésétől, más folyamatok függenek) erőforrásokat felhasználni kívánó folyamatokat.
- A folyamatok lehetőleg nem legyenek túl sokáig várakoztatva (éheztetés).
- Az operációs rendszer viselkedése kiszámítható legyen, a hatékonyság (a szempontjait lásd egy oldallal ezelőtt) időben és pénzben (energiaráfordítás) is kifejezhető legyen.
- Legyen maximális az átbocsájtó képessége. (Közelítse az elvi maximumot...)
- Növekvő terhelés hatására a teljesítőképesség fokozatosan csökkenjen, ne álljon elő egy túlterhelési pont esetén azonnali rendszerösszeomlás.

Az igények megismerése után nézzük, hogy milyen módon csoportosíthatjuk az ütemezési algoritmusokat:

- Egyszerű algoritmusok
    - Legrégebben várakozó (FCFS / First Fome First Serve)
      Klasszikus, nem preemtív algoritmus. A futásra kész folyamatok egymás után kerülnek be a várakozási sorba, és érkezési sorrendben kerülnek kiválasztásra. (Olyan, mint mikor sorba állunk a mozijegyért...) Jellemzője a nagy várakozási idő, hiszen egy hosszan futó folyamat a mögött álló folyamatokat sokáig feltartja (konvoj hatás).
    - Körbeforgó (RR / Round-Robin)
      [RoundRobin algoritmus.xls]
      Klasszikus preemtív algoritmus, az időosztásos rendszerek így működnek. Minden folyamat egy időszeletnyi futási időt kap. Az időszelet és a CPU löket hosszának viszonya sorsdöntő ebben az esetben. Amennyiben az éppen futó folyamat CPU lökete hosszabb mint az időszelet, akkor az időszelet letelte után a futási jog átszáll a következő folyamatra, a folyamatunk pedig futó állapotból visszakerül a várakozó sor végére. Amennyiben az éppen futó folyamat CPU lökete rövidebb mint az időszelet (ez jellemzően egy folyamat utolsó ütemezésekor fordul elő), akkor a CPU löket végén (új időszeletet indítva) a sorban következő folyamatra száll át a futási jog.

Az időszelet optimális meghatározása nem egyszerű feladat. A túl hosszú időszelet alkalmazása az FCFS működéséhez teszi az RR-t hasonlóvá, a túl rövid időszelet pedig a nagyszámú környezetváltás miatt lesz alacsony hatékonyságú. A tapasztalati állandó az, hogy a CPU löketek kb. 20%-a legyen csak hosszabb az időszeletnél.

- Prioritásos algoritmusok

  A megoldás alapját az képezi, hogy az összes futásra kész folyamat kap egy maghatározott tartományon belüli prioritási értéket. A CPU-t mindig a sorban állók közül a legmagasabb prioritással rendelkező folyamat kapja meg. A prioritási értékek meghatározása történhet az operációs rendszer által felállított saját rangsor szerint (belső kiosztás) illetve az érték érkezhet az felhasználótól (külső kiosztás) is.

  A prioritás első hallásra tökéletes megoldásnak tűnik, de jobban átgondolva felmerül egy megoldandó probléma. Amennyiben egy folyamat nagyon alacsony prioritással rendelkezik, elképzelhető, hogy véges időn belül nem jut CPU-hoz (kiéheztetés), hiszen valószínű, hogy mindig lesz nála magasabb prioritású folyamat. Azért hogy ez a forgatókönyv ne fordulhasson elő a statikus (azaz időben állandó) prioritások helyett a dinamikus prioritások használata látszik célszerűnek. A dinamizmus lényege az, hogy a régóta várakozó folyamatok prioritását az operációs rendszer ciklikusan folyamatosan megnöveli, azaz öregíti (Aging).

  A prioritást az objektív fontosság mellett a CPU löketidő befolyásolja, amit viszont általában nem ismert, jellemzően becsléssel tudunk közelítő értékhez jutni. A becslésnél pontosabb információ lehet a folyamatot elindító felhasználó "önbevallása", illetve a folyamat esetleges korábbi viselkedéseiről készült feljegyzések.

Az alapvető prioritásos ütemezési algoritmusok:

- Legrövidebb löketidejű (SJF / Shortest Job First)
  Az algoritmus nem preemptív, a futásra kész folyamatok várakozási sorából a legrövidebb löketidejűnek adja a CPU-t. Közel optimális az átlagos várakozási és a futási idő, és az FCFS esetében fellépő konvoj hatás nem alakul ki.
- Legrövidebb hátralévő idejű (SRTF / Shortest Remaining Time First)
  Gyakorlatilag ez az algoritmus az SJF preemptív változata. Működési mechanizmusa az, hogy amennyiben a várakozó sorban következő futásra kész folyamat löketideje rövidebb, mint az éppen futó folyamat hátralévő ideje, akkor folyamatot cserél. Ez természetesen környezetváltással jár. Az ebből fakadó időveszteség ellenére is hatékony módszer.
- Legjobb válaszarány (HRR / Highest Response Ratio)
  Ez az algoritmus az SJF olyan változata, ahol az öregítés segítségével elkerülhető a kiéheztetés, és az indítandó folyamat kiválasztásnál a löketidő mellett a várakozási időt is figyelembe van véve. A prioritást a következő képlettel számítja ki a rendszer:
  (löketidő + k * várakozási idő) / löketidő,
  ahol a ”k” egy tapasztalati konstans.

- Többszintű algoritmusok
  A többszintű algoritmusok legfőbb jellemzője az, hogy a futásra kész folyamatok nem egy, hanem több önálló, egyedi prioritással rendelkező sorban várakoznak. Az egyes sorokon belül eltérő algoritmusok végzik a kiválasztást. Az ütemező egy kisebb prioritású sorból csak akkor választhat ki egy folyamatot, ha a magasabb prioritású sor(ok) már üresek.
  - Statikus többszintű sorok (SMQ / Static Multilevel Queues)
    Mivel statikus prioritásokkal dolgozik, nem mentes a kiéheztetéstől. Egy folyamat egyik várakozási sorból nem kerülhet át egy másik várakozási sorba. A várakozási sorok is saját állandó prioritással rendelkeznek.

A rendszer logikáját jól szemlélteti a következő példa:

1. rendszerfolyamatok (alapvetően fontos folyamatok)
2. interaktív folyamatok "a" (fontos a kellően gyors válaszidő)
3. interaktív folyamatok "b" (kevésbé kritikus válaszidő)
4. kötegelt feldolgozás (nem kritikus alkalmazások, fut ha van rá idő)
5. rendszerstatisztikák (nem fontos, hiszen a rendszer működésére nincs közvetlen hatással)

- Visszacsatolt többszintű sorok (MFQ / Multilevel Feedback Queues)

  A várakozási sorok az SMQ-hoz hasonlóan épülhetnek fel, de ez esetben a folyamatok képesek egyik várakozási sorból egy másik várakozási sorba átlépni. Ez a dinamizmus segít a kiéheztetés elkerülésében.

  Kétféle irány létezik:

  1. Egy folyamat alacsonyabb prioritású sorba kerül.

     A folyamatok végrehajtása mindig a legmagasabb prioritású sorból indul. Az egyes sorokhoz tartozó időszelet hossza viszont az alacsonyabb prioritású sorok esetében nagyobb. Amennyiben egy folyamat számára nem elegendő a rendelkezésre álló időszelet, akkor átkerül egy alacsonyabb prioritású sorba. Fontos szempont továbbá, hogy az ütemezés a rövidebb CPU löketű folyamatokat előnyben részesíti.

  2. Egy folyamat magasabb prioritású sorba kerül.

     Az előző modell merev szabályait tovább finomítva időről-időre folyamatosan mérve a gyors(nak vélelmezett) folyamatokat a rendszer magasabb prioritású sorba helyezi át. Emellett a régóta várakozó folyamatok szintén egy magasabb prioritású sorba kerülnek át.