Előadás_#02.

1. Folyamatok

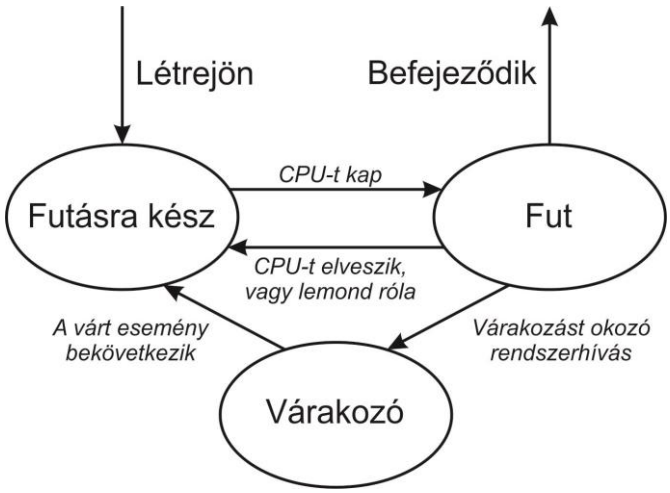
   A multiprogramozott rendszerek előtt a tiszta szekvenciális működés volt a jellemző. Egy program (itt szándékosan nem folyamatnak nevezzük) futtatása elkezdődött, teljes egészében lefutott (eredményt képezett), majd megállt. Ezután indulhatott a következő program futása. Ütemezésre nem volt szükség, pusztán a sorrend ismerete elegendő volt.

   A több folyamat (és ebből következően nyilván a több szál – amiről hamarosan szó lesz) párhuzamos megjelenését először az időosztásos (vagy időszeletes) rendszerek (Time Sharing Systems) tették lehetővé, ahol egyszerre több felhasználó is bejelentkezhetett a rendszerbe. Alapvető kérdés volt a processzor (ami nyilván egymagos volt) idejének igazságos, de hatékony megosztása (azaz időszeletenkénti kiosztása) a felhasználók közt. Az itt szerzett tapasztalatok vezettek a klasszikus multiprogramozáshoz, amikor már egy felhasználó is a több, párhuzamos(nak érzékelt) folyamat futtatásának a lehetőségét kapta meg.

   A folyamat (Process) a multiprogramozott operációs rendszerek alapfogalma. A folyamat tehát a számítási feladatok végrehajtásának alapegységét jelenti. Folyamaton így jellemzően a műveletek meghatározott sorrendben történő végrehajtását értjük. Az operációs rendszer működése folyamatok sorozata. Egy folyamat elkezdődik és a lehetséges állapotokat felvéve előbb utóbb be is fejeződik. A vonatkozó részművelet végrehajtása viszont csak akkor kezdődhet el, ha az előző részművelet végrehajtása már befejeződött.

   A folyamat állapotainak egyszerűsített leírása:

   - Futásra kész
     A folyamat már minden szükséges erőforrást birtokol, kivéve a CPU-t. (pont a CPU-ra vár)
   - Futó
     A folyamat fut, minden szükséges erőforrást birtokol, és ezen felül a CPU-t is birtokolja.
   - Várakozó
     A folyamat valamilyen feltétel teljesülésére vár.
     (pl.: I/O művelet eredményére)



Multiprogramozott rendszerekben a processzor magok számának megfelelő számú folyamat képes tisztán egyidőben futni. (Nem szabad megfeledkezni a Hyper-threading technológiáról sem, ami nem a folyamatokkal, hanem a szálakkal kapcsolatos – így arról kicsit később lesz szó.) A nem futó, hanem a várakozó, illetve a futásra kész folyamatok számának pedig csak gyakorlati korlátja van, elvi korlátja nincs.
A folyamatkezelés (Process Management) az operációs rendszerek mindazon szolgáltatásainak, tevékenységeinek körét magába foglalja, amelyek a folyamatok végrehajtásával és az ehhez fűződő adminisztrálással kapcsolatosak.

Egy folyamat viszont közel sem biztos, hogy elkezdődik, és egy menetben simán le is fut, hiszen a lényeg pont az, hogy az operációs rendszer lehetőséget ad arra, hogy egyszerre több végrehajtás alatt álló folyamat között (időszeletenként vagy más priorizálással) a CPU ütemező képes legyen váltani. [A CPU ütemező a rövid távú ütemező.] A módszer a váltások ellenére azt igyekszik (jellemzően sikerrel) "elhitetni" minden folyamattal, hogy ő az operációs rendszer – és ebből kifolyólag a hardver – egyedüli és osztatlan felhasználója.
Ne felejtsük tehát el, hogy normál működés közben, amikor egy folyamat megkapta és használja a CPU-t (CPU magot), biztosan el fog következni egy olyan időpillanat is, amikor nem egy másik folyamat igénye jelentkezik a CPU-ra, hanem maga az operációs rendszer igényli a CPU-t a saját belső mechanizmusainak a működtetéséhez. Ez pedig nem is ritkán jelentkező esemény, hanem az órajel alapján ciklikusan jelentkező esemény. Egy folyamat futása tehát megszakadhat egy másik (pl. magasabb prioritású) folyamat CPU igénye miatt, vagy az operációs rendszer CPU igénye miatt.
A megszakítás (Interrupt) az a műveletsorozat, amikor az éppen feldolgozás alatt lévő folyamat futásának félbehagyásával egy magasabb prioritású igény végrehajtásának indítása történik, oly módon elmenve az éppen futó folyamat környezetét (ez a fogalom hamarosan részletesen tárgyalásra kerül) egy átmeneti tárolóba, hogy az a későbbiekben folytatható legyen. A megszakítást, mint eszközt az teszi szükségessé, hogy a folyamatok végrehajtása során felléphetnek olyan események, melyek egyszerű, de hatékony kezelése csak az folyamat végrehajtás normális menetének átmeneti felfüggesztésével lehetséges.

[Magát a "megszakítás" elnevezést és terminológiát először az IBM használta, és terjesztette el. Természetesen gyorsan lettek követői, de például a Motorola a "kivétel" (exception) fogalmat használja. További érdekesség, hogy a Microsoft és a VAX egyes rendszerei mindkét fogalmat használják ráadásul nem is rokon értelmű kifejezésként, hanem mindkettőnek önálló értelmet adva.]

Kicsit közérthetőbben fogalmazva, az operációs rendszernek rugalmasan reagálnia kell a "külvilág" rendszert érintő eseményeire is. Ezt a kitűzött célt a megszakítás segítségével az operációs rendszer úgy éri el, hogy az éppen futó folyamatról a vezérlés ideiglenesen átadódik egy olyan másik folyamatra, amely képes kiszolgálni (kezelni) a bekövetkezett eseményt. A megszakítást kiszolgáló folyamat lefutása után pedig a megszakított folyamat végrehajtása a soron következő lépésétől kezdve folytatódik. Egy számítógép esetén például érzékelni kell a klaviatúra használatát, vagyis azt, hogy a felhasználó valamit begépel, vagy például azt, hogy egy periféria befejezte a kért műveletet és készen áll az eredmény vagy a kért adat átadására. Ez esetben például a néhány mikroszekundumos késleltetés/késlekedés nem jelent problémát, de a valós idejű (Real Time) és a folyamatirányítási (Process Control) rendszerek esetében az azonnali szuper gyors válaszadási idő már elsőrendű feladat.

[Real Time: Semmiképpen sem szabad azt gondolni, hogy itt valami olyan csodáról van szó, amely elvi és gyakorlati késedelem és időveszteség nélkül hajtja végre az elvárt műveleteket. Egy rendszert akkor tekinthetünk Real Time rendszernek, ha ELVÁRT sebességgel reagál. Például egy csiga Real Time irányításához vagy vezérléséhez a másodperc nagyságrendű beavatkozási illetve válaszidő is elegendő, mert közismert, hogy a legfürgébb csiga sem kapkodja el a dolgokat... A Real Time összegezve tehát azt jelenti, hogy a valós időben érkező igényeinkre valós időben kapunk választ, azaz egyszóval (még éppen) időben megérkezik a válasz illetve az eredmény ahhoz, hogy a szóban forgó folyamat zökkenőmentesen futhasson.]

A legjellemzőbb ilyen események csoportosítása:

- meghatározott külső műveletek befejezése, melyek bekövetkezésére számítani lehet, de ezek időpontja pontosan nem tervezhető
  (például egy periféria jelzi, hogy egy I/O művelet befejeződött)
- szándékos, azaz programvezérelt módon generált események
  (rendszerhívások)
- meghatározott programhibák
  (például 0-val való osztás)
- teljesen véletlenszerűen és váratlanul fellépő események
  (például hardverhiba vagy áramkimaradás)

A megszakítások két nagy csoportra bonthatók:

- Maszkolható megszakítás (Interrupt Request / IRQ [vagy IT])
  Ezek végrehajtásának bekövetkezése az IRQ prioritási szintjétől függ.
- Nem maszkolható megszakítás (Non Maskable Interrupt / NMI)
  Ezek jellemzően azonnal végrehajtandók (például órajel alapú megszakítás).

A folyamatok is két nagy csoportra bonthatók:

- Oszthatatlan műveleteket is tartalmazó
  Ezt még az NMI sem szakíthatja meg, ha éppen oszthatatlan műveletnél tart!
- Oszthatatlan műveleteket nem tartalmazó
  Tetszőlegesen megszakítható.

A végrehajtás alatt álló folyamatok közötti váltáshoz a váltandó folyamat környezetét (Context) el kell menteni, illetve az indítandó (vagy folytatandó) folyamat környezetét pedig be kell tölteni. Ez a tevékenység a környezetváltás. A környezetnek, mint információhalmaznak tartalmaznia kell a folyamat illetve a számítógép aktuális állapotát, beleértve mindazt az információt, ami a folyamat folytatásához, azaz újraindításához szükséges.

A folyamat és a számítógép aktuális állapotát egy összetett adatstruktúra a PCB (Process Controll Block / Folyamatvezérlési Blokk) tartalmazza.
A PCB struktúrájának 3 fő része, és azok legfontosabb összetevői:

- A folyamatot azonosító adatok
    - Memória információ, mely tartalmazza a folyamat által használt memóriaterületek adatait, illetve a címtranszformációs táblázatot.
    - A folyamat azonosítója mellett a szülőjének és gyerekeinek azonosítói.
    - I/O státus információ, mely a folyamathoz rendelt, illetve lefoglalt I/O-eszközök listáját, és a megnyitott fájlok listáját tartalmazza.
- A CPU állapotának adatai
    - CPU-regiszterek, melyek futás közbeni processzor szintű adatokat tartalmaznak.
    - CPU-ütemezési információ, mely az adott folyamat ütemezésével, prioritásával, jogosultságaival kapcsolatos adatok összessége.
    - Elszámolási információ, mely a felhasznált CPU-idő és valóságos idő, idő-korlátok, egyéb számlálók, stb. adatainak összessége.
- A folyamat vezérlési adatai
    - A folyamat állapota. (A folyamatok bővített állapotátmeneti diagramja szerinti állapotok közül az aktuális állapot.)
    - Programszámláló, mely a folyamat soron következő végrehajtandó utasításának a címét tartalmazza.
    - Mutatók (Pointer), melyek a tárolásra használt adatszerkezet mutatói. Segítségükkel a tárolt adatok gyorsabban érhetők el.

Az I/O műveletek paraméterei az (Input Output Control Block / be- és kimenetek vezérlő blokkja) IOCB-ben tárolódnak. Fontos része a rendszer struktúrájának, hogy az IOCB-k az I/O műveletet kezdeményező folyamat PCB-jére fűződnek fel. Ezzel azt is elértük, hogy az elindított I/O műveletek és az azokra várakozó folyamatok összekapcsolása is megoldottá vált. (lásd: A folyamatot azonosító adatok harmadik hivatkozása.)

2. Folyamatok modellezése

   Egy folyamat vagy a CPU egy magját használja (CPU Burst / CPU löket), vagy I/O műveletet hajt végre (I/O Burst / I/O löket). Ezzel el lehetett érni, hogy a CPU és az I/O (azaz a perifériák) valóban párhuzamosan működjenek. Legelső lépésként minden folyamat CPU lökettel kell hogy induljon. Ezen löketek kiosztásáról az operációs rendszer gondoskodik, aminek természetesen a saját működéséhez is szüksége van a CPU-ra. Szintén az operációs rendszer feladata a periféria vezérlők felprogramozása. Érdekes és hasznos megoldás a DMA (Direct Memory Access), a közvetlen memória elérés, aminek segítségével az arra alkalmas és arra képes perifériák közvetlenül használhatják a memóriát. Nyilván a DMA működéshez is elengedhetetlen a CPU, de ebben a megoldásban csak addig van szükség a CPU-ra, amíg a perifériavezérlő felprogramozása tart.

   Összefoglalva, tehát a rövid távú ütemező elkezdi ütemezni a folyamatok végrehajtását a CPU-n, szépen ütemezetten (CPU löketenként). Ebbe a "levesbe köp bele" ciklikus rendszerességgel az operációs rendszer CPU igénye egy NMI segítségével, illetve szinte sztohasztikusan az I/O műveletek egy IRQ, és a nem várt műveletek egy IRQ vagy egy NMI segítségével.

3. A folyamatok bővített állapotátmeneti diagramja