Előadás_#11.

1. A Windows NT alapú operációs rendszerek belső mechanizmusai
   Amikor a Windows NT alapú operációs rendszerek belső mechanizmusai kerülnek terítékre, nyilván a kernel módú komponensek (Executive illetve Device Driver) által megvalósított legfontosabb, legalapvetőbb funkciókra kell gondolni:
   - megszakítás- és kivételkezelés
   - objektumkezelés
   - szinkronizáció
   - lokális eljárás hívás (Local Procedure Calls / LPCs)

   Megszakítás- és kivételkezelés
   A megszakítások (Interrupt) és a kivételek (Exception) olyan műveletsorozatok, melyek hatására az éppen feldolgozás alatt lévő folyamat futásának félbehagyása történik, mivel egy magasabb prioritású igény végrehajtása válik indokolttá.
   A megszakítások és kivételek egyaránt lehetnek hardver vagy szoftver alapúak. Jellemző azonban, hogy a megszakítások esetében a hardver alapú megszakítások aránya a nagyobb, míg a kivételek esetében a szoftver alapú kivételek aránya a nagyobb. A kivételek kezelését egy erre a célra szolgáló kernelmodul, a kivételkezelő (Exception Dispatcher) végzi. A kivételkezelő feladata a bekövetkezett kivétel beazonosítása (pl. hibás memóriacímzés, nullával való osztás, túlcsordulás, a nyomkövető programok töréspontjai), a szükséges adminisztráció elvégzése, a probléma megoldása, majd a felfüggesztett folyamat futásának folytatása. Az előbb felsorolt példák szoftveres események, melyek nyilván függetlenek az architektúra kialakításától.
   A Windows NT alapú operációs rendszerekben létezik az úgynevezett csapda (Trap) fogalma is. Ez esetben azonban nem egy folyamat, hanem egy szál futásának a félbehagyásáról van szó. A mechanizmus végrehajtásához a CPU-t user módból kernel módba át kell váltani, majd meg kell hívni a trapkezelőt (Trap Dispatcher), amely azt hivatott eldönteni, hogy szoftveres úton kezelhető-e az adott probléma. Azért, hogy a szál végrehajtása az adott probléma lekezelése után folytatható legyen, a trapkezelő elvégzi a megfelelő adminisztrációt, eltárolja az operációs rendszer számára a folytatáshoz szükséges adatokat.

Objektumkezelés

A Windows NT alapú operációs rendszerekben az objektumok kezelését az executive egyik komponense az objektumkezelő (Object Manager) végzi. Feladata az objektumok előállítása, kezelése, törlése, és védelme. Az előbb felsorolt feladatokból következik, hogy az objektumkezelő tartalmazza az erőforrás kezelő műveleteket, melyek így jól definiáltan, egy helyen találhatók meg az operációs rendszeren belül.

Az objektumkezelő felé támasztott igények:

- Biztosítson egységes mechanizmust a rendszer erőforrásainak elérésére, illetve használatára.
- Az objektumok kezelését, és védelmét biztosító kód egy helyen legyen.
- A rendszer erőforrásainak használatát optimálisan korlátozva, folyamatosan kövesse nyomon a folyamatok objektumhasználatát.
- Megfelelő névtérrel biztosítsa az objektumok és objektum csoportok egyértelmű azonosítását.
- A Windows NT alrendszerek által támasztott követelmények kielégítése.
    - A Win32 (és anno a POSIX alrendszer) esetében szempont az, hogy egy folyamat legyen képes erőforrásokat örökölni a szülő folyamattól.
    - Legyen lehetőség kis- és nagybetűt megkülönböztető fájlnevek használatára.
- Biztosítani kell az objektumok életben tartását, ami azt jelenti, hogy egy objektumnak mindaddig rendelkezésre kell hogy állnia, amíg az összes vonatkozó folyamat be nem fejezte az objektum használatát.

A Windows NT alapú operációs rendszerekben a következő két objektumtípust különböztetünk meg:

- executive objektum típus
- kernel objektum típus

Az executive objektumok az összetettebb szerkezetűek, ezeket az executive réteg különböző komponensei hozzák létre. (a folyamatkezelő, a biztonsági alrendszer, a virtuálismemória-kezelő, az I/O-alrendszer)

A kernel objektumok egyszerűbb szerkezetűek, mégis alapvető funkciókat valósítanak meg. Ilyen például a szinkronizáció, amelyet az executive objektumok is igénybe vesznek. Az executive objektumok jellemzően több kernel objektumból épülnek fel.

A folyamatok az objektumokhoz egy úgynevezett handle (szószerinti jelentése fogantyú, nyél, – „igazi” magyar elnevezése nincs, de „leíró”-ként hivatkozunk rá)

igénybe vételével férhetnek hozzá. A handle tartalma tehát az objektumhoz való hozzáférési információk összessége. Tartalmilag egy összetett elérési címről van szó, ami gyakorlatilag egy összetett pointer, aminek igénybe vételével lényegesen gyorsabban érhető el egy objektum annál, mintha az objektum nevére keresnénk, vagy hivatkoznánk.

Szinkronizáció

A folyamatok szinkronizálásának alapvető eszköze a kölcsönös kizárás, különös tekintettel a nem osztott elérésű erőforrások elérésének szabályozására. A kölcsönös kizárás különösen fontos a Windows NT alapú operációs rendszerek esetében, mivel a több egymagos CPU, illetve a több magos CPU architektúrák használatakor is ez elengedhetetlen. A kernel, hogy elláthassa a több szál által is használt adatstruktúrák kezelését, a kölcsönös kizárást megvalósító alapmodulokat (a korábban már említett primitíveket) veszi igénybe. Ebben az esetben is elengedhetetlen a szinkronizáció, melyet az executive réteg biztosít, szintén a primitívek használatával. A kernel kódok végrehajtásakor – a végrehajtás minden fázisában – a kernelnek biztosítania kell, hogy egy kritikus szakaszt csakis egyetlen egymagos CPU, vagy többmagos CPU esetén pedig egyetlen CPU mag hajt végre.

Mivel megszakítások gyakorlatilag bármikor bekövetkezhetnek, kezelésük a szinkronizáció során nem mindig oldható meg zökkenőmentesen. Amennyiben a kernel éppen egy globális adatstruktúrát módosít, előfordulhat, hogy pont egy olyan megszakítás történik, aminek a kezelő rutinja pontosan ugyanazt a globális struktúrát módosítaná. Az ilyen helyzetekben a Windows NT alapú operációs rendszerek más megoldást alkalmaznak egy CPU mag esetén, illetve több CPU vagy több CPU mag esetén. Egy CPU mag esetén a Windows NT alapú operációs rendszerek mozgástere kicsi. Az operációs rendszer mindössze annyit tehet, hogy a globális erőforrások használata előtt ideiglenesen letiltja azokat a megszakításokat, amelyek megszakítás kezelő programja ugyanazt a globális erőforrást venné használatba. Technikailag ez úgy történik, hogy a CPU futási szintjét a szóban forgó megszakítások közül a legmagasabb szintre emeli az operációs rendszer. Több CPU mag esetében ez nyilván nem elegendő, hiszen hiába emeli meg az operációs rendszer az egyik CPU mag prioritását, egy másik CPU mag ettől még készen állhat a megszakítás elfogadására. A kölcsönös kizárás ilyen esetben az úgynevezett lezárással (Locking) garantálható, ami úgy valósítható meg, hogy a szóban forgó globális erőforráshoz tartozó Spin-Lock (szószerinti jelentése forgózár, – „igazi” magyar elnevezése nem használatos) segítségével a kernel zárolja az erőforrást.

A Spin-Lock megszerzése után az operációs rendszer már priorizálhatja a globális erőforrás használatára jelentkező igények végrehajtási sorrendjét.
Multiprocesszoros környezetben a globális adatstruktúrákhoz való hozzáférést az executive réteg komponenseinek is szinkronizálnia kell. A kernel funkciók hívásakor is célszerű a lezárás használata, annak ellenére, hogy járulékos időveszteség nélkül elvileg sem vitelezhető ki maga a lezárás, hiszen a prioritási sorban elfoglalt helyének megfelelően minden folyamat több-kevesebb ideig mindenképpen várakozni kényszerül, sőt az összes érintett CPU mag is várakozni kényszerül. Ezért a módszer nem alkalmazható korlátozásmentesen:

- A lezárással védett erőforrásnak gyorsan elérhetőnek kell lennie, azaz nem lehetnek szerteágazó logikai kapcsolatai.
- A kritikus szakaszok nem lapozhatók ki a memóriából, nem hívhatnak külső eljárásokat (rendszerszolgáltatásokat sem), valamint nem okozhatnak se megszakítást, se kivételt.

Ezek a korlátozások – különösen annak figyelembe vételével, hogy az executive objektumok a kölcsönös kizárás mellett igényt tarthatnak az előidejűségre, illetve az egyidejűségre is, valamint a user folyamatokkal is megkövetelt szinkronizáció miatt – nem minden esetben érvényesíthetők pusztán a Spin-Lock igénybevételével. Szükség van további szinkronizációs objektum kezelőkre (Dispatcher Object) is, melyek megfelelő függvényhívások segítségével képesek dinamikusan kezelni – azaz főleg várakoztatni – a hozzáférést igénylő objektumokat (WaitForSingleObject, WaitForMultiplyObject). Az operációs rendszer a várakoztatott objektum felszabadulásakor, annak típusától függően, egy vagy több várakozó folyamatot is futásra kész állapotba tehet.

Lokális eljáráshívás (LPC)

A lokális eljáráshívás (LPC / Local Procedure Call) a folyamatok közötti kommunikációt biztosítja gyors üzenet továbbítással. Windows NT alapú operációs rendszerek esetében az LPC kizárólag a rendszer komponensek számára használható, így például a Win32 API-n keresztül nem is érhető el. Az LPC tipikus esete a Windows NT alapú operációs rendszerek szolgáltatásait realizáló szerver folyamatok és a kliens folyamatok közötti kapcsolattartás. LPC kapcsolat létrehozható két user módú folyamat között, vagy egy kernel módú komponens és egy user módú folyamat között. Maga az üzenetváltás realizációja az üzenet méretétől függ. Rövid, azaz 256 bájt üzenetméret alatt egy egyszerű puffer elegendő, nagyobb adatmennyiség esetén azonban közös elérésű, osztott memóriaterületre van szükség.

2. A Windows NT alapú operációs rendszerek folyamatai és szálai

Alapfogalmak:

- A program maga a végrehajtható kód.
- A folyamat egy végrehajtás alatt lévő program(rész).
- A folyamat egy szála az, ami éppen fut egy CPU magon, és nem maga a folyamat.
- Minden folyamathoz tartozik legalább egy szál, ami elinduláskor elkezdi futni.

A Windows NT alapú operációs rendszerekben a programok futtatásához a folyamatok egy vagy több – a program aktuális kódját végrehajtó – szálból állnak. A szálak az erőforrásaikat nem közvetlenül az operációs rendszertől kapják, hanem azokat az őket létrehozó folyamatoktól öröklik.

A Windows NT alapú operációs rendszerek folyamat modelljének tartalma:

- A végrehajtandó program kódja és adatai.
- A saját virtuális memória címtere, ami a folyamat rendelkezésére áll.
- Rendszererőforrások (szemaforok, fájlok, stb.) amelyeket az operációs rendszer foglalt le a folyamat részére. Ezek megfelelő része öröklődik igény esetén a folyamathoz tartozó szálakhoz.
- A folyamat egyedi azonosítója (Process ID / PID).
- Legalább egy végrehajtható szál.

A szálak a következő komponensekből állnak:

- A szálat végrehajtó CPU mag regiszterei, melyek a CPU mag állapotát írják le.
- Két veremtárat (Stack), egyiket a kernel módú-, a másikat a user módú futáshoz.
- Egyedi, osztatlan tárterületet a DLL-ek, Run-time könyvtárak részére.
- A szál egyedi azonosítóját (Thread ID). Mivel a Thread ID és a Process ID ugyanabból a névtérből kerül kiosztásra, így kizárt az azonos név használata.

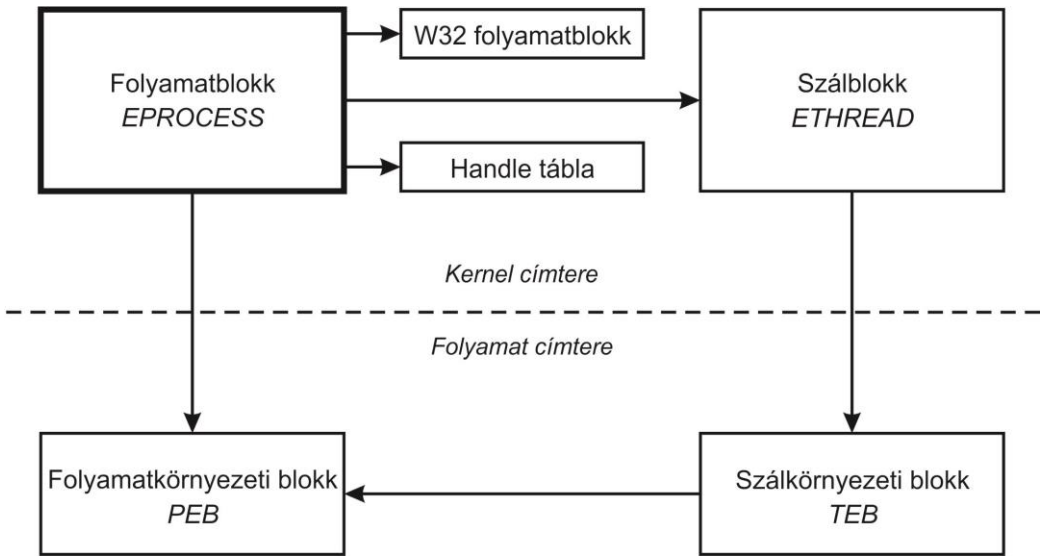
A fenti lista első három eleme együttesen alkotja a szál környezetét (Thread Context).

Az egy folyamathoz tartozó szálak közös virtuális címtartományt használnak. A különböző folyamatok természetesen külön, önálló címtartományban futnak, így például csak az osztott elérésű memória használata esetén lehet átfedés a két vagy több folyamat által használt memóriaterületek között.

A folyamatok – hosszabb távon a folyamatokat alkotó szálak – által használni kívánt memóriaterületeket természetesen az operációs rendszertől kell igényelni, és le is kell foglalni azokat. A folyamatnak továbbá a használni kívánt erőforrásokat is le kell foglalnia, amelyeket a Windows NT alapú operációs rendszer objektumokként reprezentál. A folyamat a későbbi gyorsabb elérés érdekében egy egyedi handle-t is rendel közvetlenül a megnyitás után minden ilyen objektumhoz. A rendszer erőforrásainak védelme miatt, valamint az erőforrások használatának szabályozása érdekében minden folyamathoz tartozik egy elérési token. Az elérési token az adott folyamat jogosultságainak leírása mellett a folyamat biztonsági azonosítóját is tartalmazza.

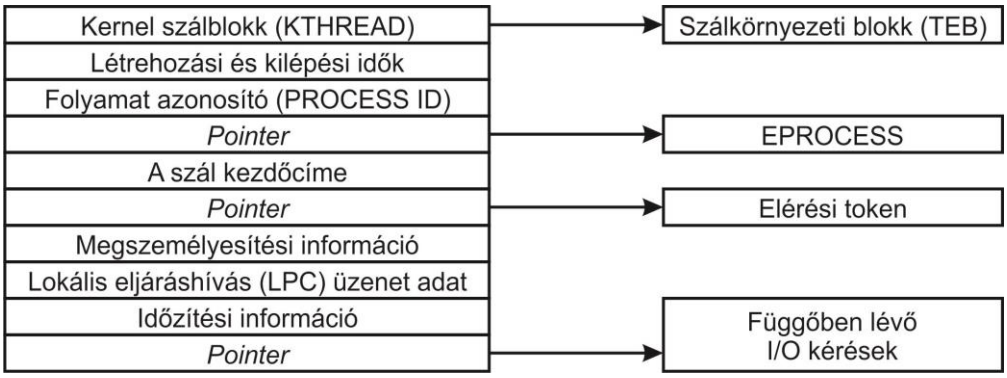
A folyamatok és a szálak adatstruktúrája

A folyamatokhoz tartozó adatokat az executive réteg egy folyamat blokkban (EPROCESS) tárolja el. Az EPROCESS és a hozzá tartozó adatstruktúrák – egy blokk kivételével – mind a kernel címterében találhatóak. Ez a kivétel a folyamatkörnyezeti blokk (Process Environment Block / PEB) ami a folyamat címterében található. Ennek oka, hogy a PEB – ami a folyamat futási környezetét leírja – az az adatstruktúra, ami olyan információkat tartalmaz, melyeket a user módban futó kódok is meg kell hogy tudjanak változtatni. A Win32 alrendszerben a CSRSS.EXE folyamat hozzárendel a Win32 kódot végrehajtó folyamtokhoz ezen felül még egy további folyamatleíró adatstruktúrát (W32PROCESS) is.



A Windows NT alapú operációs rendszerek a folyamatokhoz már azonnal, az indulásukkor hozzárendelik EPROCESS-t és annak adatstruktúráit, így kapcsolva össze a folyamat kísérő adatait, és a kapcsolódó adatstruktúrákra vonatkozó mutatókat. A folyamathoz tartozó egy vagy több szálat az executive szálblokk (ETHREAD) – vagy más néven: az adminisztratív szálblokk – írja le.

Az executive szálblokk felépítése a következő ábrán látható.



Az összes olyan információt, amire a kernelnek szüksége van a szálütemezéshez és a szinkronizációhoz a kernel szálblokk, a KTHREAD tartalmazza. A többi mező a szál vonatkozó adatait, illetve a vonatkozó mutatóit (Pointer) tartalmazza.

Folyamat létrehozása

Egy Win32 folyamat úgy keletkezik, hogy egy alkalmazás meghívja a Win32 CreateProcess függvényt. A folyamat létrehozása komplex, összetett feladat, melyben az operációs rendszer következő három eleme vesz részt:

- a Win32 kliensoldali könyvtárában a KERNEL32.DLL
- a Windows NT executive
- a Win32 alrendszer folyamat (CSRSS)

Az előbb felsorolt három elem a következő hét lépésben hajtja végre a Win32 CreateProcess függvény hívását:

1. A vonatkozó paraméterek ellenőrzése után, az alrendszer jelzőit (Flag) és opcióit – valamint az attribútum listát is – elemzés és érvényesítés után át kell állítani a natív megfeleőikre.
2. Azon végrehajtandó fájl (.EXE) megnyitása, aminek az elindítandó folyamaton belül végre kell hajtódnia.
3. A Windows NT executive folyamat- és szálkezelője létrehozza az új folyamat objektumot.
4. Az indításhoz szükséges, úgynevezett kezdeti szál (Initial Thread) létrehozása.

5. Az alrendszer specifikus inicializáció, azaz Win32 alrendszer értesül az új folyamatról és a kezdeti szálról.
6. A kezdeti szál elindítása (kivéve azt az esetet, ha a CREATE_SUSPEND flag érvényes rá).
7. Az új folyamat és szál környezetben a címtér inicializálása (a vonatkozó DLL-ek betöltése) után elkezdődik a program végrehajtása.

Általános megjegyzések a fenti lépésekkel kapcsolatban:

- A CreateProcess függvényben az új folyamat prioritási osztálya független bitként van megadva a CreationFlags paraméterben. Így egyetlen CreateProcess híváshoz akár több prioritási osztály is specifikálható.
- Amennyiben nincs egyedi prioritási osztály hozzárendelve az új folyamathoz, a prioritási osztály a normál (Normal) besorolással indul, kivéve, ha a létrehozó folyamat prioritási osztálya tétlen (Idle) vagy a normál (Normal) osztály alatti. Ez utóbbi esetekben az új folyamat prioritása ugyanolyan osztályú lesz, mint amilyen a létrehozó folyamaté volt.
- Grafikus megjelenítés esetén minden ablakot egy adott grafikus környezethez (Desktop) rendel a rendszer. Egy user akár több desktopot is használhat. Ha a CreateProcess külön nem definiálja, hogy melyik desktopot használja a folyamat, akkor alapértelmezetten a hívó aktuális desktopjához rendelődik hozzá a folyamat.
- Amennyiben az új folyamathoz valós idejű prioritásosztály (Real Time) van megadva, de a folyamat hívója nem rendelkezik az ütemezési prioritás növelésének jogosultságával, akkor a magas prioritású (High) osztályt kell használni.

Szálak létrehozása és kezelése

A kezdeti szál és környezetének létrehozása a következő lépésekkel indul.

1. Ezen a ponton a Windows NT executive folyamat objektuma teljesen készen áll. Ennek ellenére még mindig nincs szála, így még nem tehet semmit. Az új szál létrehozásáért a PspCreateThread függvény felelős, amit az NtCreateThread hív meg. Mivel azonban a kezdeti szálat a kernel hozza létre – akár felhasználói módú bemenet nélkül is – a PspCreateThread két segítő függvényt kell hogy igénybe vegyen, a PspAllocateThread és a PspInsertThread függvényeket.

2. A PspAllocateThread kezeli az executive szál objektum tényleges létrehozását és inicializálását. A PspInsertThread végzi a handle és a biztonsági attribútumok létrehozását, valamint a KeStartThread hívását, ami az executive objektumot ütemezhető szállá alakítja. A szál azonban még nem tesz semmit, mivel felfüggesztett állapotban jön létre.

Az ezután következő lépéssort a PspAllocateThread a hajtja végre:

1. Létrejön és inicializálódik egy executive szálblokk (ETHREAD).
2. A szál végrehajtása előtt kell beállítani a szál környezetet (Thread Context).
3. A szálkörnyezeti blokk (TEB / Thread Environment Block) hozzárendelése az új szálhoz.
4. A user módú szál kezdési címe az ETHREAD-ben tárolódik. Ez a rendszer által biztosított szálindítási funkció az Ntdll.dll fájlban (RtlUserThreadStart). A felhasználó által megadott kezdőcím is az ETHREAD blokkban található, de egy másik helyen, így a hibakeresési eszközök, például a Process Explorer lekérdezhetik ezeket az információkat.
5. A KeInitThread meghívásra kerül a KTHREAD blokk beállításához. A szál kezdeti és jelenlegi alap prioritásai a folyamat alap prioritására vannak beállítva, processzor affinitása és kvantuma szintén a folyamaté. Ekkor lesz definiálva a kezdeti szálhoz az ideális processzor is. A KeInitThread ezután kernel veremtárat foglal le a szálhoz, és inicializálja a szál gépfüggő környezetet (Thread Context), a csapdát (Trap) és a kivételeket (Exception). A szál környezete úgy van beállítva, hogy a szál kernel módban indul a KiThreadStartup segítségével. Végül a KeInitThread inicializált állapotba állítja a szál állapotát, és visszatér a PspAllocateThread függvényhez.

Az előzőleg leírt lépések után, az NtCreateUserProcess meghívja a PspInsertThread függvényt a következő lépések végrehajtásához:

1. Létrejön egy szál azonosító (Thread ID) az új szálhoz.
2. A folyamathoz tartozó szálak száma eggyel növekszik, és a szál hozzáadódik a folyamat szálainak listájához.
3. A szál felfüggesztett állapotba kerül.
4. Az ObOpenObjectByName segítségével létrejön a vonatkozó handle.
5. A szál a KeStartThread meghívásával készen áll a végrehajtásra.

3. A Windows NT alapú operációs rendszerek ütemezése:
   Alapfogalmak ismétlése, összefoglalása:
   - Windows NT alapú operációs rendszerek estén az ütemezés szál alapú, prioritásvezérelt, és preemptív.
   - Folyamat (Process): Egy program futásának számára biztosított, fenntartott rendszererőforrások egysége, mely nagy vonalakban a következőket tartalmazza. A program kódja, és adatai, saját virtuális címterület, operációs rendszer által hozzárendelt rendszererőforrások (szemaforok, kommunikációs portok, fájlok, stb.), egyedi azonosító, legalább egy szál.
   - Szál (Thread): Egy folyamathoz tartozó entitás, amin a program futása történik. Egy szálhoz a következő dolgok tartoznak: A processzor állapotát jelző regiszterek tartalma, két veremtár (kernel illetve user módbeli futáshoz), egyedi azonosító, saját tárolóterület, ahol a szálhoz kapcsolódó információk tárolhatók. A két veremtárat, a saját tárterületet és a tárolt regisztereket nevezzük a szál környezetének (Context). Ez az egyetlen olyan publikusan elérhető adatstruktúra a Windows NT alapú operációs rendszerekben, ami “számítógépfüggő”.
   - A Windows NT alapú operációs rendszerekben két processzor hozzáférési mód létezik. Az egyik a user mód, ahol a felhasználói programok futnak. Az itt futó két alkalmazás nem tudja sem egymás, sem az operációs rendszer fontos adatait módosítani, elérni. A másik a kernel mód, ahol az operációs rendszer fut. Egy kernel módban futó kód eléri a teljes memóriát, és minden processzorutasítást használhat.
   - Megszakítási szintek (IRQL): A különböző típusú processzoroknak eltérő a megszakítási rendszere, ezért definiáltak a kernelben egységes, hordozható megszakítási szinteket. Egy bizonyos szintű megszakítás után már nem fogadható el az aktuálisnál alacsonyabb szintű megszakítás, az aktuálisnál magasabb szintű megszakítás viszont félbeszakíthatja az aktuális megszakítás feldolgozásának menetét. Az ütemezés során az első három szinttel találkozhatunk (IRQL 0-2), melyek “szoftver megszakítások” (azaz nem egy hardver eszköz váltja ki őket).
     IRQL 0: a CPU egy normál kernel vagy user módú folyamatot futtat
     IRQL 1: a CPU egy aszinkron programhívást (APC) vagy laphibát futtat
     IRQL 2: a CPU egy halasztott eljáráshívást (DPC) és szálütemezést végez

Mivel az ütemezés szál alapú, ezért ha egy folyamat több szálon fut, mint egy másik folyamat, és a szálak prioritása azonos, akkor az első folyamatnak több idő fog jutni (ha – első megközelítésben – úgy tekintjük, hogy minden szál azonos ideig fut).
Az ütemezés prioritásvezérelt, vagyis az, hogy melyik szál fog futni, alapvetően a szálak prioritásától fog függeni. Alapvetően mindig a legnagyobb prioritású, futásra kész szál fut, hacsak a szál processzor affinitása ezt nem korlátozza. A processzor affinitás egy szálhoz azokat a processzorokat (magokat) rendeli, melyeken a futása engedélyezett.
Az ütemezés preemptív, vagyis egy szál futását az ütemező bármikor megszakíthatja, ha például egy magasabb prioritású szál lesz futásra kész. A Windows NT alapú operációs rendszerek ütemezője a kernel kódjában "szétszórva" található, azaz nincs egy kitüntetett ütemező modul.

Az ütemezés jellemzői:

- Preemtivitás
- Időszelet alapú ütemezés
- Többprocesszoros ütemezés

Többprocesszoros ütemezés (illetve több CPU mag) esetén a cél a szálak (azaz a terhelés) egyenletes elosztása, a következő három paraméter segítségével:

- processzor affinitás
  A szál azon paramétere, hogy melyik processzoron, illetve magon fusson.
- ideális processzor
  Azaz az előnyben részesített processzor, illetve mag. Ha a szál külön nem kéri, akkor véletlenszerűen választódik ki a tétlen processzorok közül az, amelyiken a szál futni fog. Ha nincs tétlen processzor, akkor a prioritás dönt a processzor használatról.
- következő processzor
  Azaz a másodlagosan preferált processzor, illetve mag, jellemzően az, amin a szál legutóbb futott.

Az ütemezéssel kapcsolatos rutinok (összefoglaló néven: diszpécser) IRQL2 szinten futnak, és a következő események válthatják ki a működésüket:

- egy szál végrehajtásra késszé válik
- egy szál abbahagyja a futást
- megváltozik egy szálnak a prioritása
- megváltozik a végrehajtás alatt álló szál processzor affinitása

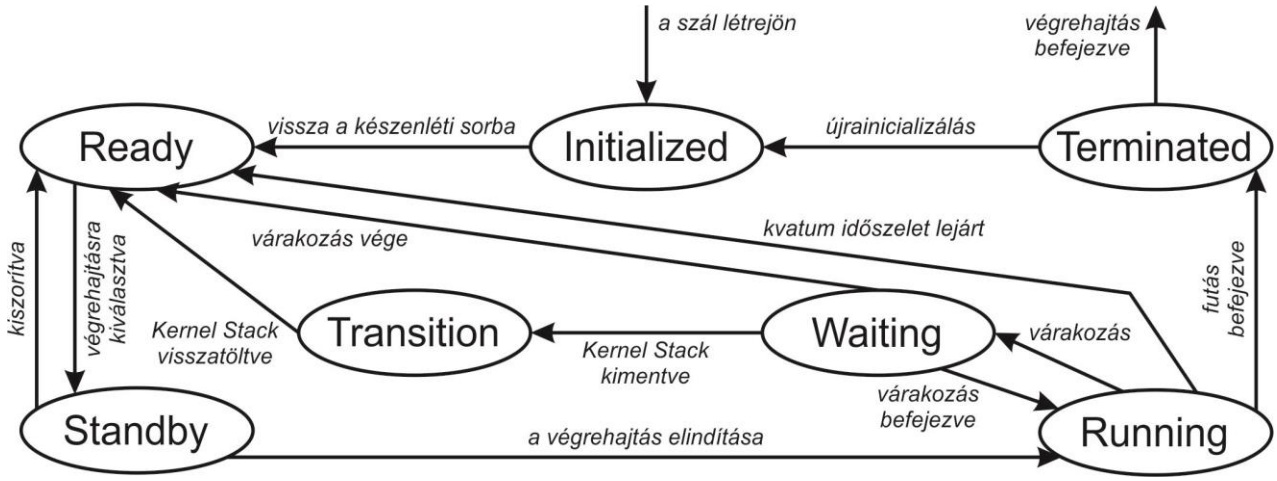
A kvantum időszelet

Minden szálhoz tartozik egy egyedi kvantum időszelet (Quantum), melynek lejárta után a szálnak mindenképpen abba kell hagynia a futást. Ezután újbóli kiválasztás és döntés születik, hogy melyik szál legyen a következő, amely futni fog. Egy kvantum időszelet tehát az a maximális időtartam, amennyit egy szál futhat, mielőtt a Windows NT alapú operációs rendszer biztosan megszakítja a szál futását. A futás megszakításának a célja az, hogy kiderülhessen, nem vár-e futásra egy másik, ugyanakkora prioritású szál, illetve, hogy szükség esetén csökkenteni lehessen a megszakított szál prioritását. Előfordulhat az is, hogy egy szál el sem jut a kvantum időszelete végéig, a preemptív ütemezés következtében. Ugyanis, ha egy másik, nagyobb prioritású szál futásra kész állapotba kerül, akkor a futásban levő szál futása még a saját időszelete lejárta előtt meg lesz szakítva, vagyis a szál ki lesz szorítva. Így az is előfordulhat, hogy egy szál hiába választódik ki futásra, még azelőtt ki is lesz szorítva, hogy elkezdené a kvantum időszeltét.

Az egyes szálak futására rendelkezésre álló kvantum időszeletek száma nyilván egy viszonylag kis érték kell hogy legyen, hiszen szálak komplexitása töredéke a folyamatok komplexitásának. Ez a kvantum érték.

Órajelmegszakításonként a Windows NT alapú operációs rendszerek a szálak kvantum értékét 3-al csökkentik. A kvantum érték a Windows NT Workstation esetében 6-ról, a Windows NT Server esetében pedig 36-ról indul. Ez azt jelenti, hogy Windows NT Workstation esetén 2, Windows NT Server esetén 12 alkalom alatt van lehetősége lefutni egy szálnak. A Windows NT Server esetében azért nagyobb ez az érték, mert statisztikailag ennyi idő alatt jellemzően befejezhető egy kliens kiszolgálása is. Egy előtérben futó alkalmazás szálainak kvantum értéke indokolt esetben a fent leírtakhoz képest akár nagyobb is lehet.

A szálak állapotai



Egy szálnak a fenti ábra szerint a következő állapotai lehetnek:

- Inicialzált (Initialized)
  Az induló lépés egy újonnan létrejött szál objektum inicializálása. Egy már befejezett szál objektum az újrainicializálása után is ebbe az állapotba kerül.
- Végrehajtásra kész (Ready)
  Azok a szálak, melyek futásra készek, itt arra várnak, hogy a diszpécser által kiválasztásra kerüljenek. A Terminated állapot kivételével bármelyik másik állapotból ide kerülhet a szál.
- Készenléti (Standby)
  Ez a végrehajtásra kiválasztott, processzoronként (CPU magonként) egyetlen szál helye. Minden processzorhoz (azaz CPU maghoz) csak egy olyan szál tartozhat, amely már ki van választva futásra, de még nem fut, mert pl. maga az ütemező is ezen a processzoron fut. Közvetlenül a Ready állapotból kerül ide egy szál, és innen vagy a Running állapot következik; vagy ha egy magasabb prioritású szál kiszorítja, akkor visszakerül a Ready állapotba.
- Futó (Running)
  A végrehajtás alatt álló száll állapota. A szál ide vagy a Standby állapotból, vagy egy várakozás befejeződése után kerülhet. Ha a szál futása befejeződik, akkor Terminated állapotba kerül. Ha a szál várakozni kényszerül, akkor Wait állapotba kerül, illetve ha a kvantum időszelete lejár, de kvantum értéke még nem nulla, akkor a Ready állapotba kerül vissza.
- Várakozó (Waiting)
  A szál akkor kerül Wait állapotba, ha valamilyen szinkronizációs objektumra, vagy I/O műveletre vár, illetve ha egy környezeti alrendszer utasítja erre a szálat. A szál a várakozási idejének letelte után, a prioritásának megfelelően, kerülhet újra Running állapotba, Ready állapotba vagy Transition állapotba.

- Átmeneti (Transition)
  Ebbe az állapotba csak akkor kerülhet egy szál, ha a szál kernel veremtára kilapozódott a memóriából a várakozás közben. A veremtár visszalapozása után a szál újra Ready állapotba kerül.
- Befejezett (Terminated)
  A szál befejezte a futását, végrehajtása véget ért. Ezután vagy törölhető a szál objektum, vagy újrainicializálható.

Prioritási szintek

Amikor prioritási szintekről beszélünk, két dolgot kell megkülönböztetnünk. Az egyik az a prioritás, mely a Windows NT alapú operációs rendszer alatt a programozó által elérhető, és a Windows API-n keresztül állítható be. A másik, mely a kernel belső ábrázolásában használatos, így ehhez csak a kernel folyamatai férnek közvetlenül hozzá. Az API-ból beállított érték is végül egy ilyen értékként képeződik le.

Egy folyamatnak adott prioritási osztállyal a folyamatunk fontosságát fejezhetjük ki. Ezen kívül a folyamathoz tartozó minden szálnak egyenként adhatunk egy prioritást, mely a folyamaton belül, a szálak közötti fontossági sorrendet jelenti. Ebben az anyagban korábban már szerepelt, hogy a Windows NT alapú operációs rendszerek ütemezése szál alapú, prioritásvezérelt, és preemptív, de fontosnak tartom erre itt újra felhívni a figyelmet, mivel mindez még hangsúlyosabbá teszi a prioritási szintek és osztályok szerepét a CPU-hoz jutás során.

Az API-ból beállított prioritásokból végül is minden szálra képződik egy 0 és 31 közötti érték, mely a folyamat és a szál megadott prioritásának kombinációjaként jön létre a következő leképezés alapján:

		Folyamat prioritási osztályok			
		Real-time	High	Normal	Idle
Szál prioritási szintek	Time critical	31	15	15	15
	Highest	26	15	10	6
	Above normal	25	14	9	5
	Normal	24	13	8	4
	Below normal	23	12	7	3
	Lowest	22	11	6	2
	Idle	16	1	1	1

A prioritási osztályok szintjei három csoportra oszthatók.

- 16-31 valós idejű szintek
- 1-15 változó szintek
- 0 zero szint (a táblázatban nem szerepel)

Az első, a real-time folyamat prioritási csoportban használatos értékek, vagyis a 16-tól 31-ig terjedő prioritásértékek. Ha egy szálnak ilyen a prioritása, akkor azt az operációs rendszer nem módosítja. A real-time, azaz a valós idő a gyakorlatban leginkább csak elvileg létezik, közelebb áll a valósághoz a just-in-time, azaz a megfelelő időben megközelítés, ami a lehető legkisebb késedelemre utal.
A másik csoport az 1-től 15-ig lévő tartomány, ahol is az operációs rendszer ezen a tartományon belül módosíthatja is a szál aktuális prioritásának értéket egy időre, a megadott érték csak egy alapérték. A módosítás viszont csak ebben a tartományban történhet, vagyis egy szál sohasem kerülhet fel ilyen módosítás során a 16-31 lévő csoportba. A módosítás lehet emelés (éheztetés vagy I/O folyamat vége után) illetve csökkentés (a szál futásának megszakítása után).
A harmadik csoport valójában nem is csoport, csak egy különálló érték, a 0 prioritás. Ez igazából egy szálhoz sem tartozhat, ezért nem is szerepel a fenti táblázatban. A Windows NT alapú operációs rendszerek minden processzorhoz (CPU maghoz) egy darab “látszólagos” szálat rendelnek hozzá, melyet a Task Manager-ben “System Idle Process” néven találhatunk meg.
Alapvetően minden szál normál prioritással kezdi a futását.