Előadás_#04.

1. Szálak (Thread)

   A szálakról általános érvényű megfogalmazásokat – általában a multiprogramozott rendszerekre vonatkozóan – csak nagyon óvatosan lehet tenni, hiszen a szálkezelés tekintetében egy Windows NT alapú operációs rendszer és a UNIX abszolút másképp viselkedik, például az ütemezés szempontjából. Ugyanis a Windows NT alapú rendszerek szálütemezést használnak, míg a UNIX kizárólag folyamat ütemezést használ.

   A folyamatok a végrehajtás során több (akár párhuzamosan is végrehajtódó) szálra felosztva hajtódhatnak végre. Fontos megjegyezni, hogy az erőforrásokat alapvetően a folyamatok kapják, de a folyamatoktól a szálak öröklik. Arról sem szabad megfeledkezni, hogy a folyamatok által használt erőforrások, mint olyan természetesen jelenthetnek hardveres és szoftveres erőforrásokat is.

   A rendszer működését átfogó logika tehát az, hogy a felhasználó által futtatott program lesz maga a folyamat, és a folyamat miután megkapta az erőforrásokat, ha lehetősége van rá szálakra bomlik.

   Például egy "IF" függvény egyes ágai mint szálak párhuzamosan feldolgozhatók, hiszen minden, a számításhoz szükséges adat közös. Abból a szempontból, hogy a folyamatoktól örökölt erőforrásokon a szálak osztoznak-e megkülönböztetünk "nehéz súlyú" folyamatokat illetve, "könnyű súlyú folyamatokat". A könnyű súlyú folyamatok szálai osztoznak az erőforrásokon. Tehát az előbb említett "IF" egy könnyű súlyú folyamat része volt, és ami képes volt szálakra bomlani. A nehéz súlyú folyamatok esetében nem beszélhetünk többes számban szálakról, csak egy végrehajtási szál van (ami így végeredményben maga a folyamat) azaz nincs ami osztozhatna az erőforrásokon.

   A szálak végrehajtásának lényege tehát az, hogy a folyamat által meghatározott közös memóriát (memória címtartományt) használják, azaz a szálak váltásához nincs szükség környezetváltásra, jellemzően egy veremtár használata elegendő. Ebből az következik, hogy a szálak váltása sokkal gyorsabb, mint a folyamatok váltása. Ezért is mondhatjuk, hogy a szálak – megfelelő CPU esetében – párhuzamos végrehajtásúak.

   Felsorolás szintjén hasonlítsuk össze, hogy milyen elemeket kell egy folyamatnak, és milyen elemeket kell egy szálnak tartalmaznia, nem elfejtve azt, hogy a folyamat elemeihez a szálak hozzáférnek (hiszen öröklik azokat).

- Folyamatra jellemző elemek
    - Memória címtartomány
    - Globális változók
    - Nyitott fájlok
    - Gyermekfolyamatok (ha vannak)
    - Függőben lévő riasztások
    - Szignálok, jelzések és azok kezelői

- Szálra jellemző elemek
    - Utasításszámláló
    - Regiszterek
    - Verem
    - Szál állapotok
    - Adminisztratív információk

2. Folyamatok együttműködése

   A multiprogramozott rendszerekben a folyamatok együttes lefutása, egymáshoz való viszonya, a köztük lévő kölcsönhatások, csatolások erőssége szerint a következő képpen alakulhat.

   - Független folyamatok

     Önmagukban szekvenciális folyamatokról van szó, melyek egymás működését nem ismerik, azt nem befolyásolják, működésük egymással nem mutat szinkronitást. Működésük tehát aszinkron, futásuk sebessége jellemzően eltér egymástól. Tulajdonképpen nincs is értelme másképp, csak önmagukban vizsgálni ezeket a folyamatokat.
   - Csatolt folyamatok

     Egymástól nem függetlenek, legalább egy erőforrást közösen használnak.
       - Versengő folyamatok

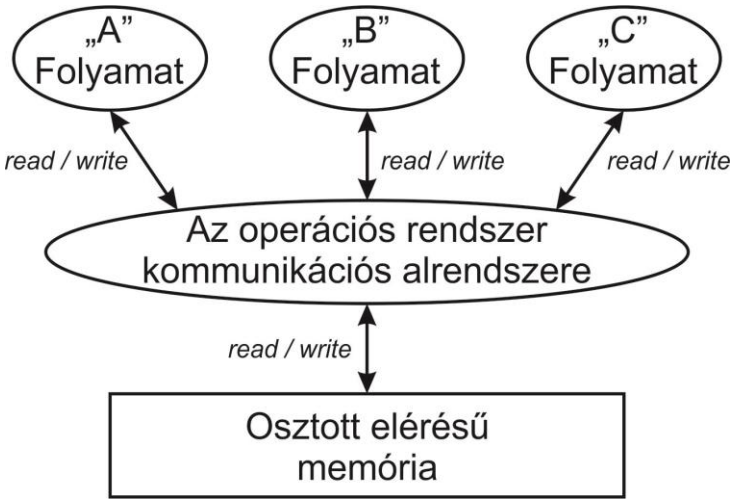
         A versengő folyamatok sem tudnak egymásról egészen addig, amíg a versenyzés tárgyát képező erőforrás(ok) – jellemzősen periféria(k) – amelye(ke)n előbb-utóbb valahogy meg kell osztozniuk előtérbe nem kerülnek. Jellemzően user folyamatok esetében beszélhetünk versengésről.

- Együttműködő, kooperatív folyamatok
  Ez esetben egy közös cél végrehajtása az ami összeköti a folyamatokat. Szükségszerű, hogy az együttműködő folyamatok képesek legyenek adatokat cserélni egymással. A kicserélt információ mennyisége az egy bit is lehet, de akár sok megabájt is elérhet. Jellemzően kernel folyamatok esetében beszélhetünk együttműködésről.

Az együttműködés tulajdonképpen az adatcsere és az erőforrások használatának összehangolását jelenti. Amennyiben ez az összehangolás időbeli, akkor szinkronizációról beszélünk. A folyamatok közötti adatcserét, információcserét röviden kommunikációnak hívjuk. (Tulajdonképpen az információ is adat.)
Ahhoz hogy két folyamat képes legyen együttműködni először is információt kell cserélniük a későbbi adatcsere időzítésével, módjával és tartalmával kapcsolatban.

Az információcsere a következő két módon történhet:

- Közös memórián keresztül



  A közös memóriát (ha nem is mindig pont egyszerre) több párhuzamos futású folyamat is írja és olvassa. A működés alapja a következő kétféle közös memória modell valamelyike:
  - RAM modell (Random Access Machine)
    A közös memória rekeszekből áll, melyek egyenként címezhetőek. A címzés, az írás és az olvasás alapegysége tehát a memóriarekesz. Egy memória rekeszt egy időben csak egy folyamat használhat. Az írási és olvasási műveletek egy memóriablokkra, azaz rekeszre vonatkoznak. A párhuzamosság klasszikus DRAM memória modulok felhasználásával, ebben az esetben leginkább csak virtuálisan valósítható meg.

- PRAM modell (Pipelined Random Access Machine)

  Ez a valódi multiprogramozott megoldás, hiszen ebben az esetben a közös memória ugyanúgy rekeszekből áll, de egy rekesz egy időben (valós időben) egyszerre több folyamat által is elérhető – akár írási akár olvasási célból. A PRAM algoritmus pusztán csak DRAM modulok felhasználásával nem oldható meg, a modell működéséhez SRAM-ok implementációja is szükséges a megfelelő működtető logikai egységgel és algoritmussal (pl. CRCW) együtt.

  Egy memória rekesz szempontjából a párhuzamos működés három féle ütközést eredményezhet:

  - Olvasás–Olvasás ütközés

    Ez problémát nem okozhat, mindkét olvasás ugyanazt az eredményt kell hogy tartalmazza.
  - Olvasás–Írás / Írás–Olvasás ütközés

    A memória rekesz tartalma csak az új adattal íródhat felül. Az olvasás értéke pedig vagy a korábbi vagy az új tartalommal fog megegyezni.
  - Írás–Írás ütközés

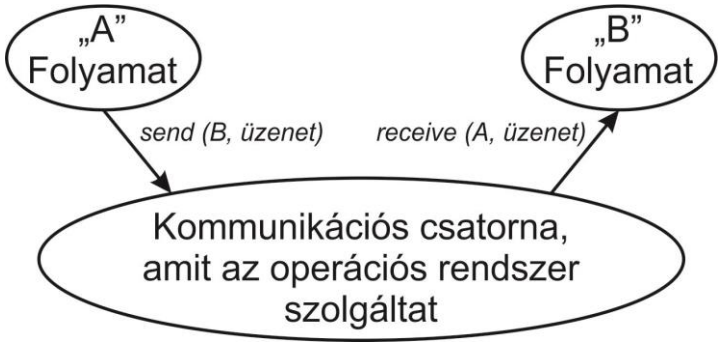
    A memóriarekesz ez esetben biztosan elveszíti régi tartalmát, és új értéke pedig az egyik írási folyamat értéke lesz.

  Látszik, hogy olvasások párhuzamos működése nem okoz gondot, de egy kis hazárdjáték van a fenti írási mechanizmusok párhuzamos működésekor. Jellemző megoldás egyrészt az EREW PRAM (Exclusive Read Exclusive Write), ami tiltja a párhuzamos műveleteket, így visszavezet a RAM modellhez, másrészt a CREW PRAM (Concurrent Read Exclusive Write) megoldás, ami lehetővé teszi a párhuzamos olvasást, de tiltja a párhuzamos írást, valamint a CRCW PRAM (Concurrent Read Concurrent Write) megoldás, ami maga a klasszikus PRAM, azaz mindent enged párhuzamosan. CRCW PRAM esetében feltétlenül szükséges a szinkronizáció, vagy kölcsönös kizárás (ezekről a későbbiekben lesz szó). Az előbbi megoldások vegyes használata a jellemző, hiszen más és más szituációkban az egyik vagy a másik megoldás használata tűnik optimálisnak.

  A Pipelined elnevezés pontosan arra utal, hogy fizikailag a memóriához egyszerre beérkező utasítások hatásukat tekintve egymás után érkező

utasításként értelmeződnek, egymást közvetlenül nem zavarják meg (nem interferálódnak), úgy viselkednek, mintha valamilyen sorba következtek volna egymás után. Az ilyen műveleteket atomi, azaz oszthatatlan műveletnek nevezzük.

- Üzenettovábbítással



Az adatcsere ez esetben nem egy közös memória rekeszen keresztül történik meg, hanem egy kommunikációs csatornán keresztül. A kommunikációs csatornát az operációs rendszer biztosítja. Ebben az esetben a két egymással kommunikálni szándékozó folyamatnak a megfelelő parancsok segítségével kell a csatornát elérnie.

- Send / Adat Küldés

  A küldő folyamat ennek segítségével adja át az adatokat – a megcímzett folyamat azonosítóját, és magát az üzenetet, azaz az adattartalmat – a csatorna számára. A küldő folyamat adatai a küldő folyamat saját kizárólagos memória helyén tárolódnak. A küldött adatokat a csatorna átveszi, ha szükséges rövid ideig tárolja.

- Receive / Adat Fogadás

  A címzett, azaz a fogadó folyamat ennek segítségével jelzi a csatorna részére, hogy készen áll az adatok fogadására. Amennyiben a csatorna tartalmaz a fogadó folyamat számára adatokat, akkor azokat a küldő folyamat azonosítójával együtt megkapja. A vevő folyamat adatai a vevő folyamat saját kizárólagos memória helyén tárolódnak.

Az adatforgalom ellenőrzésével, illetve a küldési és fogadási szándék jelzésével (például azzal, hogy a fogadó folyamat kér, vagy a csatorna kéretlenül is áttolja az adatot) nem most foglalkozunk.

3. Kommunikáció

A kommunikáció elengedhetetlen feltétele, hogy a két kommunikálni szándékozó folyamat "tudjon egymásról". Felmerülnek nyilvánvaló kérdések, melyeket meg kell tudni válaszolni, mielőtt bármilyen adatcserére is sor kerülhetne:

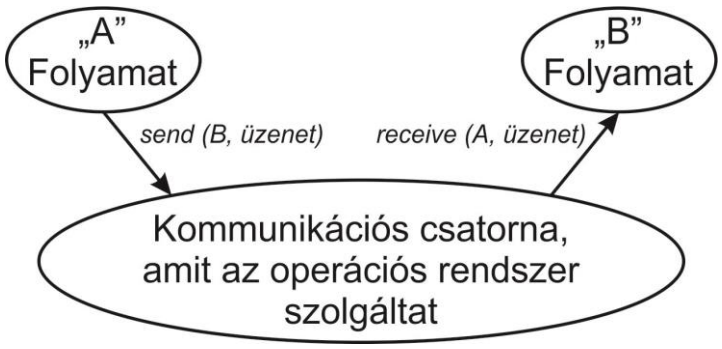
- Hogyan hivatkozzunk a kommunikációs partnerre (azaz a másik folyamatra), ismerjük-e az azonosítóját, vagy csak a funkcióját ismerjük?
- Közvetlenül fogjuk elérni a partnert, vagy valamilyen közbülső csatornán, postaládán keresztül, azaz ismerjük-e a kommunikáció adott partner szempontjából preferált vagy esetleg egyetlen lehetséges módját?
- Lehetséges-e egyszerre több partner számára elküldeni ugyanazt az üzenetet?
- Lehetséges-e valamilyen módon meggyőződni az üzenetküldés sikerességéről?
- Mi történik, ha hamarabb akarunk fogadni egy üzenetet, mint ahogyan az elküldésre (csatorna, postaláda) került?

A válaszok alapján a kommunikáció két nagy típusra választható szét:

- A partner címzése, megnevezése, elérése szerinti kommunikáció
- Az elvégzendő művelet hatása (szemantikája) szerinti kommunikáció

A partner címzése, megnevezése, elérése vonatkozásában a kommunikáció lehet:

- Közvetlen (direkt) kommunikáció

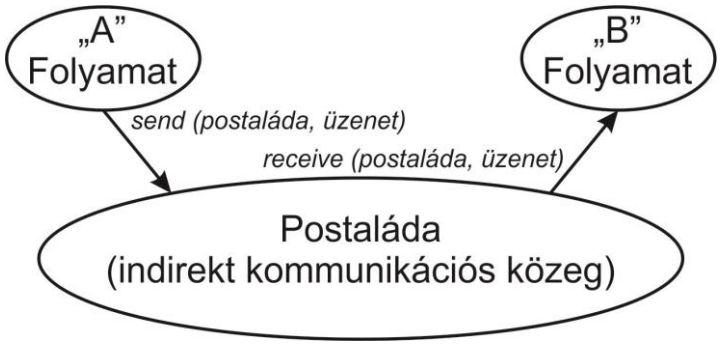


Nem véletlen, hogy itt az előző ábra ismétlődik, hiszen gyakorlatilag itt is ugyanarról van szó, csak a megközelítés más. A közvetlen kommunikáció két olyan folyamat között történhet, melyek kölcsönösen ismerik egymás azonosítóját. Azaz a küldő folyamat képes megnevezni a fogadó folyamatot, a fogadó folyamat pedig képes megnevezni a küldő folyamatot. A folyamatokra jellemző a szinkronizáció (amiről részletesen a későbbiekben lesz szó) ebben a kommunikációs formában.

A műveletek szintaktikája:

- SEND (a megcímzett folyamat azonosítója; üzenet)
- RECEIVE (a küldő folyamat azonosítója; üzenet)

- Közvetett (indirekt) kommunikáció



Ebben az esetben a kommunikálni szándékozó folyamatok, ugyan ismerik egymás folyamat azonosítóját, mégsem közvetlenül egymást nevezik, azaz címzik meg, hanem a közvetítő csatornát, vagy a postaládát. A postaláda jellemzően FIFO szervezésű, meghatározott számú (jellemzően két) folyamat használhatja, és elvileg sem végtelen kapacitású. Jellemző a több postaláda jelenléte a rendszerben. Nem szükséges a szinkronizáció ebben a kommunikációs formában.

A műveletek szintaktikája:

- SEND (a postaláda azonosítója; üzenet)
- RECEIVE (a postaláda azonosítója; üzenet)

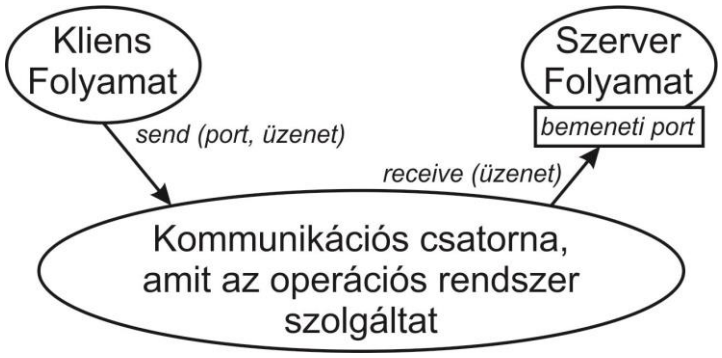
- Aszimmetrikus kommunikáció

Ebben az esetben a kommunikációban részt vevő folyamatok egyike (a küldő vagy a fogadó) kommunikációs portot használ. A port megnevezés általános megközelítésben, például TCP/IP hálózati kapcsolat esetében egy adott hoszt konkrét feladatot végrehajtó folyamatát vagy részfolyamatát jelenti. Ez esetben is a port egy feladat tipizációt, megjelölést jelent. Vételi port használata esetén a küldő oldal a vételi oldalon elvárt feladat típust (folyamat típust) jelöli csupán meg. Ebből az következik, hogy a küldő és a fogadó folyamatnak közvetlenül nem kell egymást ismerne.

Mit is jelent mindez a gyakorlatban? Ha egy folyamat például nyomtatni szeretne, és annak az információnak a birtokában van, hogy (ebben az esetben például) a 99-es porton nyomtatók várják a feladatokat, akkor elegendő megcímeznie a 99-es portot, és az első szabaddá váló nyomtató ki

fogja nyomtatni az anyagot. (Nyilván ez a példa akkor igaz, ha egy helységben több, egyforma nyomtató várakozik az anyagok kinyomtatására. Például ilyen eset lehet egy nagyvállalatnál a céges bérjegyzék nyomtatása.)

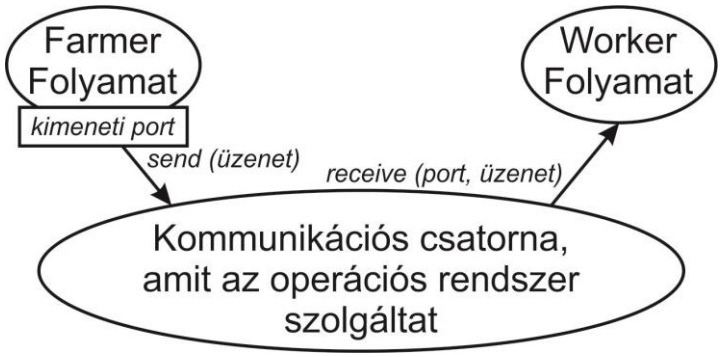
A kliens-szerver kiszolgálási modell:



Vevő oldali port használat esetében a kliens-szerver kiszolgálási modell érvényesül, azaz egy szerver, egy kiszolgáló várja a sorban beérkező kéréseket.
A műveletek szintaktikája vételi port esetében:

- SEND (a vételi oldal portjának azonosítója; üzenet)
- RECEIVE (üzenet)

A farmer-worker kiszolgálási modell:



Küldő oldali port használat esetében a farmer-worker kiszolgálási modell érvényesül, azaz a munkákat sorban kiosztó farmer (szerencsésebb a menedzser szó használata) várja a feladatokra jelentkező munkásokat.
A műveletek szintaktikája küldő oldali port esetében:

- SEND (üzenet)
- RECEIVE (a küldő oldal portjának azonosítója; üzenet)

- Csoportkommunikáció
  Jellemzően a küldő oldal ez esetben a vevő oldalon nem egy folyamatot céloz meg, hanem folyamtok csoportját. A kijelölt csoport minden tagja meg kell hogy kapja az üzenetet (multicasting). Létezik olyan eset is, amikor az összes folyamat címzetté válik, azaz az össze folyamat megkapja az üzenetet (broadcasting).
  A csoportkommunikáció szintaktikája:
    - SEND (csoport azonosító; üzenet)

Az elvégzendő művelet hatása (szemantikája) alapján a kommunikáció további tagolása már túlmutat az operációs rendszerek tantárgy tematikáján. Amit ezzel kapcsolatban meg kell említeni, az a műveletet végrehajtásának ellenőrzése, ami történhet állapotkódok cseréjével vagy hiba megszakításokkal és az azt követő hibakezelési rutin végrehajtásával. Hibajelzés történhet bizonyos időkorlátok lejárta, a kommunikáció megszakadása, illetve csak részbeni teljesülése esetén. Az időkorlátok használata alapvető igény, hiszen egyik folyamat sem várakozhat végtelen ideig. Az adatcsere nyugtázása abból szempontból is fontos, hogy a küldő folyamat az érintett memória rekesz tartalmát nyilván csak azután írhatja felül, ha már meggyőződött arról, hogy az ott tárolt és onnan továbbított adatokra a továbbiakban neki nincs szüksége, azaz azt a megcímzett folyamat sikeresen felhasználta.

A kommunikációs csatorna jellemezhető az átviteli közeg típusával, a kapcsolat lehetséges irányaival (Simplex, Half Duplex, Full Duplex), illetve a csatornát használó folyamatok számával. Ezen jellemzően logikai paramétereken túl fizikai paraméterek is jellemzik a csatornát (amely így már túlmutat a felhasználó számítógépén). További fontos paraméter, hogy a csatorna milyen hosszú ideig és maximálisan mekkora méretű üzenetek tárolására alkalmas, más szóval mekkora a csatorna kapacitása, és mekkora a csatorna megbízhatósága. Ezen kívül szempont a csatorna késleltetésének minimális értéke is, ami azt mutatja meg, hogy optimális esetben mekkora késleltetést okoz egy elvileg azonnali (tehát tárolás nélkül továbbított) üzenet továbbítása.
Távoli kommunikáció esetén az időkorlátok használata elkerülhetetlen. Szintén szükség lehet az átvitel ellenőrzésére is. (Ez a témakör innentől már a Számítógép Hálózatok tantárggyal mutat több kapcsolatot...)