Operációs rendszerek:

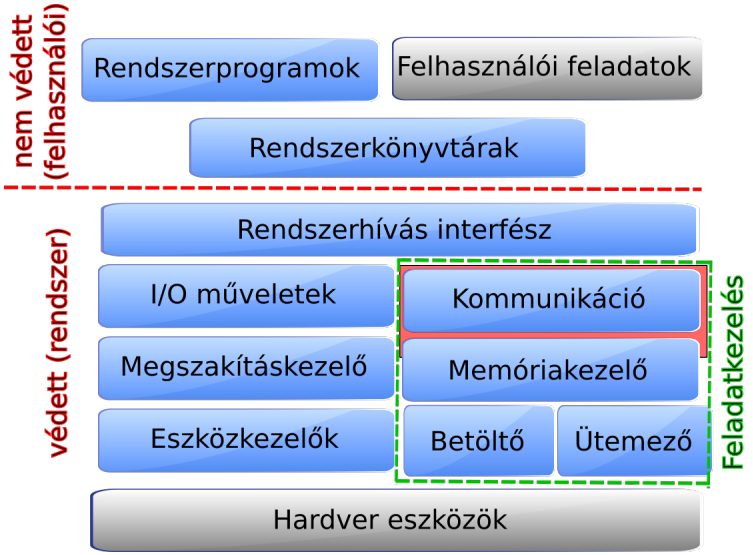
taszkok kommunikációja

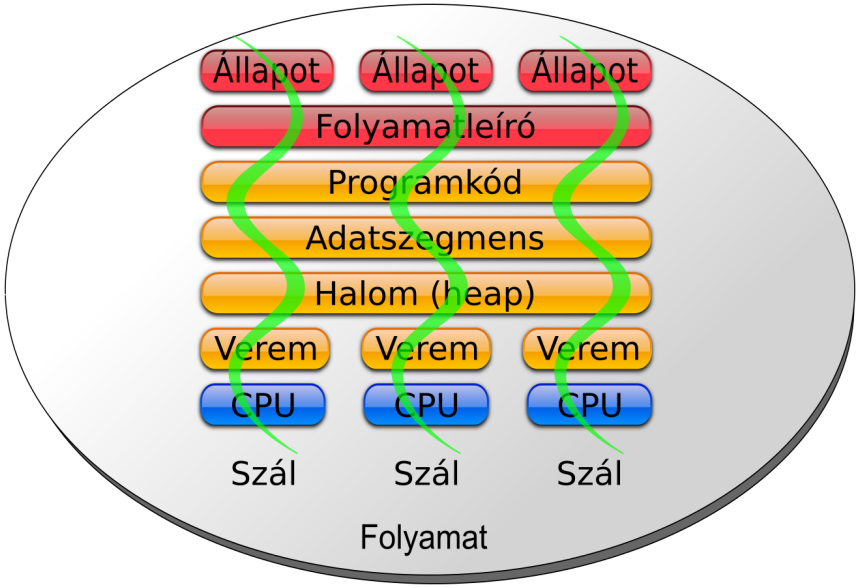
Mészáros Tamás
http://www.mit.bme.hu/~meszaros/

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Az eddigiekben történt...

- A feladatokat taszkok végzik el
  - szétoszthatnak részfeladatokat
  - egyesíthetnek részeredményeket
- Absztrakt virtuális gép
  - szeparálja a taszkokat
  - gátolja az együttműködést





- Taszkok megvalósítása
  - folyamat
  - szál
- Szál
  - szekvenciális működésű taszk
  - egy folyamaton belüli más szálakkal közös memóriát használ

A kommunikáció alapvető sémái

- Közös memórián keresztül
  - PRAM modell
  - versenyhelyzetek
  - megvalósítási példák
    - folyamaton belüli szálak
    - POSIX osztott memória

- Üzenetváltás segítségével
  - adatátviteli rendszer (nagyon sokféle)
  - alapvető műveletei:

```
Küld(címzett, adatcím[, adatméret])
Fogad(címzett, puffercím[, adatméret])
```

  - megvalósítási példák
    - hálózati kommunikáció
    - távoli eljáráshívás
    - elosztott rendszerek
    - mikrokernel

A PRAM (pipelined RAM) modell

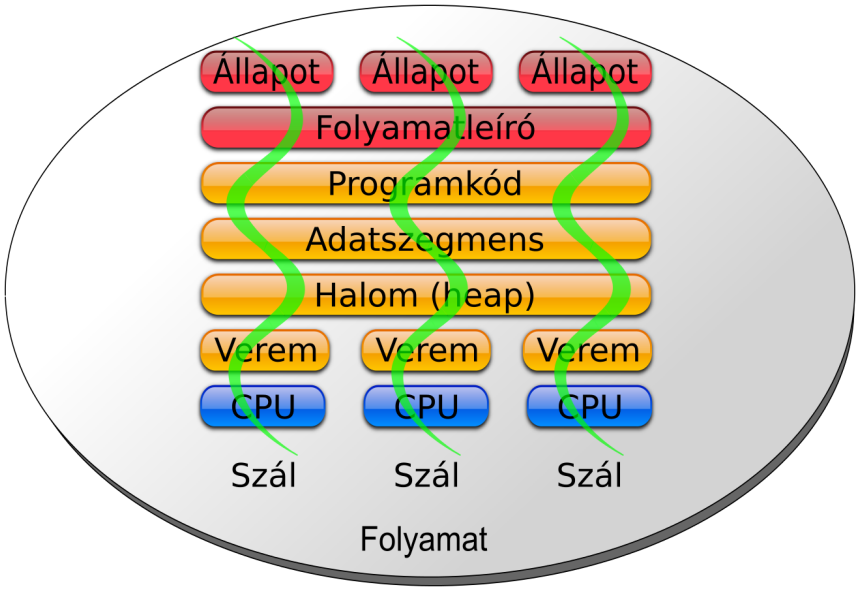
- A taszkok párhuzamosan használnak egy közös memóriaterületet
  - a taszkok műveletei egymástól függetlenek (random access)
  - ütközhetnek is

- Az ütközésfeloldás szabályai
  - *olvasás–olvasás* mindkettő a memória tartalmát adja vissza
  - *olvasás–írás* az olvasás vagy a régi, vagy az új értékét adja vissza
  - *írás–írás* a két érték valamelyike kerül a memóriába

- A szabályok hatása
  - a műveletek nem hatnak egymásra (nem keverednek)
    (**pipelined**: sorba rendezett, nem kevert)
  - a párhuzamos kérések sorrendje nem definiált
    bizonytalan, hogy melyik hajtódik előbb végre (de nem keverednek)
  - a párhuzamos kéréseket össze kell hangolni
    → **szinkronizáció**

PRAM: folyamaton belüli szálak

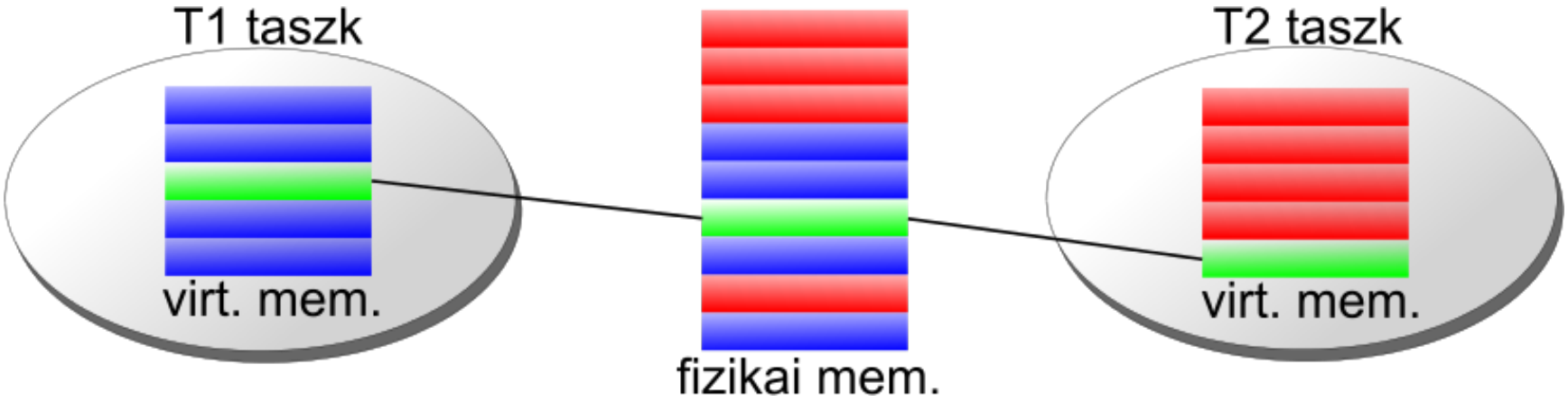
- Adatcsere globális változókkal
  - az OS egy folyamaton belül közös memóriát biztosít a szálaknak
  - a kommunikáció nem az OS fennhatósága alatt történik
  - a programozó alakítja ki a működést



Példa: munkamegosztás szálak között

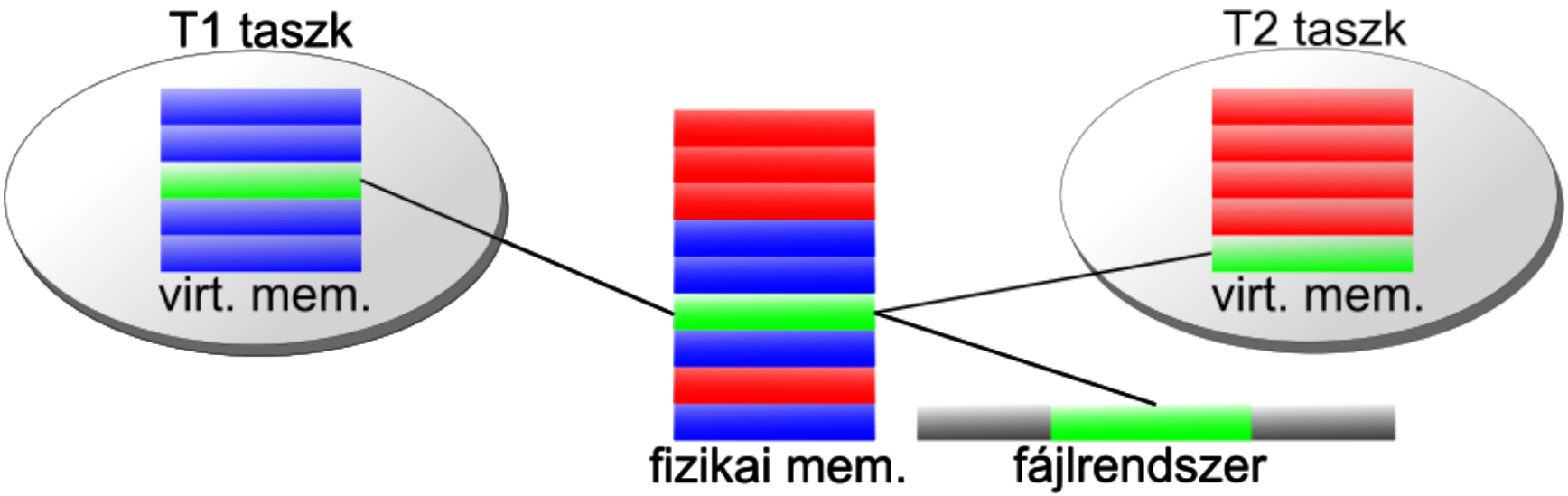
```
struct data_type data[N];

compute (void *part) {
  … műveletek a data[] *part részén ...
}

pthread_t *tid;

for (i=0; i < N; ++i) {
  // szálak indítása
  pthread_create(&tid[i],NULL,compute,&data[i]);
}

for (i=0; i < N; ++i) {
  // megvárjuk, míg elkészülnek
  pthread_join(tid[i], NULL);
}
```

PRAM: osztott memória (shared memory, SHM)



- Értékelése
  - nincs rendszerhívás, nulla rezsiköltség
  - rendkívül gyors kommunikációt biztosít (zero-copy)
  - korlátos kapacitással rendelkezik
- Megvalósítása
  - szabvány: POSIX Shared Memory
  - Unix, Windows (a kernel része, a memóriakezelés támogatja)
  - felhasználói címtérben, pl. C++ könyvtárként

PRAM: memóriába ágyazott fájlelérés



- Értékelése
  - lehet ilyen az SHM, ahol az OS nem támogatja
  - a klasszikus fájlrendszer interfész helyett is jó
- Megvalósítása
  - széles körben elérhető (`CreateFileMapping()`, `mmap()`)
  - az OS virtuális memóriakezelője végzi a leképezést
  - sokféle programozási környezetben elérhető

Üzenetváltásos kommunikáció

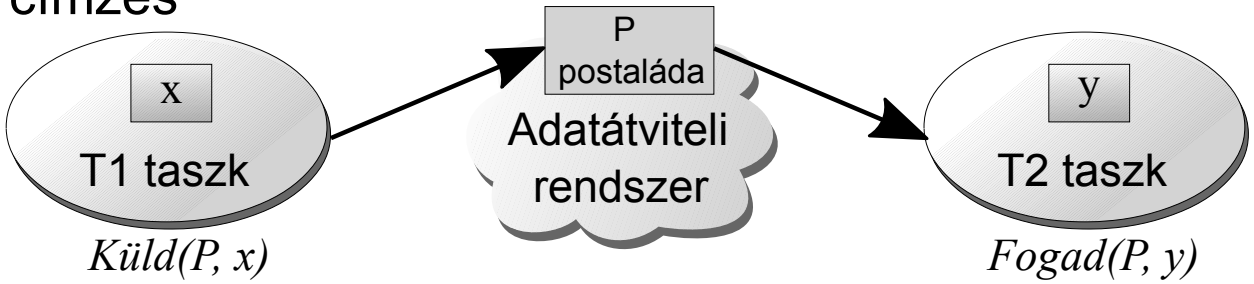
Az üzenetváltásos kommunikáció kérdései

- Címzés
  - direkt vagy közvetítővel?
  - egy vagy több címzett?
  - a címzett szűrheti a feladókat?
- Szinkronitás
  - hol az adat a `Küld()` művelet visszatérésekor?
  - hogyan értesül a feladó a kézbesítésről?
  - mi történik, ha kiadjuk a Fogad műveletet, de az adat még nem érkezett meg?
  - ha fogadtunk az adatokat, kell visszaigazolást küldenünk?
- Az adatátvitel szemantikája
  - az adat a küldőnél is megmarad, vagy a fogadónál lesz, esetleg mindkettőnél?
- **Teljesítmény, megbízhatóság**
  - milyen adatátviteli sebesség érhető el és mekkora az üzenetek késleltetése?
  - hány és milyen méretű üzenet küldése lehetséges?
  - ki veszi észre és mi történik adatátviteli hiba esetén?

Alapvető címzési módszerek (lásd még Komháló 1.)

- Direkt címzés

- Indirekt címzés



- Aszimmetrikus (küldő oldalon direkt) címzés

- Többszörös (multicast, broadcast)

Szinkronitás

- Szinkron adatátvitel
  - A `Küld()` és a `Fogad()` blokkoló művelet
    - a taszk várakozó állapotba kerül
  - egyszerűen programozható
  - az eredmény és az esetleges mellékhatások is beérkeznek
  - megszakad a taszk futása, átütemezés
  - Denial-of-Service (DoS) támadások...
  - időtúllépés...

- Aszinkron adatátvitel
  - a műveletek nem blokkolnak
  - a taszk tovább futhat, de a műveletek eredménye még nem érhető el
  - az esetleges mellékhatások, hibák sem jelentkeznek
  - a műveletek eredményeit ellenőrizni kell
  - **a még nem kézbesített üzeneteket átmenetileg tárolni kell (pl. a kernelben)**
  - hasznos, ha van más csinálnia a taszknak
  - nem hasznos, ha ciklusban ellenőrzi a küldést/fogadást

Adatátviteli szemantika, adatbirtoklás

- A kommunikáció résztvevői meddig férnek hozzá az adatokhoz?
- Kizárólagosan vagy megosztva birtokolják azokat?

  - **Másolat** (copy semantics)
    - a küldő és a fogadó is saját példánnyal rendelkezik
    - a módosítások hatása lokális

  - **Megosztás** (share semantics)
    - ugyanazt használják, jogosultság lehet különböző
    - a módosítás hatása globális (szinkronizáció!!!)
    - csökkentheti az adatmozgatást (azonos rendszeren belül)

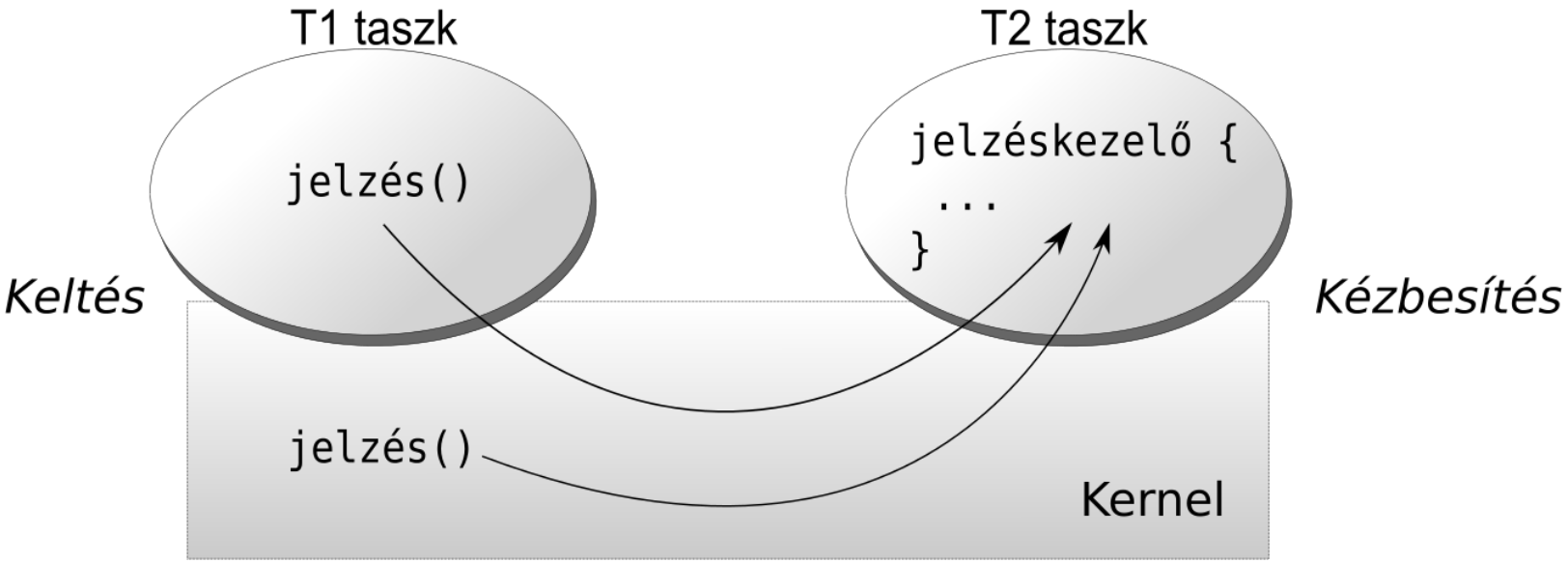
  - **Mozgatás** (move semantics)
    - a küldő elveszíti a hozzáférését az adatokhoz, amikor elküldi őket
    - megvalósítható megosztással és a jogosultságok elvételével is

- Az „**adatbirtoklás**” fontos kérdés a párhuzamos programozásban
  - szinkronizáció
  - munkamegosztás

Direkt és aszimmetrikus kommunikációs megoldások

- Jelzés (signal)
  - értesítés eseményekről: taszk → taszk, kernel → taszk
- Hálózati kommunikáció (socket communication)
  - adatátvitel akár rendszerek között is
  - aszimmetrikus kommunikáció kliens-szerver modell szerint
  - sokféle protokoll és címzés
- Távoli eljáráshívás (remote procedure call)
  - egy másik taszk programjában levő eljárás meghívása
  - aszimmetrikus kommunikáció kliens-szerver modell szerint
  - hálózati kommunikációra épül
  - adatkonverziót is végez

Jelzések

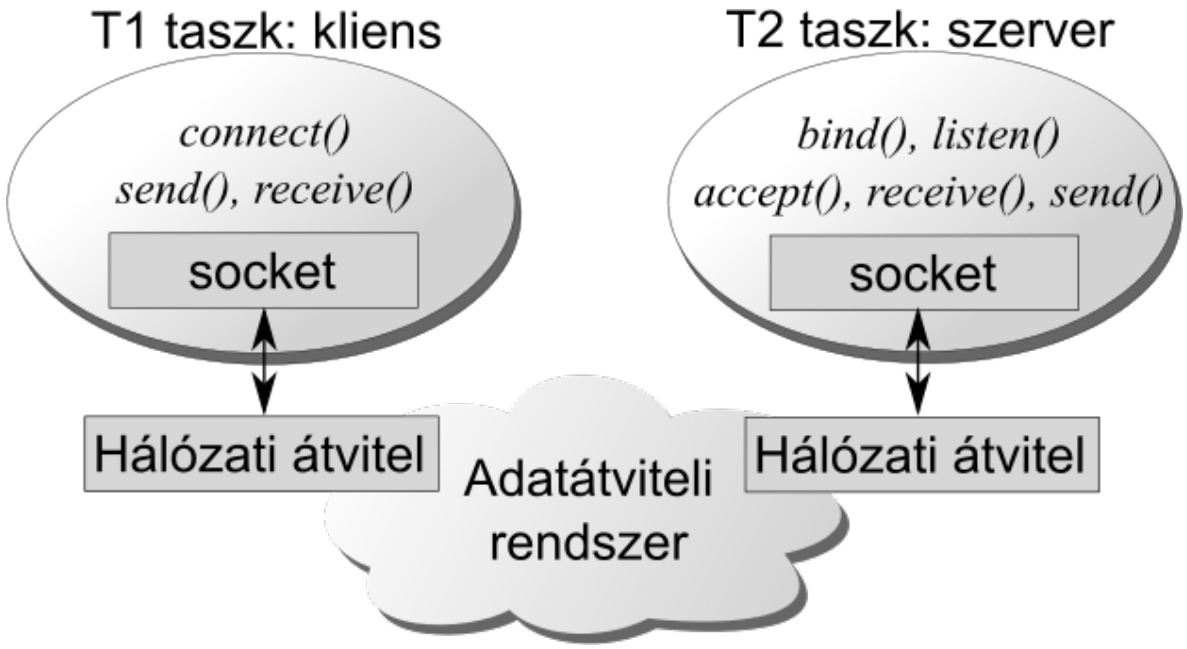


- Célja
  - értesítés eseményekről
- Kétfázisú működés
  - keltés
  - kézbesítés (kezelés)
- Típus
  - felhasználói
  - rendszer
    - kivételek, értesítések, riasztások
- Kezelés
  - kezelőfüggvény
  - figyelmen kívül hagyás

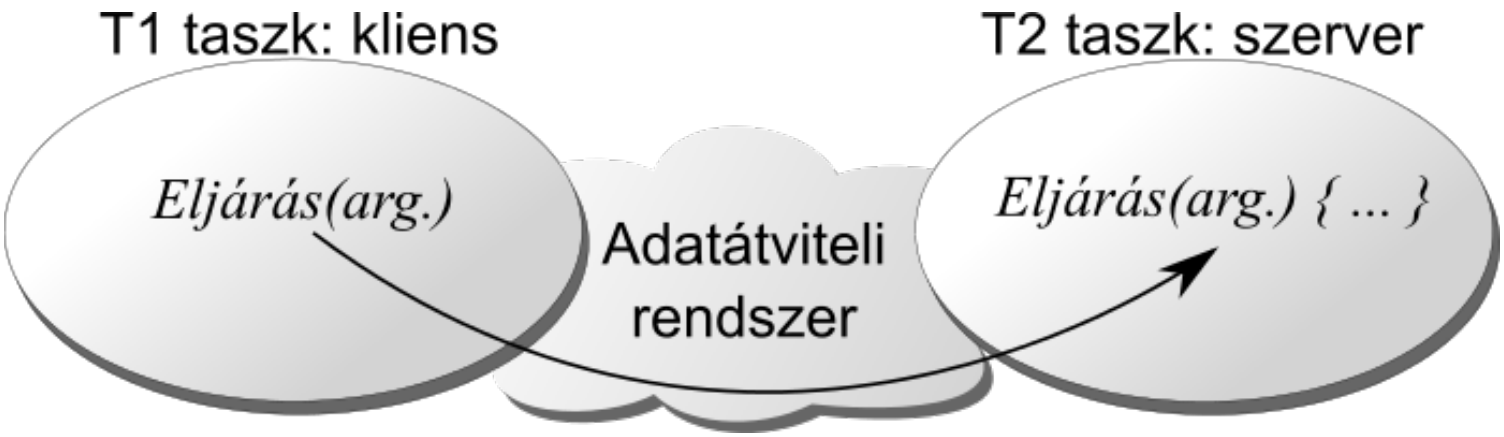
Hálózati kommunikáció (ismétlés, Komháló 1.)

- Címzés és hálózati protokollok
  - gépen belül (localhost, `127.0.0.1` ill. `::1`) és gépek között
  - többféle címzés (*cast) és protokoll (pl. IP, TCP és UDP)

- A kommunikációs csatorna leírója: **hálózati csatoló (socket)**
  - a kommunikációs végpont (logikai) azonosítója a taszkokban



A távoli eljáráshívás (remote procedure call, RPC)



Példa Open Network Computing (korábban Sun) RPC

Indirekt kommunikációs megoldások

- Postaláda (mailbox)
  - véges számú (sokszor csak egyetlen) üzenet
  - az üzenetek mérete korlátos
  - a postaláda címezhető, nem a fogadó (indirekt)
- Üzenetküldés (message queue / message passing)
  - korlátos méretű adathalmaz átküldése
  - sokféle cél és implementáció (helyi OS, elosztott rendszerek, HPC stb.)
  - jellemzően indirekt (csatornán keresztüli)
  - működhet rendszerek közötti hálózaton is
  - sokféle implementáció message oriented middleware, RabbitMQ, Java MS, MSMQ
  - többféle szabvány: POSIX üzenetsorok, AMQP, MQTT, HPC MPI stb.
- Csővezeték (pipe)
  - végtelen (gyakorlatilag korlátos) adatmennyiség továbbítására
  - folytonos adatküldésre és fogadásra alkalmas (nincs üzenethatár)
  - a csővezeték címezhető (nem a fogadó)
  - egyszerre több vevő is lehet

Kommunikációs megoldások teljesítménye

- A PRAM közvetlenül a virtuális memóriakezelésre támaszkodik
  - nincs felesleges kernel réteg
  - nincs adatmozgatás
  - nincs extra rezsiköltség

  nincs +késleltetés, nagy adatátviteli sebesség (zero copy)

- Az üzenetváltásos modell kommunikációs infrastruktúrára épít
  - mozgatja az adatokat (többször is)
  - esetenként konvertálja is
  - átmenetileg tárolhatja is

  késleltetés okoz, az adatátviteli sebesség jelentősen csökken

- Hol okoz ez igazán gondot?
  - pl. a mikrokernelekben
  - pl. a nagy teljesítményű (HPC) rendszerekben

Mi okozza a késleltetést és a lassulást?

- Rendszerhívások (`Küld()` / `Fogad()`)
  - megszakítás
  - kontextusváltás
  - esetenként átütemezés
- Adatmásolás
  - taszk1 címtér → kernel → taszk2 címtér
- Kontextusváltás
  - T1 és T2 taszkok párbeszéde
    - T2 Fogad() → blokkolódik (kontextusváltás)
    - T1 Küld() → blokkolódik (kontextusváltás)
    - T2 felébred → átütemezés → Fogad() ….. Küld() → blokkolódik
    - T1 felébred → átütemezés → Fogad() ….. Küld() → blokkolódik
    - ...
  - sok átütemezés és kontextusváltás (a TLB állandóan kiürül)

A OS védelmi mechanizmusai miatt csökken a hatékonyság.

Teljesítménynövelés modern mikrokernelekben

- A mikrokernelekben kritikus a problémák megoldása

  minél kevesebb
  - adatmásolás
  - kontextusváltás
  - átütemezés

  a kernel üzenetalapú kommunikációja miatt

- A kontextusváltás és az ütemezés rezsiköltségének csökkentése
  - **direkt kontextusváltás**
    - nem az ütemező dönt a következő futtatandó taszkról
    - a Küld() – Fogad() séma határozza meg az átütemezést
      - nem fut az ütemező (nincs rezsiköltsége)
      - kicsi lesz a késleltetés a küldés és vétel között
  - **lazy queueing**
    - Küld() – Fogad() párbeszédek kezelése
    - átmenetileg felfüggeszti a taszkok állapotváltozásának adminisztrációját
      - a taszkok nem mozognak a Fut, FK és Vár sorok között
      - ha véget ér a párbeszéd, helyreállítja a sorok tartalmát

Az adatátvitel gyorsítása

- Hogyan gyorsítható?
  - az adatmennyiségtől függ...
  - mekkora mennyiségről van szó?
    - mikrokernel belső működése
    - függvényhívások

- Nagyon kevés adat (< ~ 16 byte)
  - a processzor regisztereiben
  - nincs sok erre a célra alkalmas regiszter
  - a teljes kommunikáció gyorsulása: ARM11: 10%, CortexA9 (újabb) 4%

    x86-on akár ronthat is – Vajon miért csökken vagy negatív a hatása?

- Közepes méretű adathalmaz (kb. 16-64 byte)
  - **virtuális regiszter tároló**
    - elférhet a regiszterekben
    - erre a célra allokált memóriaterületen PRAM modell szerint
  - hardverfüggő módon implementálja a kernel (lásd pl. ARM)

„Nagyobb” adathalmazok átvitele lokális gépen

- Osztott memóriás (SHM) átmeneti tárolással
  - Küldő → SHM → Fogadó
  - korlátozott a mérete, két másolás (Copy-in / Copy-out)
- Egy másolással (single-copy)
  - Taszk-taszk memóriamásolás (pl. KMEM)
    - Küldő → Fogadó direkt másolás rendszerhívással
- Másolás nélkül (zero-copy)
  - Lapmegosztás (pl. XPMEM)
    - egy taszk megoszthat memóriatartományt másokkal (kernel-támogatással)
  - Távoli memóriaelérés (pl. CMA)
    - egy taszk elérheti egy másik memóriatartományát (kernel-támogatással)
    - ötlet: /proc/<PID>/mem olvashatóvá tétel (teljes?)
- Hardvertámogatással (kernel DMA Engine, *RDMA*)
  - számítási csomópontok között is működhet
  - pl. HPC, SMP környezetben

A kommunikáció alapvető sémái (összefoglalás)

- **PRAM modell**
  - közösen használható memória
  - az egyidejű műveleteket nem keveri valamilyen sorrendbe állítja őket
- Szinkron adatátviteli műveletek
- Beépített címzés nincs
  - közvetett módon kialakítható
- Alkalmazási példák:
  - folyamaton belüli szálak
  - **osztott memória**
- Előnyök
  - nagyon gyors adatcsere
  - egyszerűen használható
  - beállítás után nincs rezsiköltség
- Hátrányok, kockázatok
  - R-W és W-W konfliktusok
    → **szinkronizáció** szükséges
  - korlátos méretű

- **Üzenetalapú rendszerek**
  - adatátviteli rendszerrel működik
  - Küldés és Fogadás műveletek
  - az egyidejű műveleteket nem keveri valamilyen sorrendbe állítja őket
- Adatátvitel: szinkron / aszinkron
- Címzés
  - direkt, indirekt, többes (*cast)
- Alkalmazási példák:
  - postaláda
  - csővezeték
  - üzenetsor
  - **hálózati kommunikáció**
  - **távoli eljáráshívás**
- Előnyök
  - széleskörű elérhetőség (hálózat is)
- Hátrányok, kockázatok
  - kezelendő kommunikációs hibák