Feladatok (task) együttműködése

dr. Kovácsházy Tamás

6. anyagrész
Üzenet alapú kommunikáció

Méréstechnika és
Információs Rendszerek
Tanszék

Visszatekintés, kommunikációs módszerek

- Közös memórián keresztül (RAM v. PRAM modell):
  - Egy folyamaton belül futó szálak esetén (közös memória).
- Üzenetekkel:
  - Nincs közös memória.
    - Folyamatok kommunikálnak egy operációs rendszeren belül.
    - Elosztott rendszer.
  - Vagy pl. mikrokernel alapú az operációs rendszer.
- Folyamatok közötti kommunikáció (Inter Process Communication, IPC)

Üzenetek

- Nem azonos a számítógép hálózatokban küldött üzenetekkel, annál általánosabb fogalom.
- Üzenet továbbítás (Message passing)
- Lehet például:
  - Rendszerhívás.
  - TCP/IP kapcsolat (TCP) vagy üzenet (UDP) gépen belül (localhost) vagy akár gépek között.
- Többnyire az alkalmazás kódjában OS API függvény/metódus hívásként jelenik meg.
- Az operációs rendszer valósítja meg szolgáltatásaival.

Kitérő...

- Szemafor, Kritikus szakasz objektum, Mutex is OS-ben kerül megvalósításra.
  - Egy folyamaton belül futó szálak kommunikációja közös memórián keresztül (gyors, alacsony erőforrás igény).
  - Kölcsönös kizárás és szinkronizáció üzenetekkel (rendszerhívással, ritka, főleg a várakozás).
  - Van overhead-je:
    - Próbálkozások: Lockless programming, transactional memory etc.
    - Nincs jó megoldás, csak kevésbé rossz (Churchill mondása alkalmazható).
    - Alkalmazás és SW architektúra függő az optimum.

Üzenettovábbítás tulajdonságai

- A közös memóriához képest:
  - Nagyobb késleltetés.
  - Kisebb sávszélesség.
  - A csatorna nem megbízható elosztott rendszerek esetén.
    - Közös memóriát 1 valószínűséggel megbízhatónak tartjuk.
      - RAM, PRAM modell.
    - Az operációs rendszeren belül történő üzenetküldésre ez már nem igaz.
      - Túlterhelés!
    - A számítógép hálózaton történő üzenetküldés definíció szerűen nem megbízható (véletlen és szándékos hibák).

Üzenetek címzése

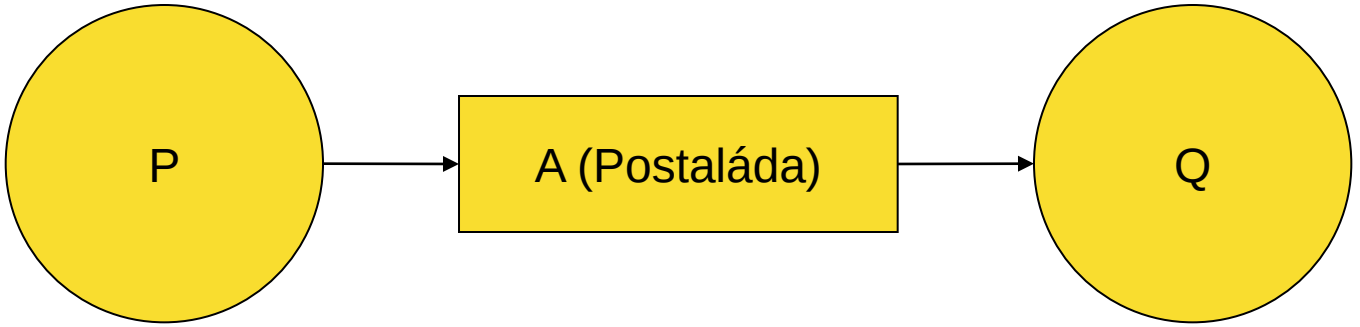
- Számítógép hálózatok...
- Egy adott folyamat (unicast cím).
- Minden folyamat (broadcast cím).
  - Pl. teljesítmény menedzsment események.
    - Standby, Hibernate, PowerOff, stb.
- Folyamatok csoportja (multicast cím).
- Egy folyamat (anycast cím).
  - Pl. Egy alkalmas folyamat, ami a kérést ki tudja szolgálni.

Direkt kommunikáció

- Szimmetrikus üzenet alapú kommunikáció.
  - send(P, message)
  - receive(Q, message)
  - P, Q folyamat azonosítók. A vevő megadja az adót
    - Q-t meg kell adni a receive() hívásakor!
  - A message egy adatstruktúra, ami az üzenetet tartalmazza.
- Aszimmetrikus üzenet alapú kommunikáció.
  - send(P, message)
  - receive(id, message)
  - P folyamat azonosító, id a küldő azonosítója. A vevő bárkitől fogad üzenetet!
    - Az id is visszatérési érték, bárki küldhet
  - A message egy adatstruktúra, ami az üzenetet tartalmazza.
- A kódban direkt hivatkozás van a másik félre.
  - Nem szerencsés...

Indirekt kommunikáció

- Egy köztes szereplőn keresztül történik a kommunikáció
  - Proxy tervezési minta.
- Ez a szereplő pl. lehet: Postaláda (Mailbox), Üzenetsor (MesssageQueue), Port, stb.
- Interface: létrehozás és megszüntetés +
  - send(A, message)
  - receive(A, message)
- Az „A” a postaláda „azonosítója”
  - Elosztott rendszerben ennek része a node (számítógép) azonosító is.



További tulajdonságok

- Minimum egy küldő folyamat.
- Minimum egy vevő folyamat:
  - 1. Vétel után törlődik (egyszer olvasható).
  - 2. Vétel után megmarad (RAM-szerű működés).
    - Osztott memória üzeneteken keresztül (Pl. SystemV Shared Memory).
- Lehet tulajdonosa:
  - Operációs rendszer.
    - Az őt használó folyamatoktól függetlenül létezhet.
  - Folyamat (annak a memória területén van).
    - A folyamattal együtt létezhet csak.

Blokkolás

- Nem blokkoló hívás = aszinkron
  - Az eredmények és mellékhatások a visszatéréskor még nem jelentkeznek (nem történtek meg).
  - Csak a végrehajtás kezdődik el a hívásra.
  - A visszatérési érték kezelése, és az eredmények és mellékhatások kezelése csak más értesítés után lehetséges:
    - Pl. esemény, jelzés (signal), callback függvény, stb.
- Blokkoló hívás = szinkron
  - Az eredmény és mellékhatások a visszatérés után jelentkeznek (megtörtént).
  - Visszatérési érték kezelés egyszerű...

Blokkolás a küldő oldalon

- Blokkoló send():
  - A send() hívás nem tér vissza, amíg az üzenetet nem vették (direkt) vagy az nem került be a postaládába.
  - Mi történik kommunikációs hiba esetén?
    - Pl. Egy hibakezelő callback függvény kerül meghívásra vagy időtúllépés történik és a send() hibával tér vissza.
- Nem blokkoló send():
  - Az üzenet lokális elküldésének indítása után azonnal visszatér.
  - Vagy nincs hibakezelés, vagy callback függvény útján értesül róla a küldő folyamat (pl. egy hibakezelő szálban).

Blokkolás a fogadó oldalon

- Blokkoló receive():
  - A receive() nem tér vissza addig, amíg nem érkezik üzenet.
  - Klasszikus példa: TCP/UDP socket listen().
- Nem blokkoló receive():
  - A receive() azonnal visszatér:
    - Ha van üzenet akkor azzal.
    - Ha nincs üzenet, akkor azt jelzi (pl. üres üzenet, null referencia, stb.).
    - Ha nincs üzenet, akkor a végtelen ciklusban az üzenetre várás busy waiting-et eredményez (zabálja a CPU időt).

Implementációk 1.

- Mailbox:
  - Indirekt kommunkáció.
  - Egy vagy több üzenet tárolása, de többnyire véges számú.
  - Operációs rendszer szintű támogatás.
- MessageQueue:
  - Indirekt kommunkáció.
  - Többnyire végtelen számú üzenet tárolására alkalmas.
    - Természetesen a rendszer erőforrásai korlátozzák.
  - Üzenet alapú middleware-ek
    - MSMQ, IBM's WebSphere MQ, Oracle Advanced Queuing (AQ), JBoss Messaging, Apache Qpid.
- Beágyazott OS-ek is jellegzetesen a Mailbox/MessageQueue megoldásokat „erőltetik”

Implementációk 2.

- TCP/IP TCP vagy UDP port:
  - Direkt kommunikáció.
  - Socket interface.
  - Gépen belül a localhost-on (127.0.0.1/8).
  - Alacsony szintű, számos más middleware alapul rajta:
    - Távoli eljárás hívás (Remote Procedure Call, RPC).
    - Távoli metódus hívás:
      - CORBA (Common Request Broker Architecture),
      - JAVA RMI (Remote Method Invocation),
      - DCOM/.NET Remoting,
      - SOAP (Simple Object Access Protocol).
    - Üzenet alapú middleware-ek (korábban volt szó).

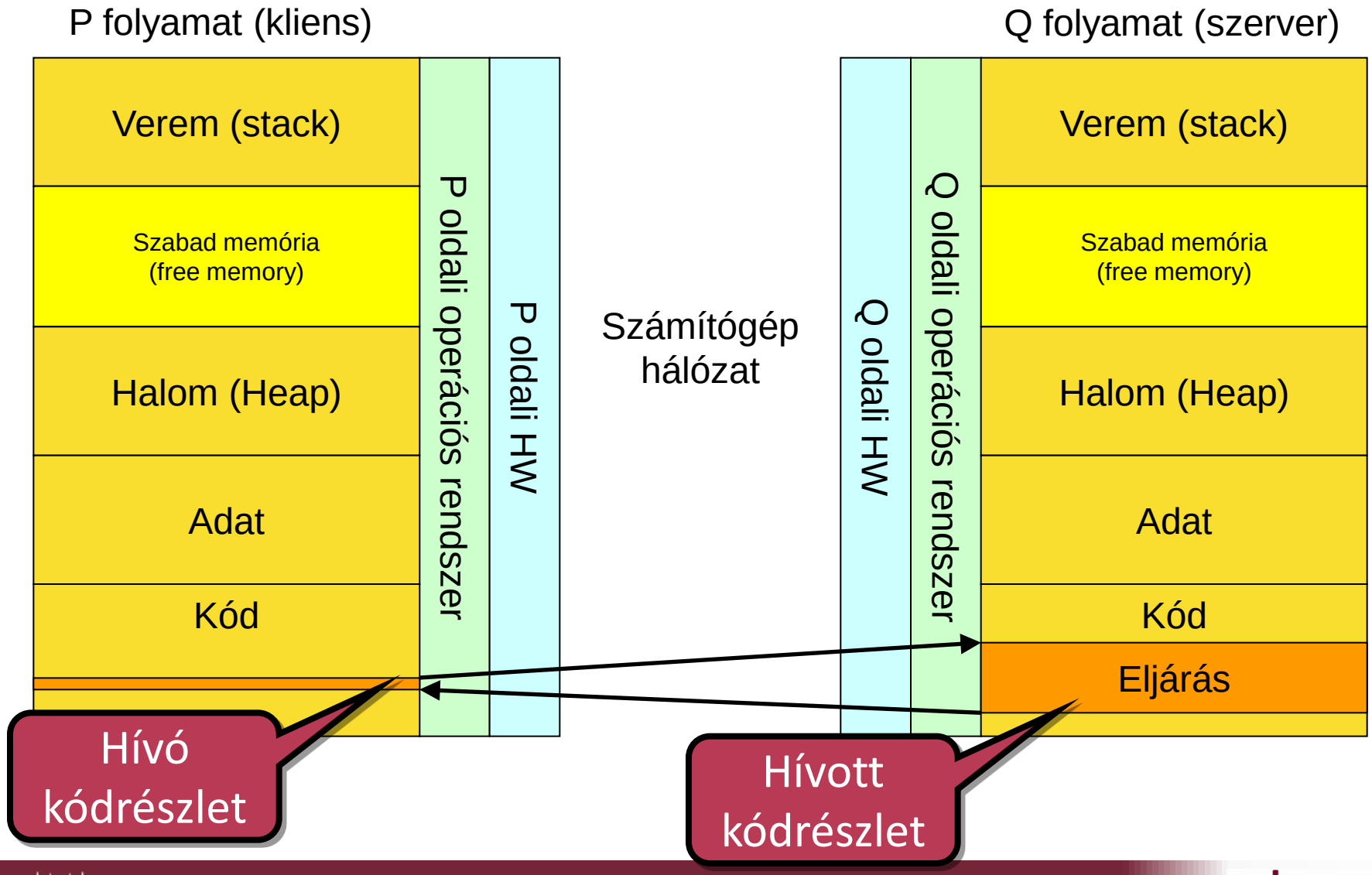
Implementációk 3.

- Különböző folyamok és csővezetékek:
  - Többnyire direkt.
  - Pl. UNIX pipe, Windows Named Pipe, RTLinux FIFO
- System V Shared Memory (UNIX, Linux)
  - Direkt.
  - Memóriaszerű interfész.
  - Később lesz szó róla a UNIX-os részben.
  - A megosztott memóriában tárolt (kommunikációra használt) adatstruktúrákra a kölcsönös kizárást a korábban tanult módokon biztosítani kell.

Távoli eljárás hívás

- Részletesen megbeszéljük, mivel:
  - Egyrészt alapja a napjainkban használt távoli metódus hívást alkalmazó megoldásoknak.
  - Nagyon tanulságos...
- Távoli eljárás hívás (Remote Procedure Call, RPC):
  - Egy másik folyamat kód memóriaterületén elhelyezkedő függvény „meghívása” a hívó folyamatból üzenetek felhasználásával.
  - A hívó fél blokkolva vár a távoli hívás lefutására.
  - A meghívott függvény az őt tartalmazó folyamat egyik vezérlési szálában fut le.

Távoli eljárás hívás architektúrája



Az RPC a programozó számára

- Hívása lényegében azonos egy „lokális” függvény hívásával.
  - Az lényegében egy programkönyvtárban található függvény.
  - Nem kell tudnia, és figyelembe vennie milyen platformon fut le.
- A ténylegesen végrehajtott függvény megírása sem különbözik egy lokálisan végrehajtható függvény megírásától.
  - Kapunk egy függvény deklarációt (interface-t), és azt meg kell valósítanunk.
  - Nem kell törődnünk azzal, hogy milyen platformról érkezik a kérés.

RPC működése 1.

- Típusos a hívás és a visszatérési érték is.
  - Strukturált adatküldés.
  - Platform függetlenség megvalósítása az operációs rendszer és a környezet (fordító vagy interpreter) feladata.
    - Szabványos formátumra konvertál minden fél, pl. Binary Encoding Rules (BER), XML, stb.
- A kliens program egy teljesen normális lokális függvényt hív meg.
  - Ez egy automatikusan generált csonk (stub) valójában.
  - Feladata a kommunikáció részleteinek elrejtése a hívó fél elől.
  - A szerver megtalálásáról nem beszélünk most, tételezzük fel, hogy a hívó fél tudja.

RPC működése 2.

- A kliens oldali csonk feladata a hívás során.
  - Feladata a hívási paraméterek összecsomagolása platform független formára és elküldése a végrehajtó folyamatnak.
  - Ehhez felhasználja a lokális operációs rendszer és a hálózat szolgáltatásait.
- A szerver oldalon az RPC-t használó szolgáltatás megkapja a szabványos formátumban megérkező hívást.
  - Platform független formáról lokális formára hozás.
  - A lokális függvény hívása.
  - Visszatéréskor a visszatérési érték platform független formára hozása.
  - A visszatérési érték küldése a kliensnek.

RPC működése 3.

- A kliens oldali csonk feladata a hívás visszatérése során.
  - Feladata a szervertől megkapott visszatérési érték konvertálása platform független formáról lokális formára.
  - Visszatérés a lokális csonkból a lokális formátumra hozott visszatérési értékkel.
- A kliens szál az RPC hívás során várakozik.
  - Eseményre várakozik a hívás során.
- A szerver szál vár a beérkező hívásokra, és amikor van kiszolgálandó hívás, akkor fut.
  - Eseményre várakozik a hívás beérkezéséig.