Operációs rendszerek

Folyamatok

Folyamatok

- A folyamat (process) a multiprogramozott rendszerek alapfogalma.
- Folyamat általános definíciója: műveletek meghatározott sorrendben történő végrehajtása.
- A műveletek sorrendjét a vezérlési szál definiálja.

Folyamatok logikai modellje

- Végrehajtó gép:
  - processzor,
  - memória (folytonos).
- Oszthatatlan utasítások:
  - a fizikai processzor utasításainak egy része,
  - rendszer szolgáltatás (pl. I/O műveletek).
- Folyamat utasításai: program kód definiálja.
- Folyamat állapotának jellemzői:
  - processzor állapota,
  - memória tartalom (kód, változók, veremtár).

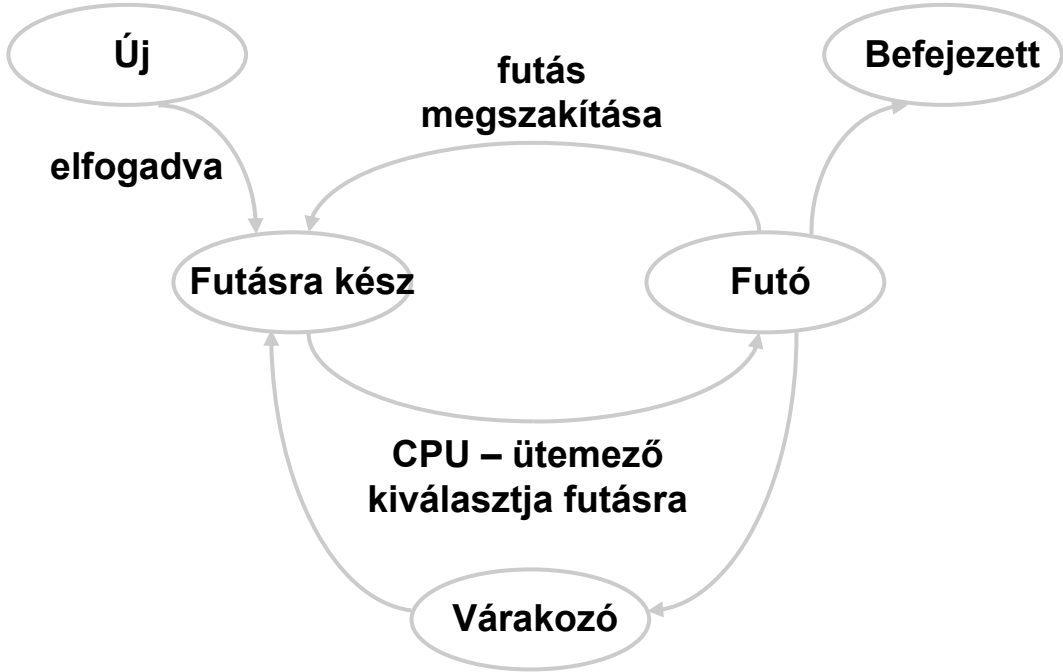
Folyamatok gyakorlati modellje I.

- Végrehajtás alatt álló "életre kelt" program (kód és adatterület együtt).
- Folyamat memóriaterületének részei:
  - a program kódja,
  - adatterületek,
  - veremtár:
    dinamikusan növelhető terület.

Folyamatok gyakorlati modellje II.

- Multiprogramozott rendszer:
  - több végrehajtás alatt álló folyamat,
  - egy fizikai processzor,
  - folyamatok közötti átkapcsolás, virtuális párhuzamos végrehajtás.
- Szükséges műveletek:
  - folyamat végrehajtásának leállítása,
  - folyamat újraindítása.

Folyamatok állapota



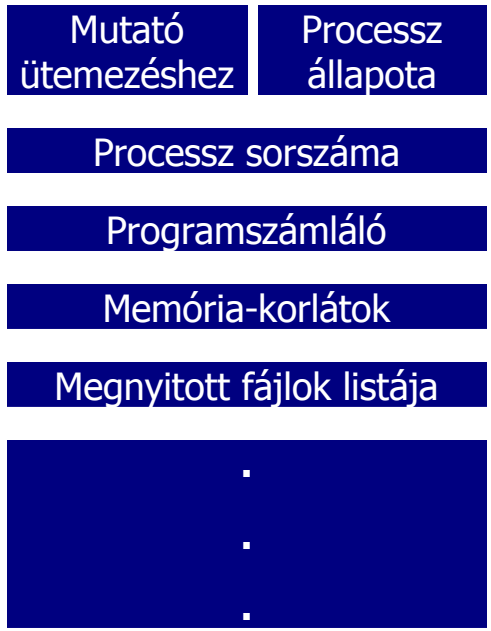
Folyamatok váltása

- Folyamat környezet (context) fogalma:
  - Program és a végrehajtó gép állapotának leírása.
  - Azon információ összessége, ami szükséges a folyamat későbbi újraindításához.
- Környezetváltás definíciója:
  - a CPU-t birtokló futó folyamat váltása.

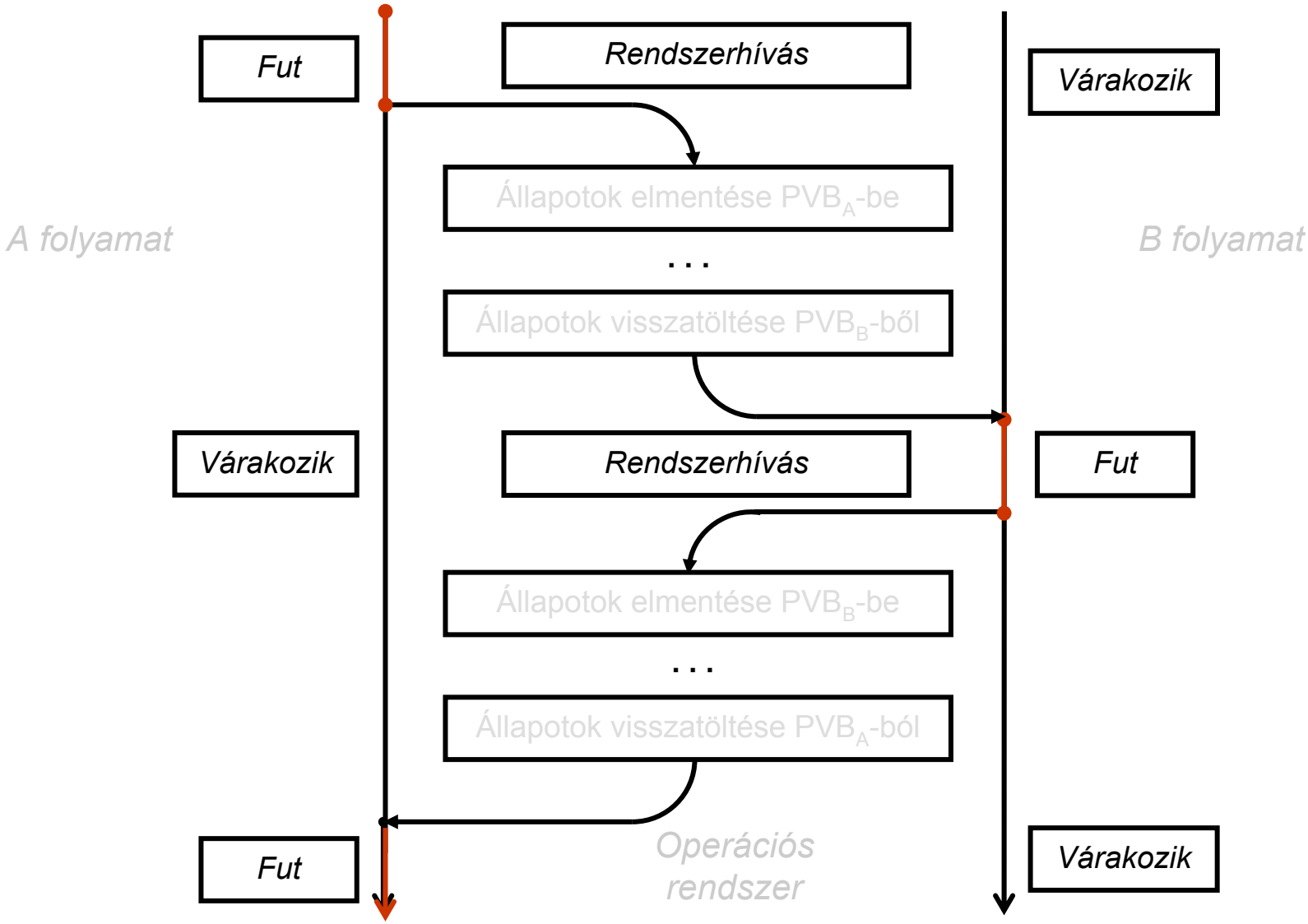
Folyamat környezet részei

- A processzor állapota:
  - program számláló,
  - CPU regiszterei.
- A folyamat leíró adatok (ami a futtatáshoz kell):
  - vezérlési információk (pl.: állapot, azonosító),
  - ütemezési információ,
  - folyamat jogosultságai (futási, állomány-elérési jogok a védelemhez),
  - folyamat által használt erőforrások,
  - a felhasználó által definiált környezeti változók.
- A memória területek tartalma.

Folyamat környezet leírása: Processz Vezérlési Blokk (Process Control Block)



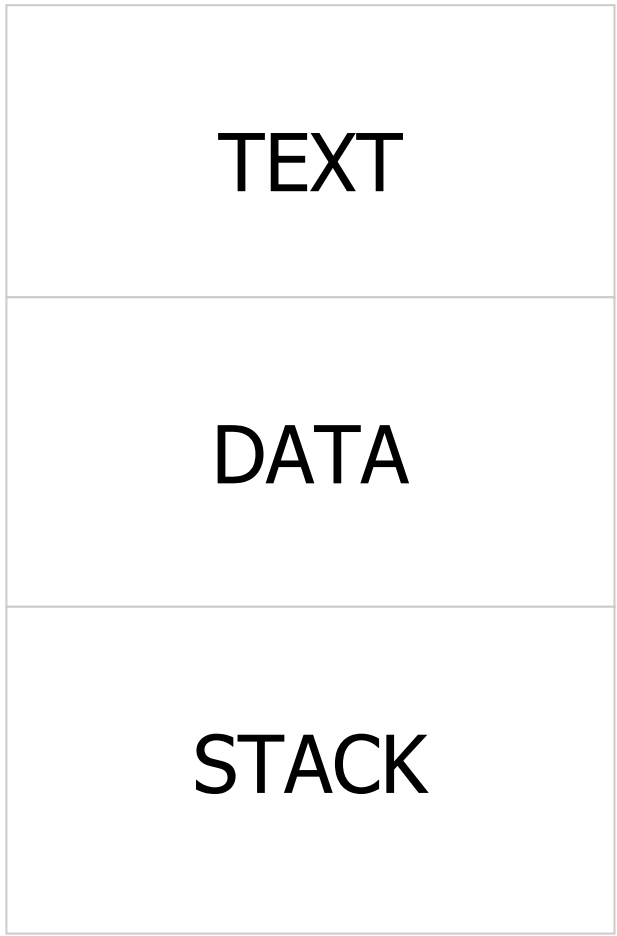
Folyamatok váltása



Folyamatok memóriakezelése

- Minden folyamat futása során úgy „tapasztalja”, mintha egyedül használná a számítógépet:
  - memóriát,
  - eszközöket (perifériákat).
- OR feladata:
  - összehangolni a folyamatok működését,
  - biztosítani a folyamatok adatainak sérthetetlenségét.

Folyamat memóriaterületének részei - logikai memóriakép



- program kód:
  - statikus,
  - dinamikus (DLL),
- adatterületek:
  - inicializált (DATA),
  - nem inicializált (BSS),
  - heap (dinamikus adatok),
- veremtár:
  - dinamikusan növelhető terület.

Szálak (thread, lightweight process)

- Együttműködő folyamatok.
- Környezetük részben közös:
  - memória (kód és adatok),
  - erőforrások.
- Egyedi környezet:
  - a program számláló,
  - CPU regiszterek tartalma,
  - veremtár.
- Előny: gyors környezetváltás.

Folyamatok, szálak használatának előnyei

- Hatékony erőforrás kihasználás:
  - CPU, perifériák.
- Párhuzamos végrehajtás lehetősége
  - feladat partícionálása,
  - Több CPU esetén gyorsítás.
- Több alkalmazás egyidejű végrehajtása.
- Áttekinthető rendszerstruktúra:
  - Egymáshoz viszonylag lazán kapcsolódó részekre osztása a feladatnak.

Párhuzamos folyamatok viszonya

- Független folyamatok:
  - Egymás futását nem befolyásolják.
  - Aszinkron futnak, egymáshoz viszonyított sebességükről nincs információ.
- Csatolt folyamatok:
  - Versengő folyamatok (többnyire user): logikailag független de közös erőforrásokat használó folyamatok.
  - Együttműködő folyamatok (többnyire rendszer): közös cél végrehajtásán munkálkodó folyamatok.

Folyamatok együttműködése

Alapfogalmak

- Szinkronizáció:
  - folyamatok futásának időbeli összehangolása.
- Kommunikáció:
  - információcsere a folyamatok között.
- Holtpont.
- Éheztetés.

Holtpont - éheztetés

A szinkronizáció problémájához kapcsolódó fogalmak:

- Holtpont - a rendszer egészére vonatkozó fogalom:
  - Holtpont akkor áll elő, ha több folyamat egy olyan eseményre várakozik, amelyet csak egy szintén várakozó folyamat tudna előidézni.
  - Végtelen várakozás.
- Éheztetés -ez egyes folyamatokra vonatkozik:
  - Gyakorlatilag végtelen várakozás – egy folyamat meghatározatlan ideig kényszerül várakozni, hogy erőforráshoz jusson.
  - Nem fair ütemezés - következtében valamelyik folyamat véges időn belül nem jut egy adott erőforráshoz.

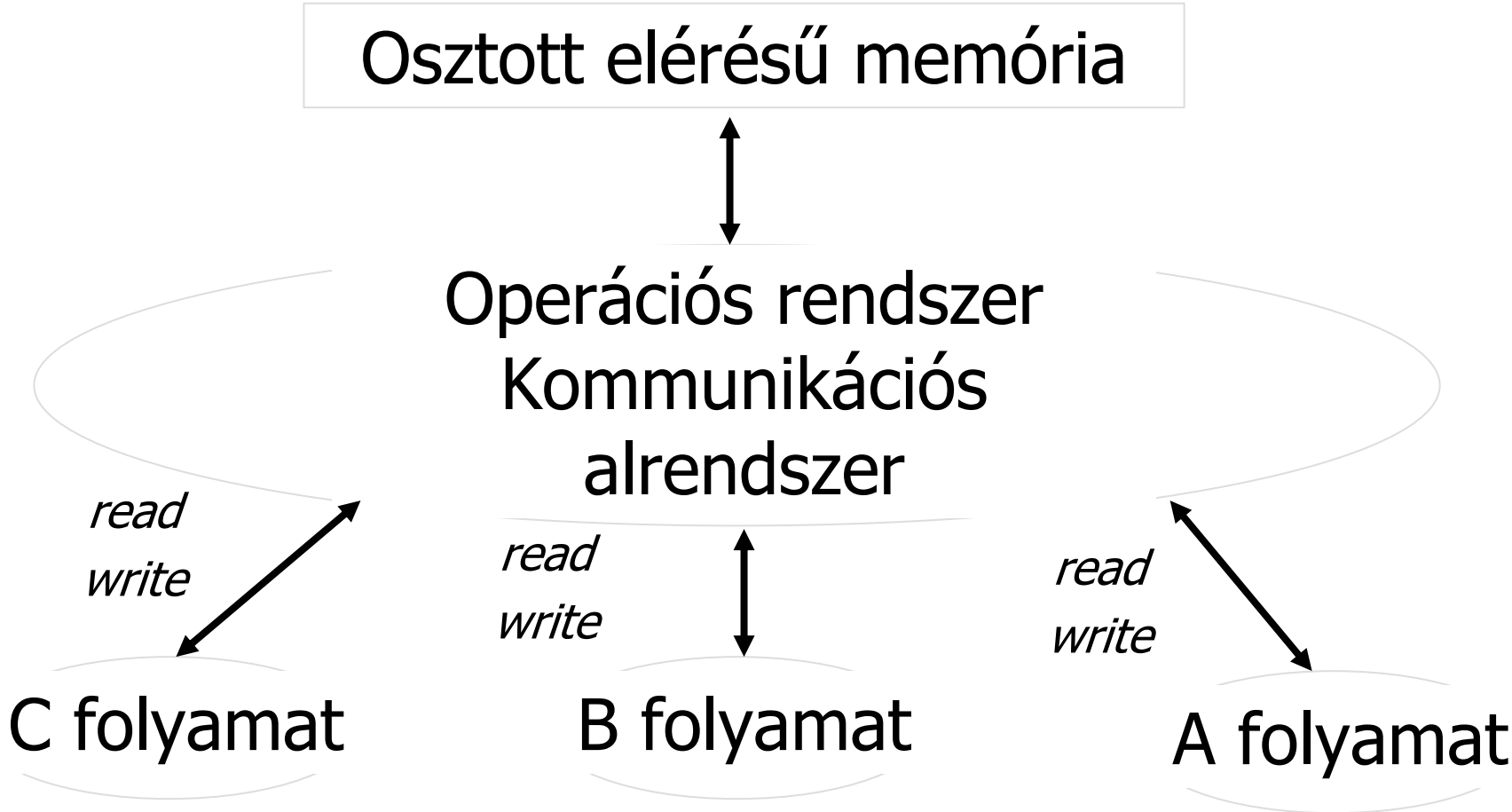
Folyamatok közötti információ-csere (kommunikáció)

Információcsere formái:

- Közös tárterületen keresztül.
- Kommunikációs csatornán keresztül.
- Mechanizmus:
  - küldő SEND(üzenet),
  - vevő RECEIVE(üzenet).

Közös tárterületen történő kommunikáció

Közös tárterületen történő kommunikáció



A RAM modell

- A memória a RAM modell szerint működik:
  - tárolórekeszekből áll,
  - rekeszenként címezhető,
  - csak rekeszenként, írás és olvasás műveletekkel érhető el,
  - írás, felülírja a teljes rekesztartalmat,
  - az olvasás nem változtatja meg a rekesz tartalmát.

Közös tárterületen történő kommunikáció

- PRAM (Pipelined/Parallel Random Access Memory) modell.
- Oszthatatlan atomi műveletek:
  - olvas,
  - ír.
- Probléma:
  - műveletek ütközése.

A PRAM modell

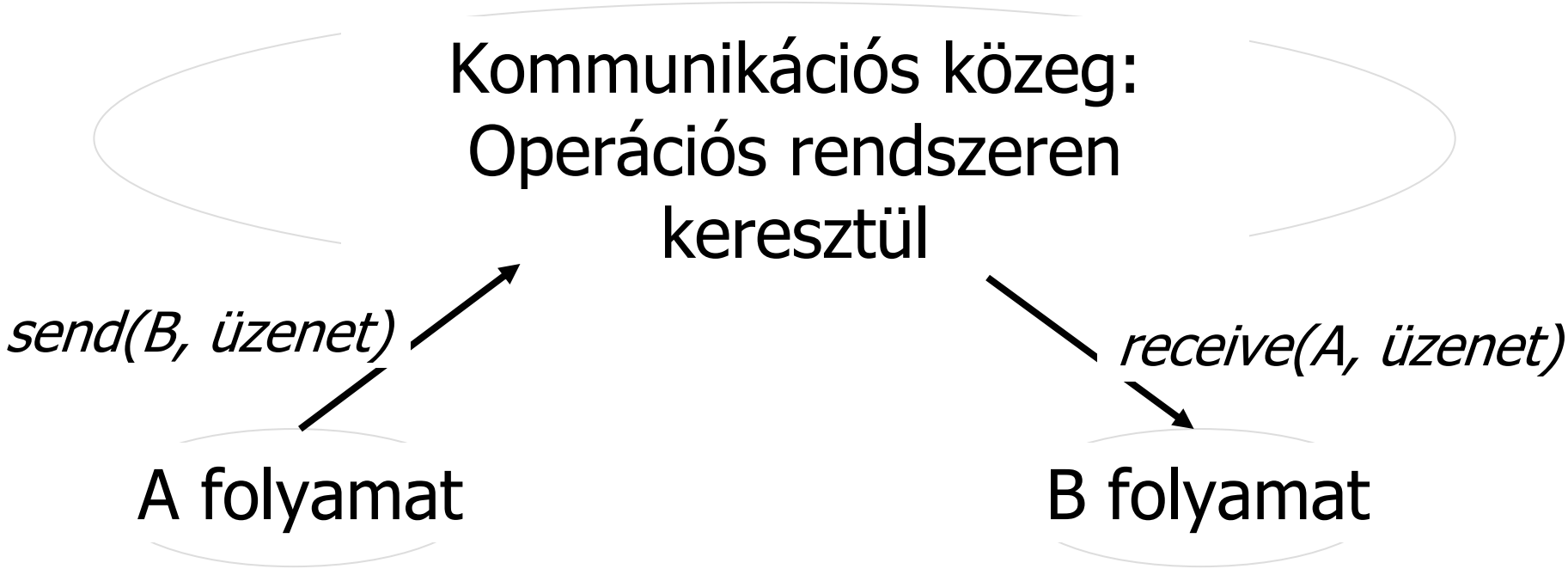
- olvasás-olvasás ütközésekor mindkettő ugyanazt az adatot kapja,
- olvasás-írás ütközésekor a régi adat felülíródik, az olvasó a régi vagy az új adatot kapja,
- írás-írás ütközésekor valamelyik írja felül az aktuális adatot.

Műveletek ütközésének kezelése

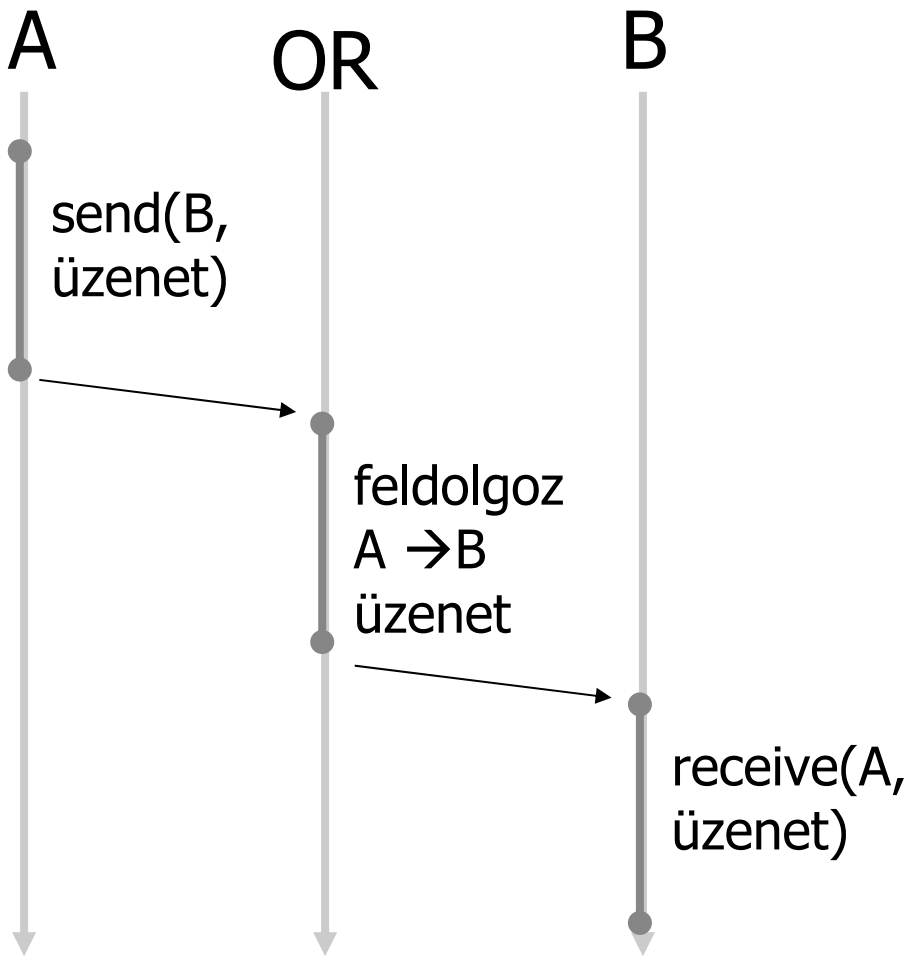
- Nem lapolódhat át időben a műveletek végrehajtása.
- Ütköző műveletek sorrendjét meg kell határozni.
- Valós használat:
  - szinkronizáció,
  - kölcsönös kizárás biztosítása.

Üzenetváltás kommunikációs csatornán keresztül

Üzenetváltás kommunikációs csatornán keresztül



Kommunikáció általános sémája



Üzenetváltás kommunikációs csatornán keresztül

Séma:

- kommunikációs csatorna (közeg),
- műveletek:
  SEND(címzett folyamat, üzenet),
  RECEIVE(küldő folyamat, üzenet),
- "üzenet" küldésnél:
  - a saját memória címen tárolt adat tömb,
- "üzenet" vételnél:
  - a saját memória címre letöltendő adat tömb.

Kommunikációs modell

- Nincs egységes modell.
- A műveletek hatása, paraméterei az alkalmazott megoldástól függ.

Rendszerezési szempontok (tartalom)

- Kommunikáció típusai:
  - a partner megnevezés módja (1.),
  - a műveletek hatása (szemantikája) (2-3.).
- Csatorna jellemzői (2-3.).

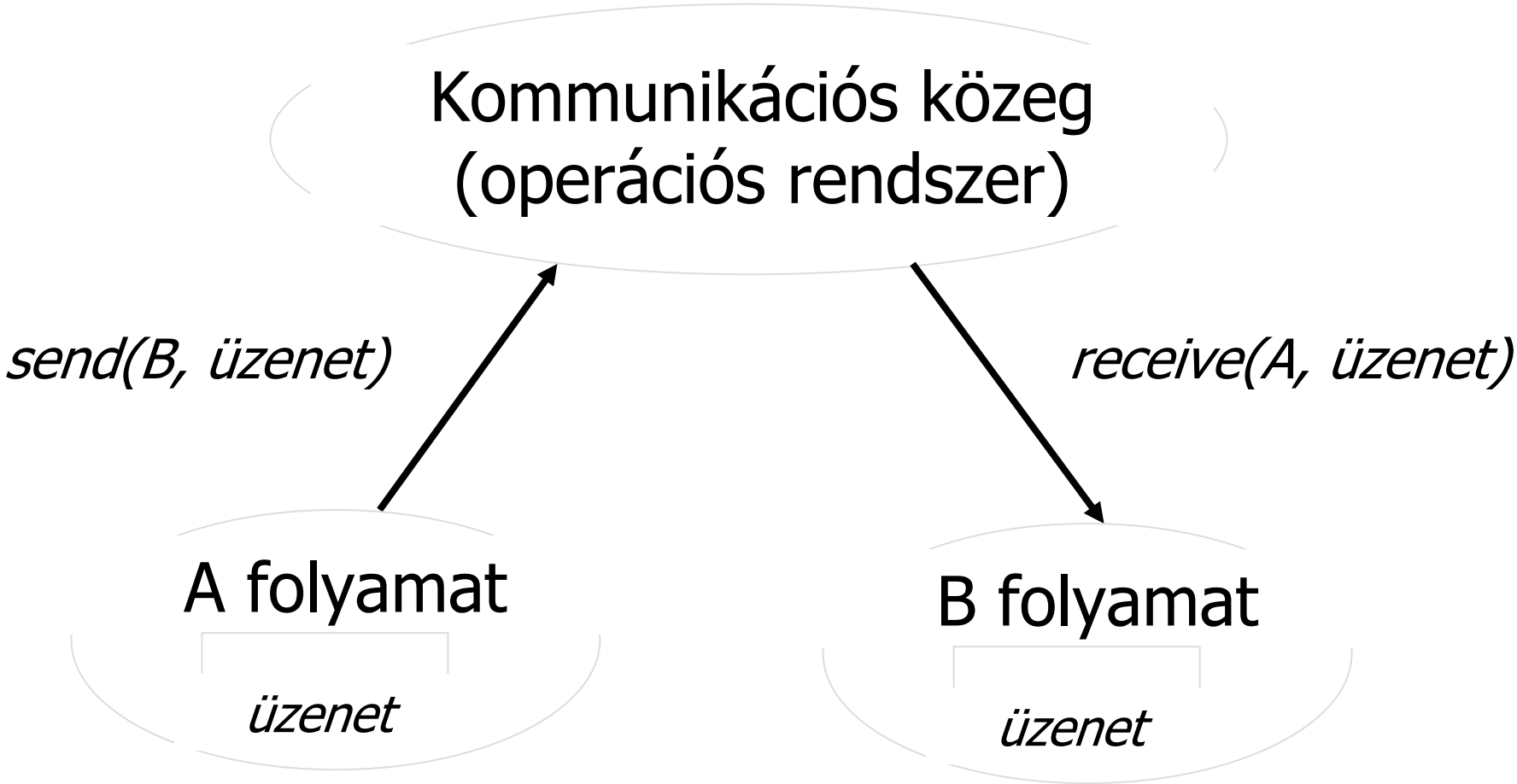
Kommunikáció típusai

- Partner megnevezés alapján:
  - a partner megnevezésére vonatkozó kérdések.
- Műveletek szemantikája (jelentése) alapján
  - mi történik a kommunikációs műveletek végrehajtásakor,
  - milyen hatása van a műveleteknek az egyes folyamatok állapotterére.
    (Üzenetküldési műveletek a folyamat működésére gyakorolt hatása alapján.)

Kommunikáló partner megnevezése alapján

- közvetlen (direkt) kommunikáció,
- közvetett (indirekt) kommunikáció,
- csoport kommunikáció.

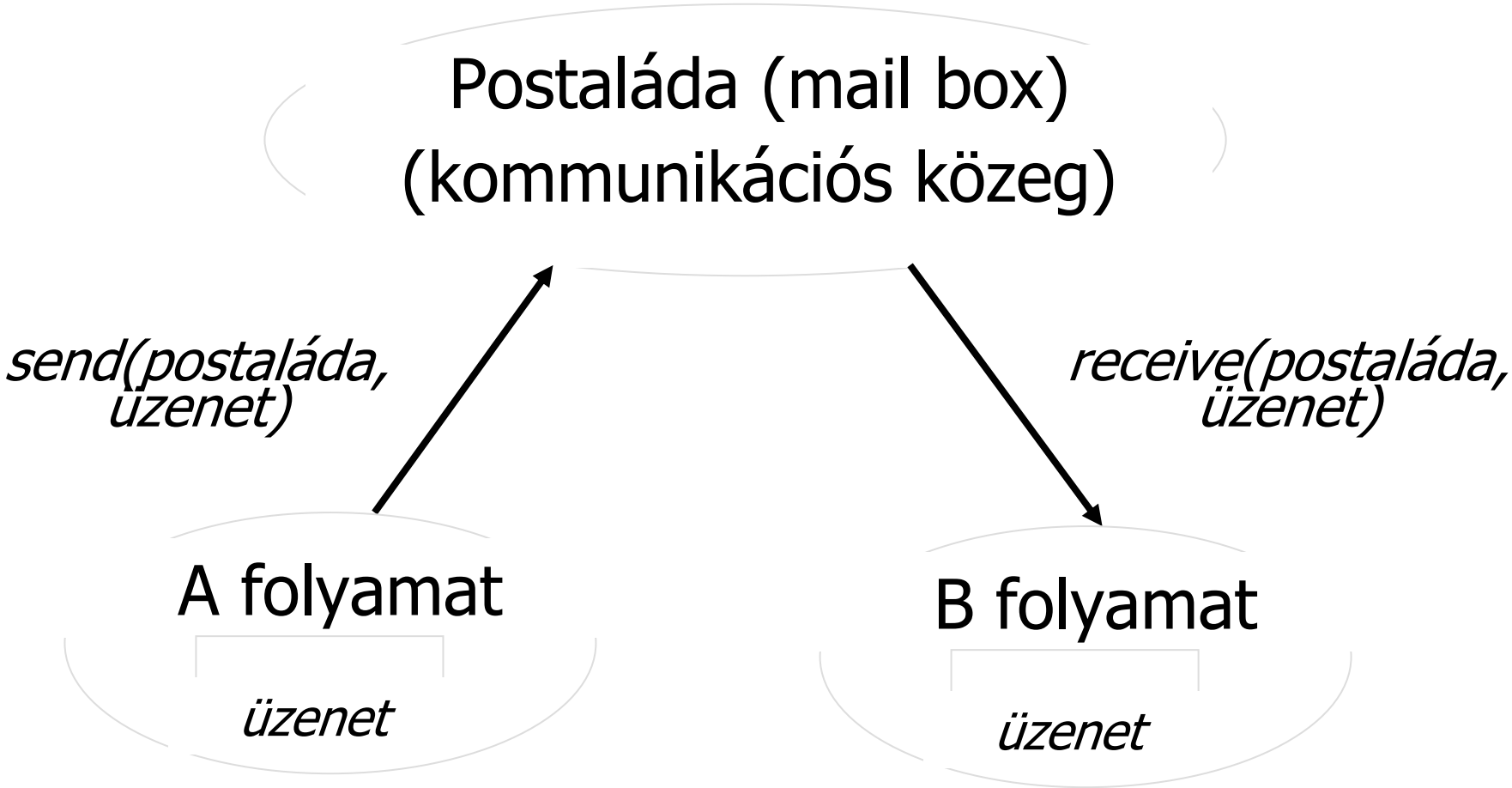
Közvetlen kommunikáció



Közvetlen kommunikáció

- Két folyamat között zajlik, mind a Küld és a Fogad művelet megnevezi a partner folyamatot.
- A két folyamat között pontosan egy kommunikációs csatorna létezik, s ezen keresztül más folyamatok nem kommunikálhatnak egymással.
- Általában szinkronizáció a folyamatok között.
- Műveletek szintaktikája:
  - SEND(címzett folyamat, üzenet),
  - RECEIVE(küldő folyamat, üzenet).

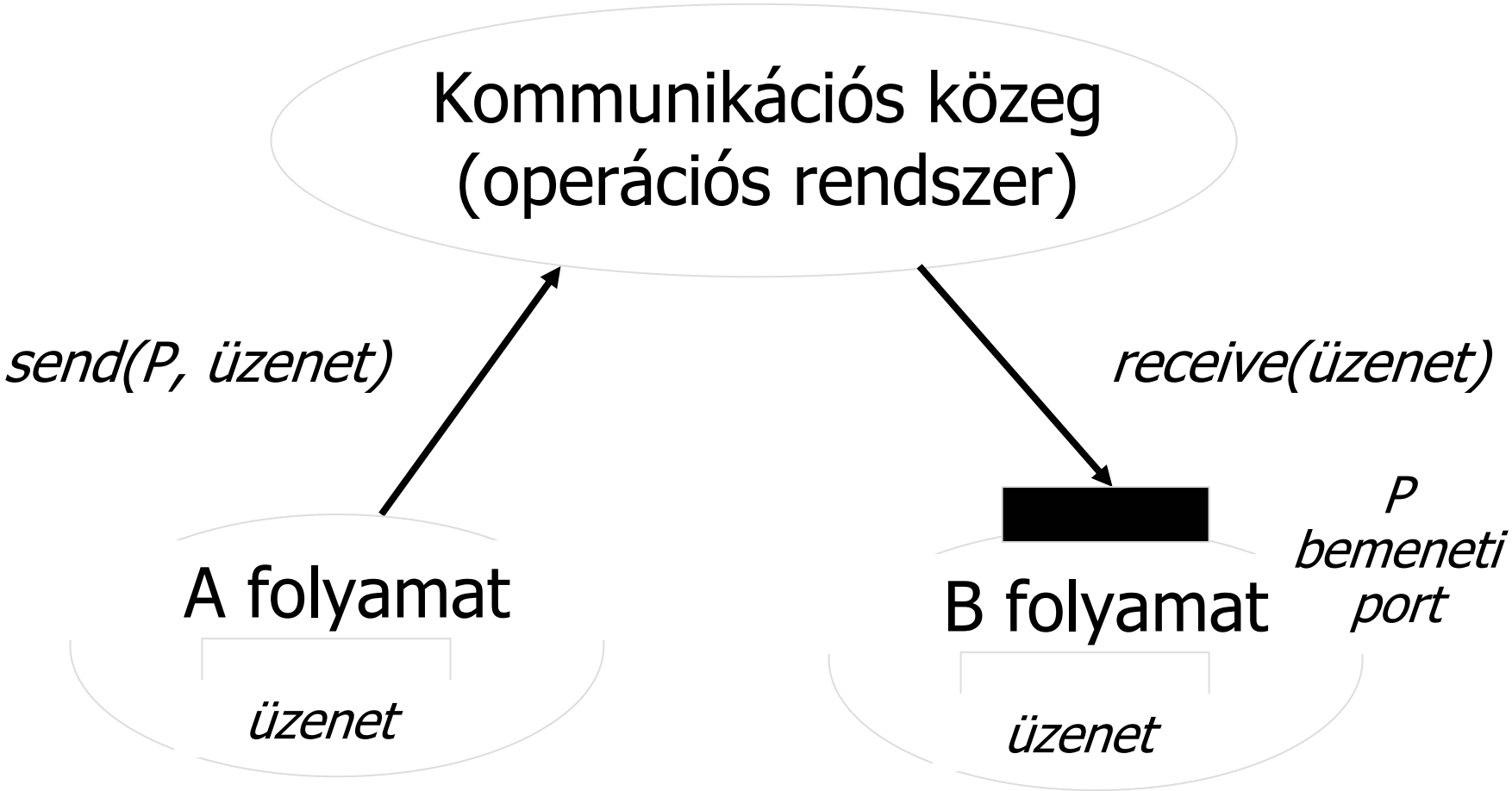
Közvetett kommunikáció



Közvetett kommunikáció

- A partnerek nem egymást, hanem egy közvetítő objektumot (postaláda, csatorna) neveznek meg.
- ebben az esetben két folyamat több postaládát is használhat, és egy postaládát több folyamat is felhasználhat.
- Postaláda (FIFO tároló) használata.
- Nem szükséges szinkronizáció.
- Műveletek szintaktikája:
  - SEND(postaláda, üzenet),
  - RECEIVE(postaláda, üzenet).
- Kérdések:
  - postaláda kapacitása,
  - használó folyamatok száma.

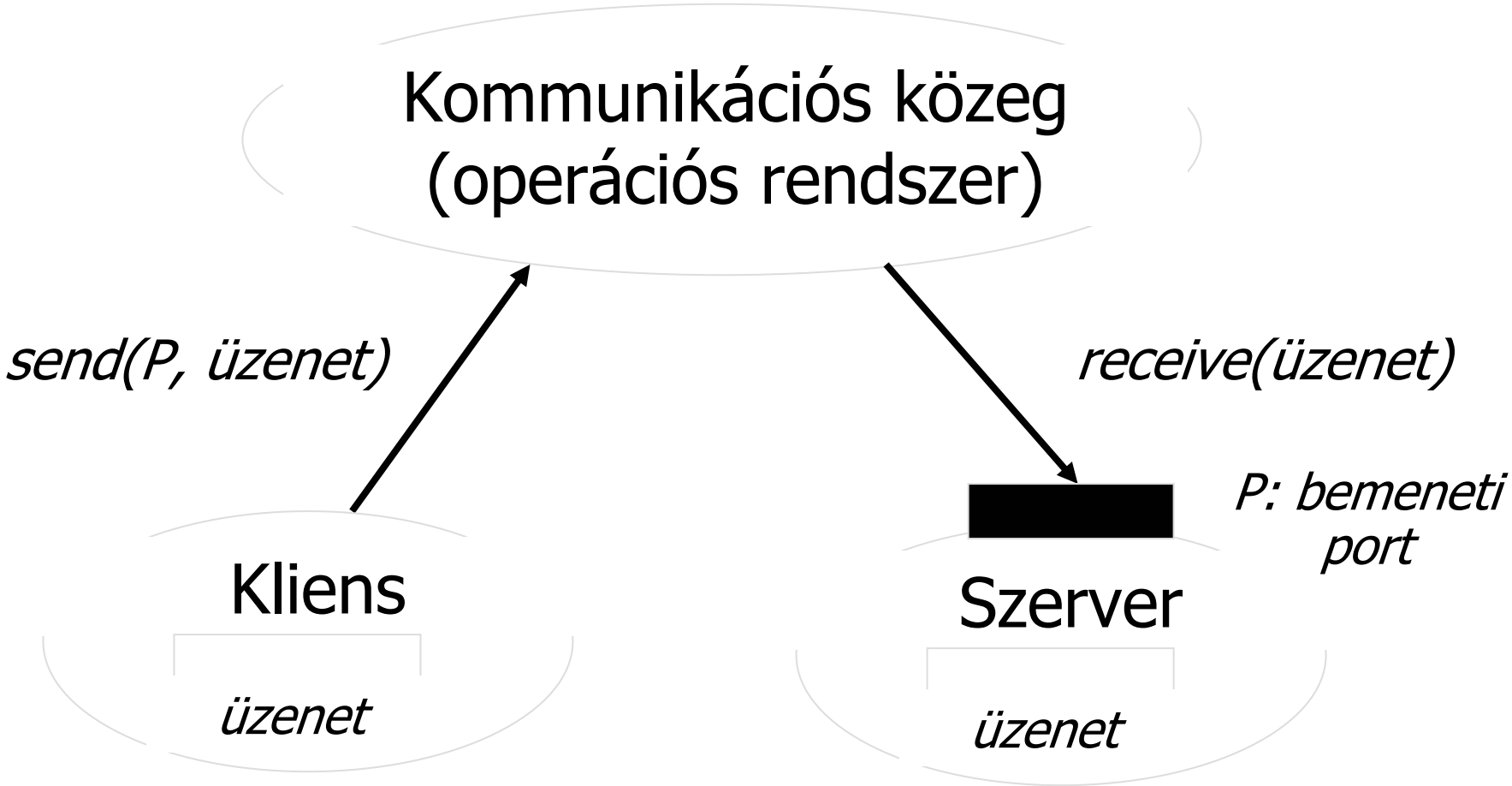
Aszimmetrikus kommunikáció



Aszimmetrikus kommunikáció

- Az egyik folyamat, az adó vagy a vevő megnevezi, hogy melyik folyamattal akar kommunikálni, a másik viszont egy saját be/kimeneti kaput használ.
- Aszimmetrikus megnevezés esetén az a fél, amelyik a kaput használja, információt kap arról, hogy a kommunikáció másik végén melyik folyamat áll.
- Csatorna használata:
  - FIFO összeköttetés két folyamat között.
- A kommunikáló folyamat nem kell, hogy ismerje a partner azonosítóját.

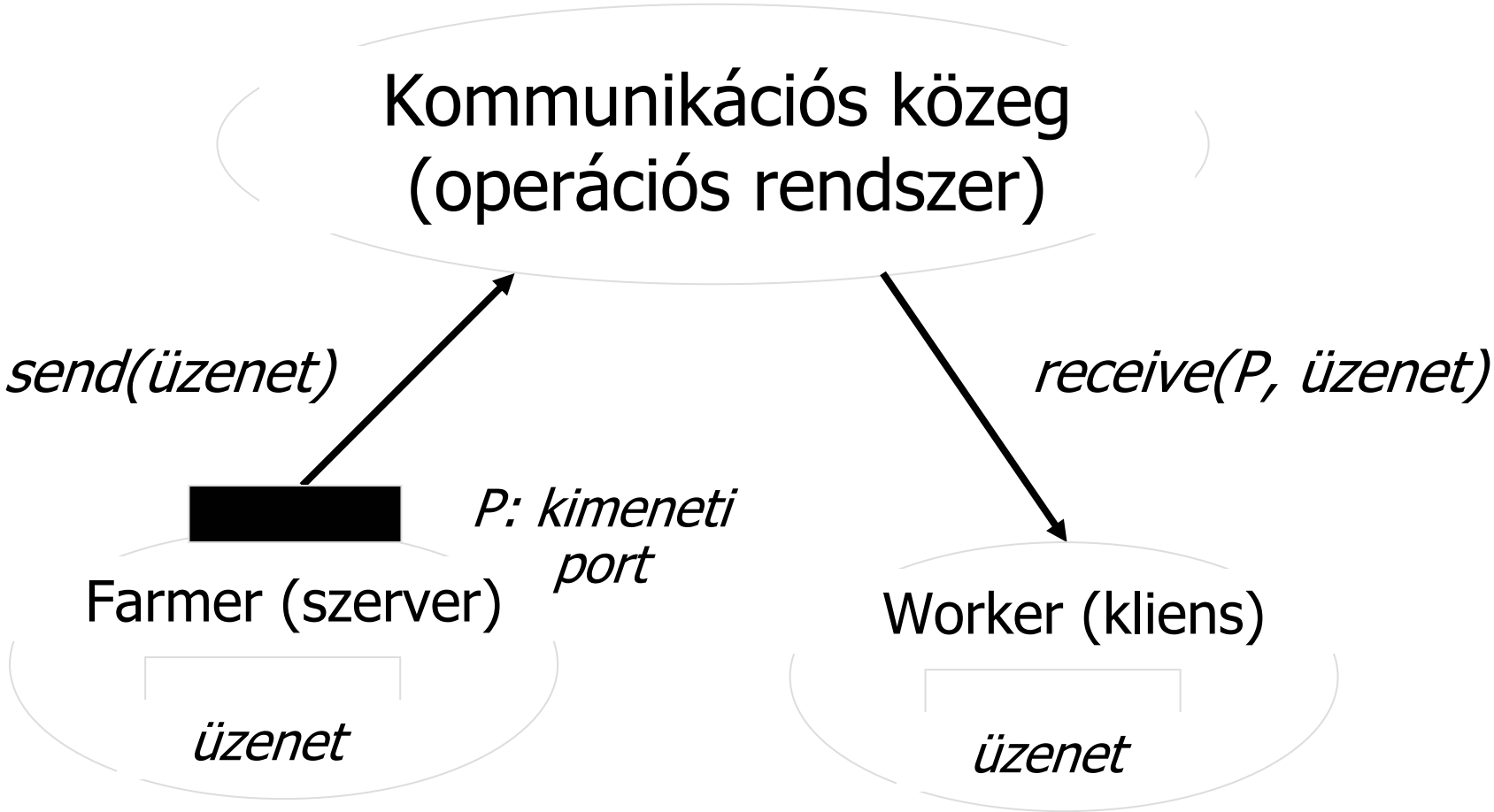
Vevő oldali bemeneti port



Vevő (szerver) oldali bemeneti port használata

- Vevő nem ismeri a küldőt.
- Pl. kliens-szerver modell:
  - szerver szolgáltató várja a kliens kéréseket.
- Műveletei:
  - SEND(címzett folyamat bemeneti portja, üzenet),
  - RECEIVE(üzenet) (implicit a port-ra).

Adó oldali kimeneti port



Adó (szerver) oldali kimeneti port

- Adó nem ismeri a címzettet.
- Pl. farmer – worker modell:
  - a munkát szétosztó manager folyamat várja a munkáért versengő feldolgozó folyamatokat.
- Műveletek:
  - SEND (üzenet),
  - RECEIVE (adó folyamat kimeneti portja, üzenet).

Csoport kommunikáció

- A küldő, a folyamatok egy csoportját nevezheti meg vevőként.
- A kijelölt folyamat-csoport minden tagja megkapja az üzenetet.
- Üzenetszórás, ún. broadcasting üzenet:
  - minden folyamat megkapja az üzenetet.
- Művelet:
  - SEND(csoport, üzenet).

Csatorna jellemzői - Üzenetküldő műveletek szemantikája

Csatorna jellemzői I.

- Átviteli közeg típusa:
  - pl.: mail box, hálózati összeköttetés.
- Kapcsolat iránya:
  - Szimplex (egyirányú) csatorna,
  - Fél duplex (osztottan kétirányú) csatorna,
  - Duplex (kétirányú) csatorna.
- Csatornát használó folyamatok száma:
  - kettő vagy több.

Csatorna jellemzői II.

Üzenetek tárolása a csatornán:

- automatikus pufferelés.

Kérdések:

- csatorna kapacitása,
- csatorna megbízhatósága.

Csatorna kapacitása

- Üzenetváltáskor implicit szinkronizáció: a folyamatok közötti, üzenettovábbításos információcsere szinkronizációs mellékhatással jár.
- Az egyes műveletekkel járó szinkronizációs mellékhatás a kommunikációs rendszer átmeneti tárolójának a kapacitásától függ.
- Csatorna tartalmazhat:
  - 0 kapacitású átmeneti tárolót – tárolás nélküli átvitel.
  - Véges tárolás.
  - Végtelen tárolás.

Tárolás nélküli átvitel

- Nincs tárolás (közvetlen kommunikáció).
- A kommunikációs rendszer csak közvetít.
- A Küld és Fogad műveleteknek be kell egymást várni az információcsere érdekében.
- A két folyamat ezen utasításaira az egyidejűség érvényes - szinkronizáció (randevú) szükséges.

Véges kapacitású tároló

- Várakoztatás szükséges:
  - a fogadó folyamat várakozik, ha az átmeneti tároló üres,
  - a küldő folyamat akkor várakozik, ha nincs szabad hely a tárolóban.
- Az adó és a vevő egyaránt hozzáférhet egy véges tárkapactitású puffer tartalmához. A pufferbe való beírás és kiolvasás ekkor nem lehet egyidejű.
- Ugyanazon üzenet elküldésének meg kell előznie az üzenet fogadását – ezen üzenetekre a sorrendiség érvényesül.
- Rövid távú szinkronizációs hatás érvényesül: a tároló kezelése kölcsönös kizárással valósul meg.
  - a vevőnek várnia kell, amíg az adó befejezi egy üzenet küldését, ill. az adónak várnia kell, amíg egy üzenet kiolvasása nem fejeződött be.
- Két folyamat nem hajthat végre egyidejűleg kommunikációs műveletet ugyanarra a kommunikációs objektumra.

Végtelen kapacitású tároló

- Virtuális lehetőség – modell szinten.
- Nincs várakozás – a küldő folyamatnak sosem kell várakoznia.
- Erre akkor van szükség, ha az OPR nem képes szabályozni az adó működését (pl. egy OPR-től független külső folyamat). Ekkor az adót nem lehet várakoztatni, s elvileg végtelen kapacitású puffert kell a rendelkezésére bocsátani.
- Ekkor tipikus kommunikációs hibák fordulhatnak elő:
  - túlcsordulás: a puffer tényleges betelése után küldött információ egyszerűen elvész,
  - már beírt információ sérülése: a pufferben (esetleg részlegesen) felülíródik egy még abból ki nem olvasott (esetleg részlegesen kiolvasott) információ.

Csatorna megbízhatósága I.

- Távoli kommunikáció:
  - megbízhatatlan kommunikáció.

Kezelendő hiba események:

- adó meghal,
- vevő meghal.

Felismerés lehetséges módja:

- Időkorlát (timeout) használata.
- SEND(címzett folyamat, üzenet, időkorlát),
- RECEIVE(küldő folyamat, üzenet, időkorlát).

Csatorna megbízhatósága II.

- Üzenetvesztés, sérülés esetén:
  - OPR jelzi és kezeli.
  - OPR jelzi, felhasználó kezeli.
  - Felhasználó veszi észre és kezeli.
- Hibakezelés technikája:
  - Hibakód visszatérési érték.
  - SEND(címzett folyamat, üzenet, hibakód),
  - RECEIVE(küldő folyamat, üzenet, hibakód).

Üzenetek sérülése

- Üzenetek sérülésének megakadályozása kódolással:
  - Hiba detektálása.
  - Hiba javítása.
- Nem minden üzenet érvényes.
- Érvénytelen (hibás) üzenet észlelése-javítása.

Üzenetváltás műveleteinek általános formája

- Műveletek általános paraméterei:
  - SEND(címzett folyamat, üzenet, időkorlát, hibakód).
  - RECEIVE(küldő folyamat, üzenet, időkorlát, hibakód).

Üzenetváltás módja

- Üzenetváltás módja:
  - aszinkron: a küldő a hívás után azonnal folytatja a működését,
  - szinkron: ha a küldő végrehajtott egy send utasítást, de a fogadó addigra még nem hajtott végre egy receive utasítást, akkor a küldő blokkolódik, amíg a fogadó egy receive utasítást végrehajt, és megtörténik az üzenet küldése.
    - időkorlát használata:
      - 0: 0 várakozás hasznos, ha egy fogadó folyamat működésének egy pontján csak akkor akar üzenetet fogadni, ha azt már elküldték, egyébként van más teendője.
      - végtelen: hasznos olyan folyamat esetén, amelyiknek nincs teendője, amíg valaki egy üzenettel el nem indítja.

Szinkronizáció

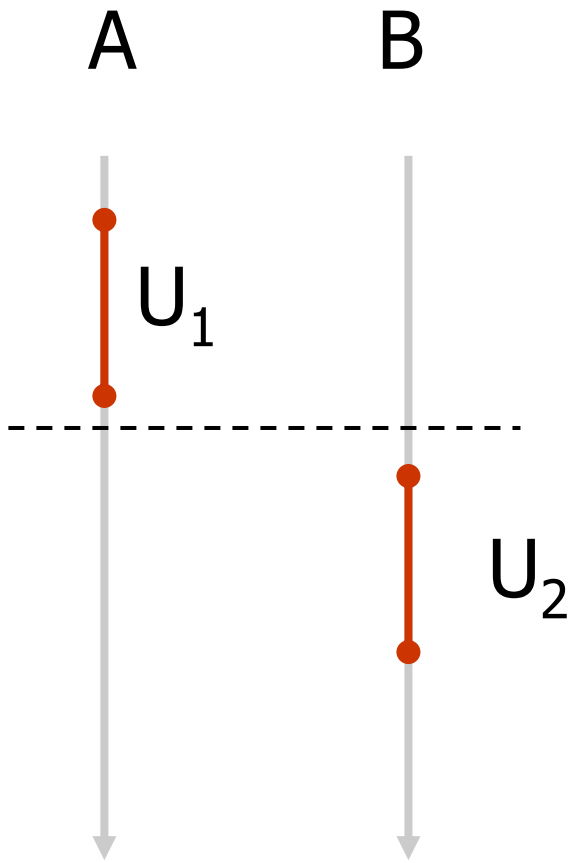
Folyamatok futásának időbeli összehangolása

A szinkronizáció formái

Egy adott folyamat végrehajtásának olyan időbeli korlátozása, amelyet egy másik folyamat futása, vagy esetleg valami külső esemény bekövetkezése határoz meg:

- előidejűség (precedencia),
- egyidejűség (randevú),
  - meghosszabbított (extended) randevú,
- kölcsönös kizárás (mutual exclusion).

Előidejűség (precedencia)



U_1 előidejű U_2-höz képest,

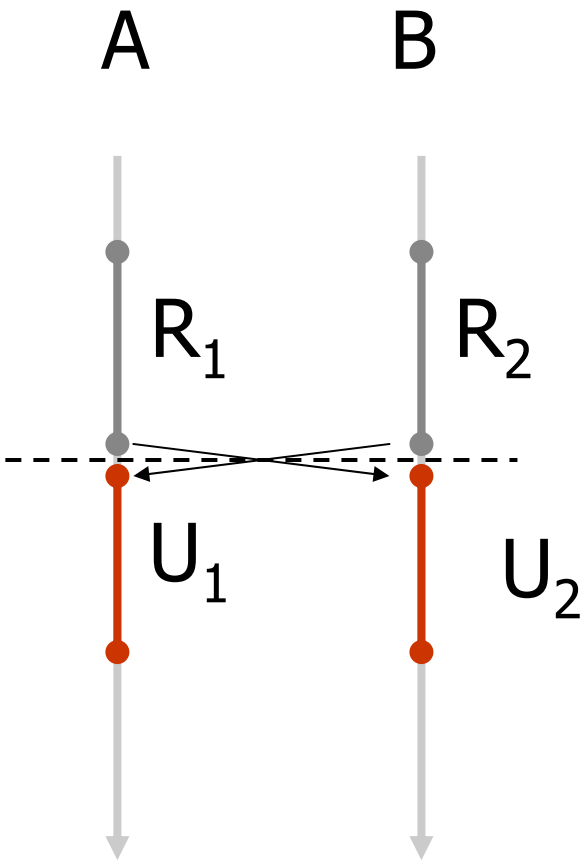
azaz U_1 vége megelőzi

U_2 megkezdését.

Előidejűség (precedencia)

- A különböző folyamatok kijelölt műveletei között végrehajtási sorrendet írunk elő.
- Például az A és B folyamatok esetén:
  - U_1 az A folyamat egy utasítása,
  - U_2 a B folyamat egy utasítása,
  - $U_1 => U_2$ precedencia előírása azt jelenti, hogy az U_1 végrehajtásának be kell fejeződni, mielőtt az U_2 végrehajtása megkezdődne, ha nem, B-nek várnia kell.

Egyidejűség (randevú)



U_1 egyidejű U_2-vel, azaz

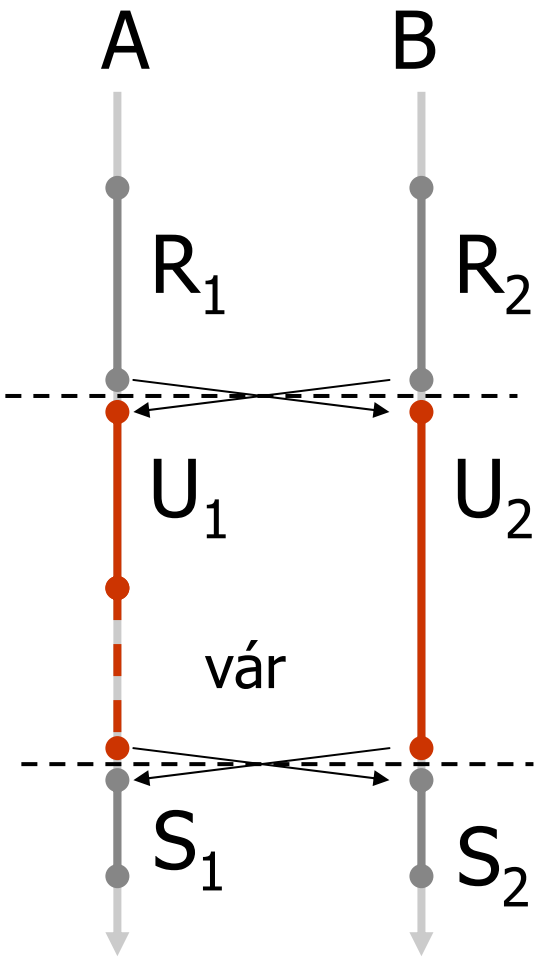
R_1 vége megelőzi U_2 megkezdését és

R_2 vége megelőzi U_1 megkezdését.

Egyidejűség (randevú)

- A különböző folyamatokban levő műveletekre előírható, hogy azok várják be egymást, és „egyidejűleg” hajtódjanak végre.
- Pontosabban, egyik folyamat sem léphet túl a benne, egyidejű végrehajtásra kijelölt utasítássorozaton mindaddig, amíg az összes többi folyamat nem kezdte meg a saját kijelölt utasítássorozatának végrehajtását.

Meghosszabbított randevú



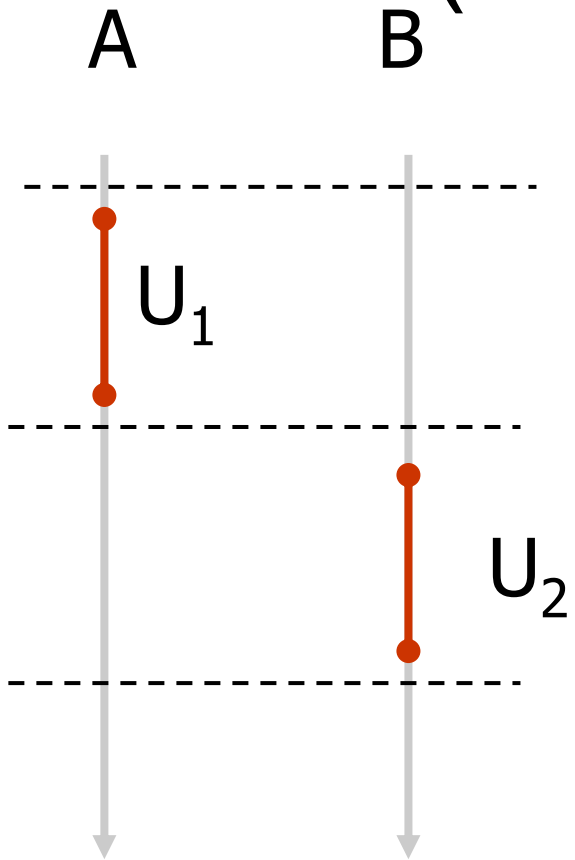
U_1 egyidejű U_2-vel, azaz

R_1 vége megelőzi U_2 megkezdését és

R_2 vége megelőzi U_1 megkezdését.

Továbbá U_1 megvárja U_2 végét.

Kölcsönös kizárás (mutual exclusion)



Az U_1 -re és az U_2 -re igaz a kölcsönös kizárás, azaz U_1 és U_2 egyidejűleg nem hajtódik végre.

Kölcsönös kizárás (mutual exclusion)

- A folyamatokban lehetnek olyan utasítássorozatok (kritikus szakasz), amelyek végrehajtása egyidejűleg nem megengedett.
- Egy folyamat akkor léphet be a kritikus szakaszba, vagyis akkor kezdheti meg a kritikus szakaszként kijelölt utasítássorozat végrehajtását, ha azon belül egyetlen másik folyamat sem tartózkodik.
- Ha ez a feltétel nem áll fenn, akkor a folyamatnak várni kell, amíg a kritikus szakaszban levő folyamat elhagyja azt.

Kölcsönös kizárás (mutual exclusion)

Szoftvermegoldások

- A szinkronizáció igénye, a multiprogramozott OPR-ren belül a kölcsönös kizárás problémájának formájában vetődött fel.
- A multiprogramozott rendszer folyamatainak kézenfekvő együttműködési módja a közös memória használata.
- Így a probléma megoldását először, a PRAM modell szerinti közös memóriát használó folyamatokra keresték.
- A probléma megoldására a folyamatok egyezményes adatszerkezeteket használnak és a kritikus szakaszokban egyezményes műveletsorozatot hajtanak végre – szoftvermegoldások.
- A mai megoldások hardvertámogatásra épülnek!!!

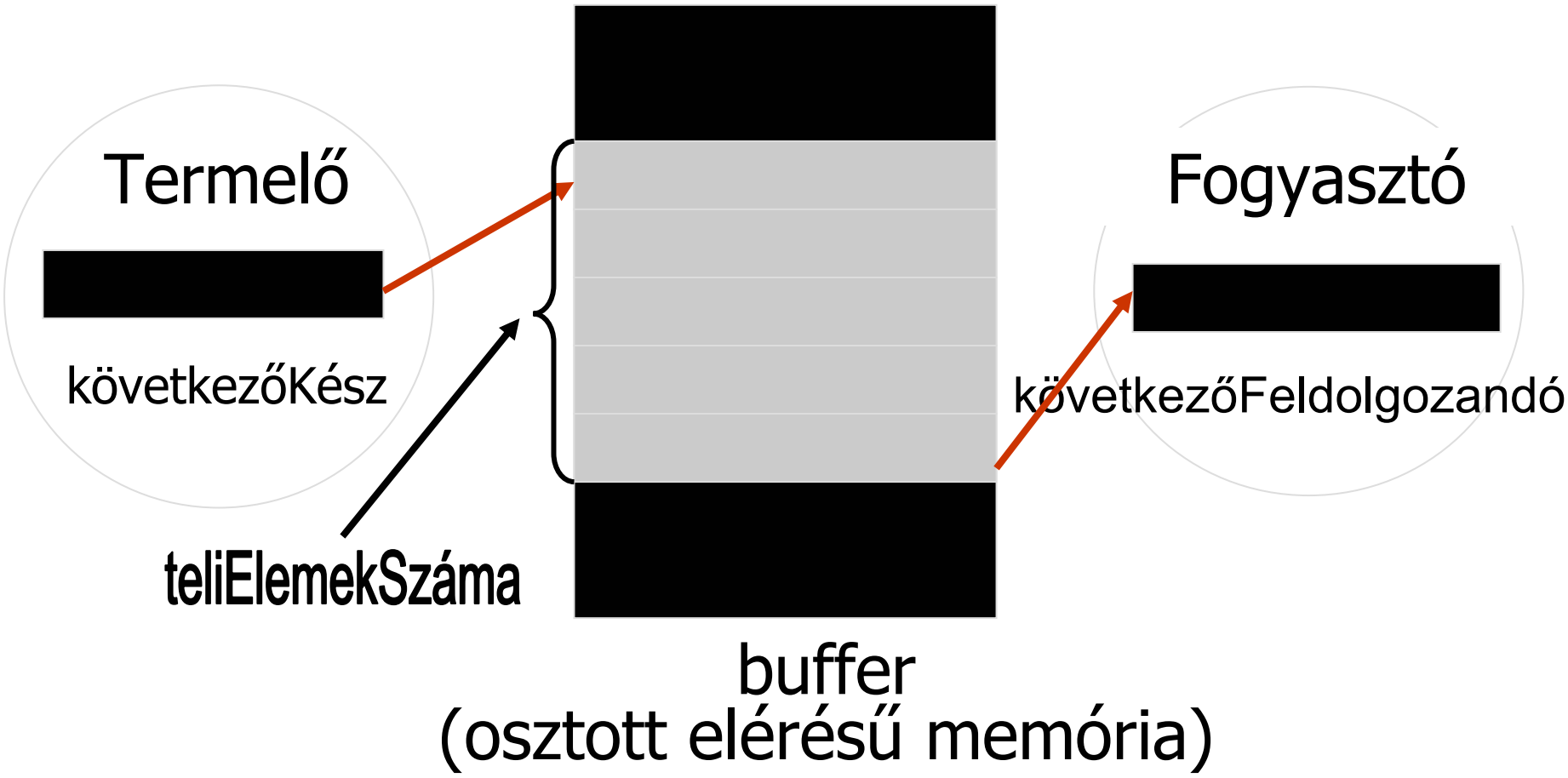
Kölcsönös kizárás (mutual exclusion)

- A leggyakrabban használt szinkronizációs eszköz.
- Erőforrások (hardver vagy szoftver) használata:
  - Egyidejűleg csak egy folyamat használhatja.
  - Párhuzamos használat hibás működést eredményezne.
- Pl.: file-ok, portok, printer.
- Programrészletek.

Kölcsönös kizárás - gyakorlati példa

- Termelő-fogyasztó (felhasználó) modell:
  - Két folyamat osztozik egy közös, rögzített méretű tárolón.
  - Nyomtatás vezérlése.
- Termelő:
  - Adatcsomagokat készít elő nyomtatásra.
- Fogyasztó:
  - Adatcsomagokat vár és kinyomtatja őket.

Termelő-fogyasztó modell



Termelő

repeat

[Készít egy új kinyomtatandó adatot a **következőKész** változóban.]

while teliElemekSzáma >= N **do** üresUtasítás;

buffer[következőÜres]:= következőKész;

következőÜres:= (következőÜres + 1) mod N;

teliElemekSzáma := teliElemekSzáma + 1;

until false;

(N: a tárolóban az elemek maximális száma.)

Fogyasztó

repeat

while teliElemekSzáma <= 0 do üresUtasítás;

következőFeldolgozandó := buffer[következőTeli];

következőTeli := (következőTeli + 1) mod N;

teliElemekSzáma := teliElemekSzáma - 1;

[Kinyomtatja a **következőFeldolgozandó** változóban tárolt adatot.]

until false;

Probléma

- A teliElemekSzáma változó inkrementálásának, ill. dekrementálásának gépi utasításai párhuzamosan (átlapolódva) hajtódhatnak végre.
- Eredmény a végrehajtás sorrendjétől függ!

Inkrementálás-dekrementálás

- Inkrementálás

```
MOV AX, <teliElemekSzáma>
INC AX
MOV <teliElemekSzáma>, AX
```

- Dekrementálás

```
MOV AX, <teliElemekSzáma>
DEC AX
MOV <teliElemekSzáma>, AX
```

Végrehajtás 1

```
MOV AX(4), teliElemekSzáma(4)
INC AX(5)
MOV teliElemekSzáma(5), AX(5)
```

```
MOV AX(5), teliElemekSzáma(5)
DEC AX(4)
MOV teliElemekSzáma(4), AX(4)
```

Eredmény:

`teliElemekSzáma=4`

Végrehajtás 2

```
MOV AX(4), teliElemekSzáma(4)
INC AX(5)
```

```
MOV AX(4), teliElemekSzáma(4)
DEC AX(3)
MOV teliElemekSzáma(3), AX(3)
```

```
MOV teliElemekSzáma(5), AX(5)
```

Eredmény:

`teliElemekSzáma=5`

Végrehajtás 3

```
MOV AX(4), teliElemekSzáma(4)
INC AX(5)
```

```
MOV AX(4), teliElemekSzáma(4)
DEC AX(3)
```

```
MOV teliElemekSzáma(5), AX(5)
```

```
MOV teliElemekSzáma(3), AX(3)
```

Eredmény:

```
teliElemekSzáma= 3
```

Kritikus szakasz

Kritikus szakasz

- Kölcsönös kizárás megvalósítása.
- Olyan utasítássorozat a programban, amelyen belül a végrehajtás során egyidejűleg csak egyetlen folyamat tartózkodhat.

Kritikus szakasz felépítése

- Belépő (*entry*) utasítások (szakasz).
- Kritikus szakasz (védett utasítások).
- Kilépő (*exit*) utasítások (szakasz).

A KSZ helyes implementálásának kritériumai

- Kölcsönös kizárás biztosítása.
- „Haladás” biztosítása.
- Korlátozott várakozás.

Kritériumok

- Kölcsönös kizárás biztosítása:
  - Az egymáshoz tartozó kritikus szakaszokban mindig legfeljebb egyetlen folyamat tartózkodhat.
- Haladás biztosítása:
  - Ha a kritikus szakasz szabad és van a kritikus szakaszra várakozó folyamat, akkor ezek közül egyet engedjen be.
- Korlátozott várakozás:
  - Elkerüli a várakozó folyamatok kiéheztetését.

A kritikus szakasz implementálása

A kritikus szakasz implementálása

A rendszernek módot kell adni a kritikus szakasz:

- elejének (belépés, *entry*), illetve
- végének (kilépés, *exit*) megadására.

- Ezeket tekinthetjük szimbolikus utasításoknak.

Tisztán programozott megoldás

- Közösen használt változókon alapulnak, több processzoron párhuzamosan futó folyamatokat tételeznek fel, bonyolultak, s mindhárom kritérium teljesíthető velük.
- Két folyamat esetén alkalmazható megoldás:
  - A két folyamat: P_1 és P_2.
  - Mindkét folyamatnak:
    - egy-egy jelző: a belépési szándékot fejezi ki,
    - egy változó: megmutatja egyidejű belépési szándék esetén melyik folyamat léphet be.
  - A P_i folyamathoz tartozó belépő (*entry*) és kilépő (*exit*) szakasz.

Belépő és kilépő szakaszok

```
entry:
   flag [i] := IGAZ;
   következő := j;
   while ( flag[j]==IGAZ and
     következő == j )
     do { üres utasítás };
```

....kritikus szakasz utasításai...

```
exit:
   flag [i] := HAMIS;
```

Belépő és kilépő szakaszok P1

```
entry:
   flag [1] := IGAZ;
   következő := 2;
   while ( flag[2]==IGAZ and
     következő == 2 )
     do { üres utasítás };
```

....kritikus szakasz utasításai...

```
exit:
   flag [1] := HAMIS;
```

Belépő és kilépő szakaszok P2

```
entry:
   flag [2] := IGAZ;
   következő := 1;
   while ( flag[1]==IGAZ and
     következő == 1 )
     do { üres utasítás };
```

....kritikus szakasz utasításai...

```
exit:
   flag [2] := HAMIS;
```

Programozott KSZ megvalósítás tulajdonságai

- Két folyamat esetén alkalmazható.
- Mind a három kritikus szakaszra vonatkozó feltételt teljesíti.
- Tetszőleges n folyamat esetén használható megoldás:
  - jóval bonyolultabb.

PRAM modell kiterjesztése

- A szoftvermegoldások bonyolultak.
- HW megoldás:
  - a processzor utasításkészletébe épülve támogatják a folyamatok szinkronizációjának megvalósítását hardvertámogatással.

KSZ megvalósítása HW támogatással

Speciális, megszakíthatatlan, egyidejűleg több műveletet végző hardver utasítások:

- TestAndSet(lakat),
- swap(lakat, kulcs),
- szemafor.

TestAndSet

- Korábbi megoldások: a jelző értékét többen is kiolvasták, mielőtt az első olvasó azt szabadról foglaltra (igaz) állította volna.
- PRAM modell módosítása: a foglaltságjelzőkre bevezetett TestAndSet utasítás.
- Az utasítás kiolvassa és az olvas művelethez hasonlóan visszaadja a jelző értékét, majd azt foglaltra állítja.
- Ha egy szabad jelzőre több TestAndSet utasítást egyidejűleg hajtanak végre, ezek közül csak egy ad vissza szabad értéket, a jelző végső értéke foglalt lesz.

TestAndSet

TestAndSet(lakat) művelet működése:

- lakat: boolean változó,
- beállítja a lakat változót *IGAZ* értékre,
- visszatér a lakat változó eredeti értékével.
- Használat:
  - lakat = IGAZ, ha foglalt a KSZ.

TestAndSet - KSZ megvalósítása

```
entry:
    while TestAndSet(lakat) do
    { üres utasítás };
```

(amíg a rekesz zárt, tartsd zárva és ne lépj be a kritikus szakaszba)

(ha a rekesz nyitott, zárd le és lépj a kritikus szakaszba)

....kritikus szakasz utasításai...

```
exit:
    lakat:= HAMIS;
```

(ha végeztél, engedd meg másnak a belépést)

Swap

- A közös memória egy rekeszének és a folyamat saját memóriája egy rekeszének tartalmát cseréli fel oszthatatlan művelettel.
- swap(lakat, kulcs) művelet működése:
  - lakat, kulcs: boolean változók,
  - kicseréli a lakat és a kulcs változókban tárolt értékeket.
  - Használat: lakat = IGAZ, ha foglalt a KSZ.

Swap

```
entry:
   kulcs := IGAZ;
   do
        swap(lakat, kulcs);
   while (kulcs == IGAZ);
```

(Amíg a lakat zárt, tartsd zárva és ne lépj be a kritikus szakaszba, ha a lakat nyitott, zárd le és lépj a kritikus szakaszba)

....kritikus szakasz utasításai...

```
exit:
   lakat:= HAMIS;
```

TestAndSet, Swap - tulajdonságok

- ✓ Kölcsönös kizárás biztosítása.
- ✓ Haladás biztosítása.
- ✗ Korlátozott várakozás.

- A harmadik feltétel kielégítése: jóval bonyolultabb megoldások.

Szemafor

- E. W. Dijkstra definiálta az 1960-as évek végén:
  - egy vasúti példa alapján: 1 vágányos sínszakasz védelme.
- A szinkronizációs problémák megoldásának univerzális eszköze.
- Egy speciális változó, amelyet csak a hozzá tartózó két oszthatatlan művelettel lehet kezelni:
  - ha egy folyamat módosítja a szemafor-értéket, akkor egy másik folyamat már nem módosíthatja egyidejűleg azt.
- Speciális adatszerkezet és a rajta értelmezett oszthatatlan utasítások.

Szemafor

- Szemafor, s: egész típusú változó, amely a kezdeti érték beállításától eltekintve csak két műveleten keresztül érhető el.
  - a két művelet elnevezése többféle lehet pl. Belép és Kilép, Vár és Jelez, Vizsgálat (proberen) és Növelés (verhogen).
- A műveletek oszthatatlanok:
  - init(s, v):
    - inicializálás.
  - P(s):
    - prolagen = proberen te verlagen ~ try to decrease.
  - V(s):
    - verhogen ~ increase.

Műveletek jelentése

```
init(s, v):
```

```
s := v;
```

```
P(s):
```

```
while (s <= 0) do { üres utasítás };
s := s - 1;
```

....kritikus szakasz utasításai...

```
V(s):
```

```
s := s + 1
```

Szemaforok típusai és tulajdonságai

- Az ”s” egész változó, értéke az alkalmazástól függően értelmezhető.
- Például, a kritikus szakaszba beléptethető folyamatok száma.
- Bináris szemaforok:
  - csak 0 vagy 1 (true és false) érték.
- Univerzális eszköz szinkronizációs feladatok megvalósítására.

Kritikus szakasz megvalósítása szemaforral

```
init(s,1)
entry:
    P(s)
```

....kritikus szakasz utasításai...

```
exit:
    V(s)
```

Előidejűség

Utasítás1 **megelőzi** Utasítás2-t:

```
init(s,0);
```

1. folyamat:

```
Utasítás1;
V(s);
```

2. folyamat:

```
P(s);
Utasítás2;
```

Szemafor implementálása passzív várakozással

P(s):

if s <= 0

  then *tedd a várakozó listára és altasd el*;

else s := s - 1;

V(s):

if (*várnak rá és a várakozó lista nem üres)*

  then (*leveszi és felébreszti az elsőt a várakozók közül és a KSZ-ba engedi)*;

else s := s + 1;

Szemafor tulajdonságai

- Aktív várakozás:
  - erőforrások pazarlása.
- Passzív várakozás:
  - korrekt megoldás, a szemafort nem lehet a felébresztett folyamattól ellopni.
- Szemafor általános problémája:
  - túl alacsonyszintű eszköz,
  - nehéz megbízhatóan programozni.

KSZ megvalósítása magas szintű programnyelvekben

- Magas szintű nyelvi szerkezetek.
- Erőforrások ill. változók.
- **Kritikus régió:**
  - védett közös változók jelölése,
  - hivatkozások csak a kritikus régióban,
  - **`region`** *`változó`* **`do`** *`utasítások`* **`end.`**
- **Feltételes kritikus régió:**

```
region változó when feltétel
  do utasítások end.
```

A megvalósítás problémái

- Az azonos erőforráshoz tartozó kritikus szakaszok a program szövegben szétszórtan fordulnak elő.
- Erőforrás felszabadulása:
  - Az erőforrásra várakozó összes folyamatot fel kell ébreszteni, mert a kernel nem tudja kiértékelni a belépési feltételeket.

"Monitor" típusú programszerkezet

- Ebbe programrészeket és eljárásokat zárhat a programozó.
- A programszerkezet biztosítja, hogy az abba zárt folyamat-részletek közül egyszerre csak az egyik lehet aktív. Ha már van benne egy aktív részlet, a többinek automatikusan várakoznia kell.
- A fenti "fogalmi keretekre alapozott megoldások" hátránya: azok aktív várakozással állandóan használják a processzort, a belépési feltétel tesztelésével.

Monitor

- A folyamatok bármikor hívhatják a monitorban levő eljárásokat, de a monitor belső adatszerkezetét közvetlenül nem érhetik el.
- Minden időpillanatban csak egy folyamat lehet aktív egy monitorban.
- Forrásnyelvi konstrukciók, ezért a fordítóprogram tudja, hogy ezek speciálisak, másképpen kezeli a monitoreljárásokat, mint más eljáráshívásokat.

Monitor

- A közösen használandó erőforrás és az összes rajta végezhető művelet egyetlen szintaktikus egységben.

A monitor felépítése

- Privát kívülről el nem érhető adatok.
- Ezen adatokon csak a saját eljárásai tudnak műveletet végezni.
- Az eljárások kívülről is hívhatók, a belépési pontok segítségével.
- Inicializáló rész.

Monitor

- Automatikusan kölcsönös kizárást biztosít a belépési pontok (entry point) eljárásaira.
- Csak kölcsönös kizárást valósít meg.
- Egyéb szinkronizáláshoz más eszköz kell:
  - pl. feltételes változó.

Szinkronizáció monitorral

- Feltételes változó (conditional variable).
- Bináris szemaforhoz hasonló adattípus.
- Két speciális belépési ponttal:
  - wait() illetve delay():
    - mindig blokkol,
    - a folyamat várakozó állapotba kerül.
  - signal() illetve continue():
    - csak akkor van hatása, ha várakozik valaki,
    - felébreszti a változóra várakozó folyamatokat.

Monitor - problémák

Feltételes változók:

- alacsony szintű eszköz,
- nehéz áttekinthetően és biztonságosan szinkronizációt szervezni.