Párhuzamos és elosztott rendszerek

Készítette: Dr. Mileff Péter

2. Processz menedzsment az elosztott rendszerekben

Processzek az elosztott rendszerekben

<u>Processzek és rendszer struktúra</u>

a. A processzek ugyanazon címtérben helyezkednek el

1. **Osztott kód szegmens**
   - a processzek címzési területei védve vannak és megoszthatók a processzek között.
   - minen processz címzési területe egy virtuális címtérben értelmezett és ugyanazon fizikai memóriára van leképezve.
2. **Osztott adat szegmens**
   - a processzek rendelkeznek egy olyan megosztó mechanizmussal, amelye segítségével bizonyos adatokat megosztanak, bizonyodokat pedig elzárnak a külvilágtól.

b. A processzek külön címtérben helyezkednek el

1. **Elosztott processzek**
   - a processzek külön számítógépeken vannak és saját címzési teret foglalnak.
   - a processzek vezérlő blokkjai és állapotai elosztottak az elosztott rendszer számítógépei között.
   - a folyamatok kommunikációs hálózaton keresztül kommunikálnak egymással
   - egy feladat folyamatai nem csak 1 számítógépen, hanem akár több gépen is végrehajtódhatnak egyszerre (távoli futtatás, processz migráció)

<u>Processz típusok</u>

a. Nehézsúlyú processzek

- saját címtérben futnak és saját bejegyzése van a processz vezérlő blokkban, a címzési teret nem osztják meg más processzekkel.

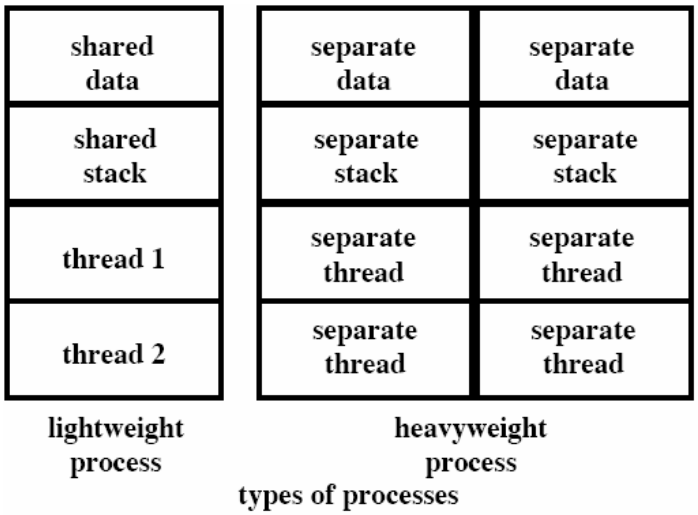
- minden processzben egy szál található

b. Közepes súlyú processzek

- kompromisszum a könnyűsúlyú és a nehézsúlyú processzek között
- a vezérlésnek több szálja van, ahol minden szál egy szeparált processzként értelmezhető

c. Könnyűsúlyú processzek

- több mint 1 szál tartozik a processzhez, és mindegyik ugyanazt az adatot, stack-ek, és szövegeket használja.
- a szálak számára megengedett ugyanazon memóriaterület írása és olvasása
- uniprocesszoron hajtódnak végre, vagy egy osztott memóriás multiprocesszoron



Processz műveletek az elosztott rendszerekben

Új processz létrehozása

process creation procedure

1. A cél „host" kiválasztása

- a processzek egy processzorhoz allokálódnak. Ennek két módja van:
  - a rendszer allokál minden processz, és ő irányítja a processz migrációt.

- a programozó vagy a felhasználó hozza létre a processzeket a megfelelő processzoron.

2. **A futatási környezet létrehozása**
   - a futási környezet az alábbiakat tartalmazza:
     - címzési terület inicializált adatokkal
     - a portok kezdeti halmaza a szolgáltatásokkal való kommunikációnak

Megközelítések a címzési terület definiálására:

a. **statikusan definiált futási környezet**
   - a címterületek egy lista alapján definiálhatók, amely definiálja a méretüket.

b. **dinamikusan definiált futási környezet**
   - figyelembe veszi az aktuális környezetet
   - a címzés területeket a gyermek processzek örökölhetik, vagy figyelmen kívül hagyhatják
   - egy örökölt terület osztozhat a szülő területével vagy logikai másolat készül a területről.

Megközelítések portok létrehozására és allokálása:
- az interprocessz komm. a portok közötti üzenetváltáson alapszik.
- egy processz gyakran egy saját adat porttal, és egy renszerhez tartozó szerver porttal jön létre azért hogy tudjon kommunikálni a szerverrel
- csakis a port létrehozója képes üzeneteket fogadni a porton keresztül
- ezeket a jogokat a létrehozó processz minden olyan processznek elküldheti, amelyikkel kommunikálni akar.

a. Port létrehozható akkor, amikor egy processz létrejön:
   - a szülő processz egy gyermek processznek portok halmazát adhatja, amely magában foglalja az input output portokat is, amelyek azonnal használhatók is.

b. Port létrehozható akkor, amikor egy processz igényli:
   - a szülő processz portokat ad a gyermek processznek, hogy csakis a rendszerrel tudjon a kommunikálni
   - amikor a gyermek processz kommunikálni akar más processzekkel is, akkor ennek igényelnie kell portokat a kommunikációhoz.

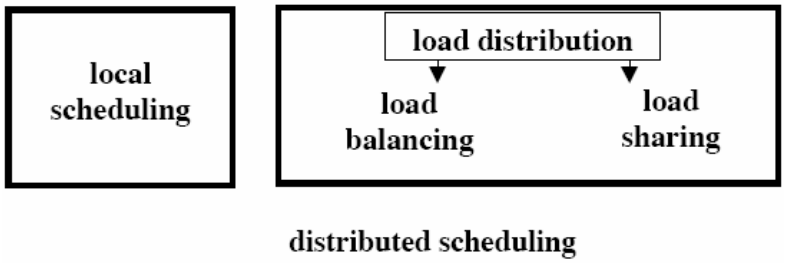
3. **Szál létrehozása a processzen belül**

Processzek terminálása

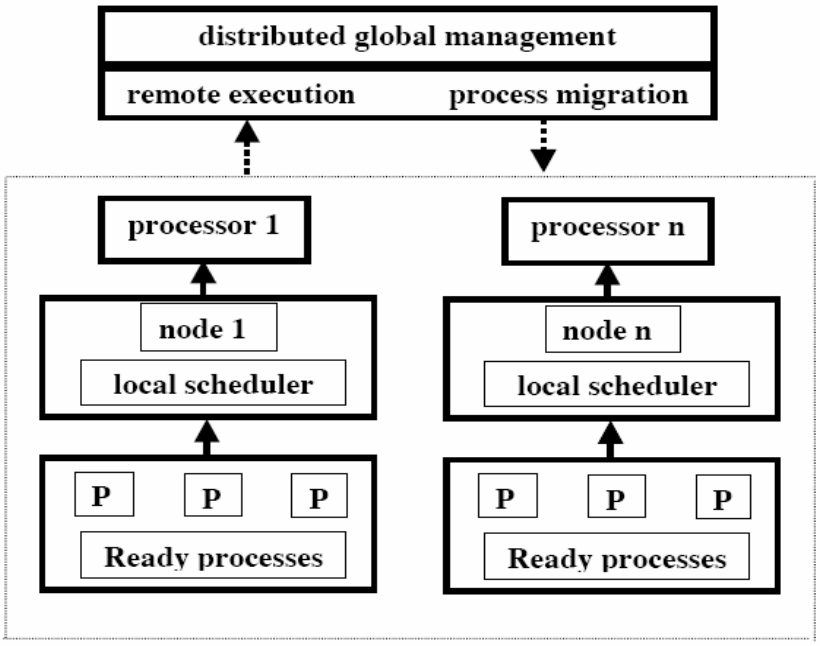
- néhány rendszerben a processzek örökké léteznek, ezért processz termináló mechanizmus szükséges. Számos termináló módszer létezik:
    1. a processz automatikusan terminálódik, amikor a program futási listája végetér.
    2. a szülő processz képes a gyermek processzek megszüntetésére a „kill" paranccsal.
- egy folyamat portja a következőképpen terminálódhat:
    1. bármikor, amikor a processz tulajdonosa akarja
    2. amikor a processz megszűnik, akkor az összes allokált portja megszűnik.

Elosztott folyamat ütemezés és elosztás

- **Ütemező:** egy futásra kész folyamatot egy rendelkezésre álló processzorhoz rendel.
- Az ütemezést globálisan és lokálisan is el kell végezni
- **Elosztó:** a szabad listáról az ütemező által kiválasztott folyamatot leveszi a listáról, megváltoztatja a státuszát, és „betölti a processzorba".
- Az elosztás mindig lokálisan történik.
- **Az elosztott ütemezést az alábbi két csoportja bonthatjuk:**



a. A lokális ütemezés az, ahogyan egy processzort a meglévő processzekhez rendelnek
    - a helyi ütemezés nem garantálja a legjobb teljesítményt a processzek számára, így a processzeket globálisan kell ütemezni
b. A terhelés elosztás, amely a teljes terhelést az egész elosztott rendszer gépei között osztja meg távoli eljáráshívással és processz migrációval.
    - a terhelés elosztást „load balancing"-al és terhelés megosztással van elvégezve:
    - **load balancing:** különböző gépek processzorain próbálja a terhelés egyenletesen elosztani.
    - **terhelés megosztás:** biztosítja azt, hogy egyetlen gép se várakozzon üresen, miközben néhány processzor pedig túl van terhelve.



local and distributed scheduling

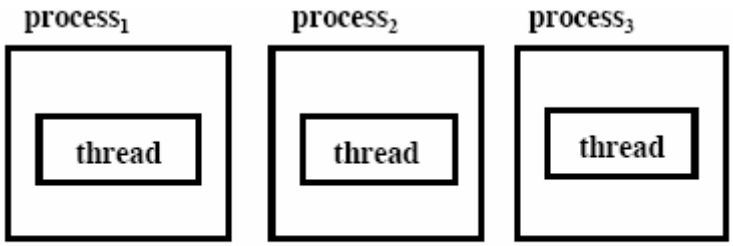
Szálak az elosztott rendszerekben

Processzek és szálak

Probléma: amikor egy processzhez valamilyen aktivitást társítunk, akkor a létrehozása és a menedzselése elég erőforrás igényes.

Megoldás: egy processz és a futási környezete egy vagy több szállal.

- egy process futási környezete az alábbiakat tartalmazza:
  - címzési terület
  - alacsony szintű erőforrások (szál kommunikáció és szinkronizáció)
  - **Megjegyzés:** magas szintű erőforrások (pl. a fájlok vagy ablakok) nem része a futási környezetnek, hanem a szzerverek biztosítják.

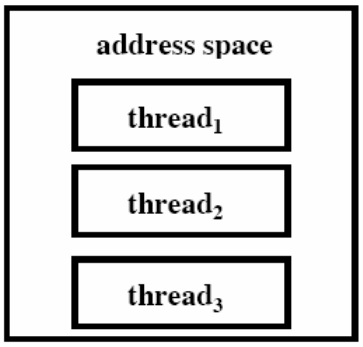


three processes with single-thread

Szál (Thread): Egy processz aktivitásának az operációs rendszer szintű absztrakciója.

- a szálak megosztanak minden erőforrást egy futási környezeten belül azért, hogy csökkentsék a processz menedzsment költségét. A szálak így osztoznak a globális változókon és azonos operációs rendszer erőforrásokat használnak.
- más futási környezetek adatai és erőforrásai hozzáférhetetlenek más futási környezet száljai számára.
- a szálak létrehozhatnak és megszüntethetnek gyermek processzeket
- a szálak változtathatják állapotukat végrehajtásuk alatt, blokkolódhatnak rendszerhívások kapcsán.

processz komponensek	**szál komponensek**
saját program számláló	saját program számláló
saját regiszter állapot	saját regiszter állapot
saját stack	saját stack
saját címterület	közös címterület a processzeknek
különböző globális változók	azonos globális változók
védelem a folyamatok között	nincs védelem a szálak között
IPC kommunikáció	kommunikáció a közös címterületen keresztül



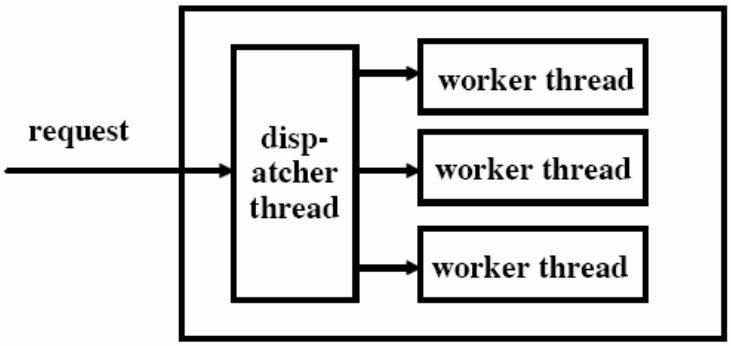
one - multithreaded - process with three threads

Szál-alkalmazási modellek

Elosztó-dolgozó modell (dispatcher-worker model):

- olyan processz, amely egy elosztó szállal és több dolgozó szállal rendelkezik.
- az elosztó processz fogadja a kliensek kéréseit és kiosztja azokat a szabad dolgozó szálak egyikének feldolgozása.
- elmenti az állapotát az adott üzenetnek és veszi a soron következő üzenetet

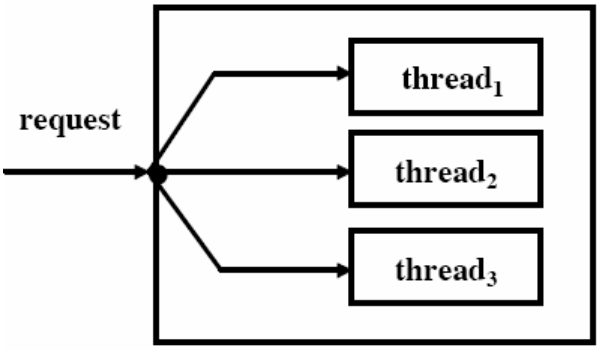
- minden dogozó szál különböző kéréseken dolgozik.



dispatcher-worker model

Csoport modell (Team model):

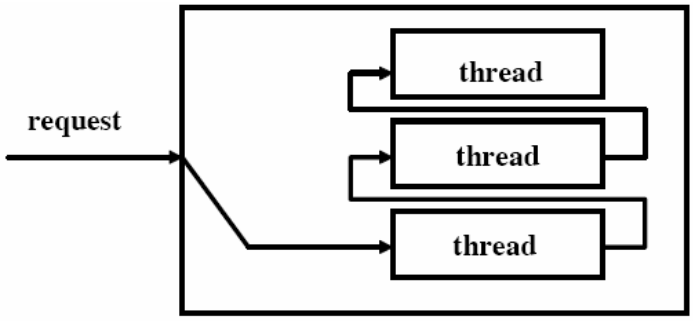
- a kliensek kérési nem osztódnak el a szálak között.
- minden szál megkapja a kliensek kéréseit. A szálak gyakran speciális kliens kéréseket szolgálnak ki.
- a rendszerben gyakran van egy munkatároló (Job Queue) a még függőben lévő kérések tárolására és azért, hogy a ki lehessen válogatni a speciális kéréseket, amelyeket a megfelelő kliensek között pedig elosztanak.



team model

Csővezeték modell (Pipeline model):

- a szálak csővezetékként vannak szervezve, így az egyik szál kimenete a másik szál bemenete lesz.
- az adatfeldolgozás ekkor szálról szálra történik.

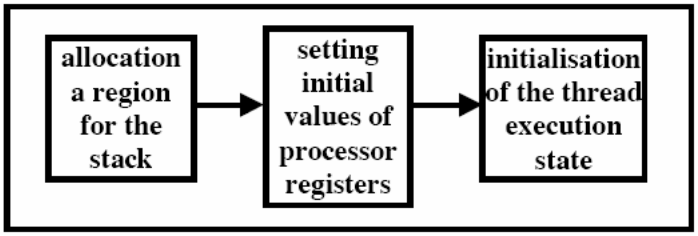


pipeline model

Szál-műveletek

1. szálak létrehozása

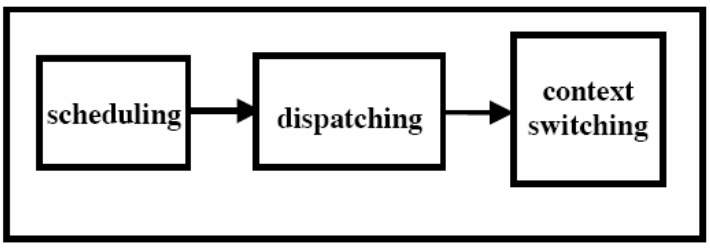
- új processz létrehozása esetén létre kell hozni a hozzá tartozó címterületet is.
- új szál létrehozása esetén nem kell létrehozni új címterületet, hanem a létrehozó processz címterületét használja.
- egy új processz létrehozásának költsége mindig nagyobb, mint egy új szál létrehozásának.



creation of a new thread

2. Szálak közötti kapcsolás

- a szálak közötti kapcsolás megosztva a címteret sokkal egyszerűbb, mint a processzek közötti váltás.
- a címtér és az operációs rendszer erőforrásainak megosztása a szálak között csökkenti a processzor kapcsolások (*context switch*) számát ellentétben a külön címterű processzekkel.



switching between threads

- a szálak közötti kapcsolás az alábbi két folyamatból áll:
    a. **Ütemezés**
        1. **nem megszakítható ütemezés:**
            - addig fut a folyamat, amíg nem intéz egy olyan függvényhívást, amely az újraütemezését eredményezi. Ekkor más szálak következhetnek a végrehajtásban.
            - a kód minden olyan része, amely olyan hívást tartalmaz, amely újraütemezést eredményezhet, automatikusan egy kritikus szekciót jelent.
        2. **megszakítható ütemezés:**
            - egy szál futása felfüggesztődhet azért, hogy más szálak futhassanak.

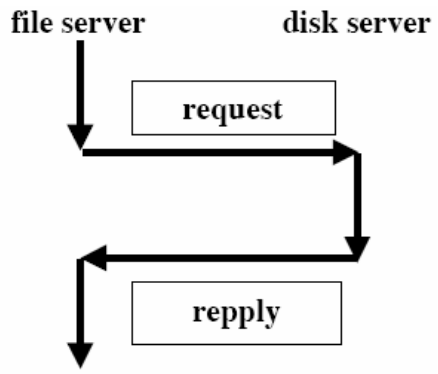
    b. **Elosztás**

    c. Kontextus váltás (Context switch): A többfeladatos rendszerek működése során a felhasználói és a kernel mód, valamint az egyes folyamatok közötti vezérlésátadás során bekövetkező művelet. A környezetváltás során a processzor szinte összes regisztere és számos más adatterület is újratöltsére kerül, ezért igen "költséges" művelet, ami szükségtelenül gyakran végrehajtva növeli a rendszer működési többletét, és csökkenti a hasznos számítási teljesítményt.
    A kontextusváltás során az alábbi műveletek történnek:
        - eredeti processzor állapot lementése
        - új állapot betöltése
        - domain átment (átmenet egy új védett környezetbe)
    - Egyazon processzhez tartozó szálak kontextus váltása a felhasználói mód és a kernel mód közötti váltáskor történik.
    - Különböző folyamatok száljai közötti kontextus váltás magába foglal egy további kontextus váltást, mégpedig a felhasználói szintű futási környezet váltását.

3. Szálakat használó szerver processz létrehozása

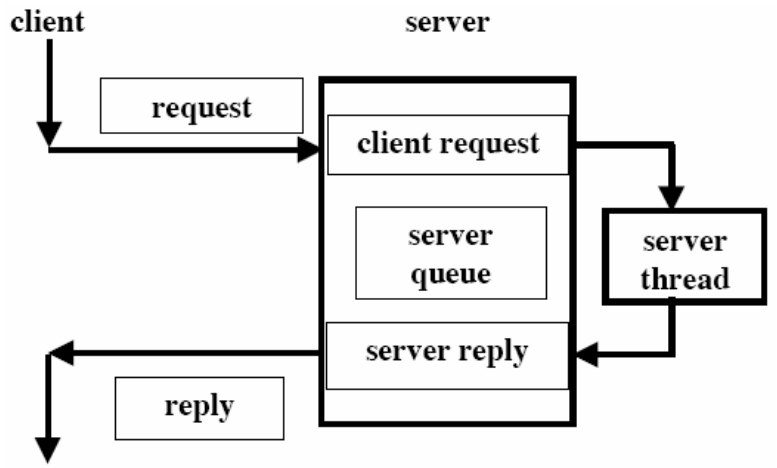
- a szálak lehetővé teszik a párhuzamosítás, a szekvenciális végrehajtás, és a blokkolási hívás komibálását.
- a szálakat használó szerver folyamatok létrehozásának számos útja van:

a. **Egyszálas processz:** blokkoló rendszerhívások párhuzamosság nélkül. Példa: a file szerver egy kérést küld a disk szerver felé:



A file szerver blokkolódik, amíg a válasz meg nem érkezik a disk szervertől.
<u>Előnyök:</u> könnyű használat, blokkoló rendszerhívások.
<u>Hátrány:</u> alacsony teljesítmény

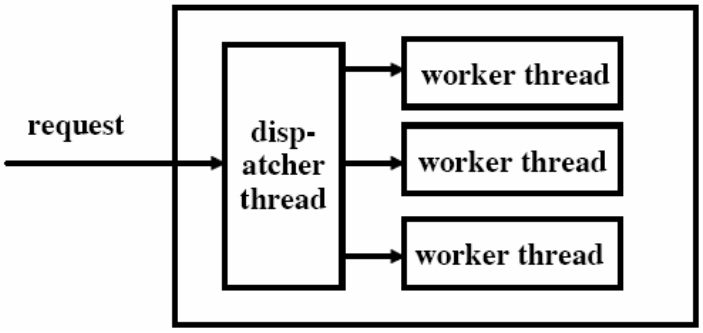
b. **Véges állapotgép:** nem blokkoló rendszerhívások és párhuzamosság.
- Példa: egy szerver egy egyszálas processz, amely véges állapotgépként működik.
- van egy esemény tároló a kliens kéréseknek és a szerver válaszüzeneteknek.
- amikor a szerver szál szabaddá válik, veszi a következő üzenet az esemény tárolóból.
- Amennyiben kliens kérés az üzenet, ellenőrzés történik, hogy a kliensnek van-e hozzáférni az adott szerverhez.
- Ha a kérés kiszolgálásához szerver hozzáférés szükséges, egy hozzáférési üzenet kerül küldésre



- nincs blokkolódás, a szerver elmenti a kliens kérésének állapotát egy táblázatban és azután kiveszi a következő üzenetet az esemény tárolóból.
- ez a kivett üzenet lehet új kliens kérés vagy pedig szerver válasz

- amennyiben az üzenet egy szerver válasz, akkor a hozzá tartotó kliens kérés állapota „felhozódik" a táblázatból
  Előnyök: nagy teljesítmény, párhuzamossággal együtt.
  Hátrány: nehéz a programozása úgy, hogy az eredmény nem blokkoló hívások együttese legyen.

c. **Szálak csoportja:** blokkoló hívások és párhuzamosság
   - **Példa:** egyetlen elosztó szál és több dolgozó szál
     - az elosztó szál végtelen ciklusban várakozik a kliensek kéréseire
     - amikor egy kliens kérés beérkezik, a hozzáférési engedély ellenőrződik. Amennyiben engedélyezett, az elosztó szál vagy készít egy új dolgozó szálat, vagy pedig a dolgozó szálak csoportjából választ ki egy szabad szálat, és a kliens kérést hozzá irányítja.



   - A dolgozó szál ellenőrzi, hogy a kérés igényel-e szerver hozzáférést.
   - Amennyiben igényel, „szerver hozzáférés kérést" intéz a szerverhez, és a szál blokkolódik várva a választ a szervertől.
   - Ekkor az ütemező újabb futásra kész szálat választ ki a listából.

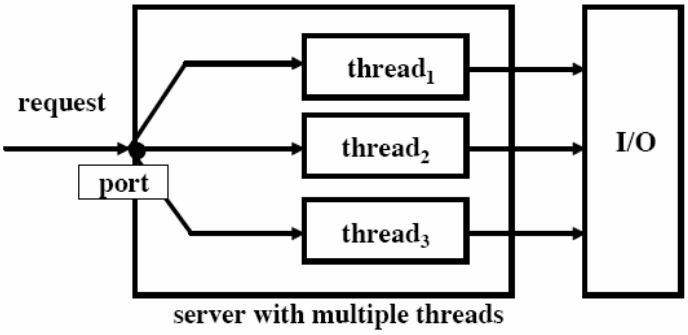
4. Erőforrás megosztás szálakkal

- az erőforrás megosztás sokkal egyszerűbb, mert egy processz minden szállja azonos címzési teret használ.

Szálak alkalmazása a kliens és szerver oldalon

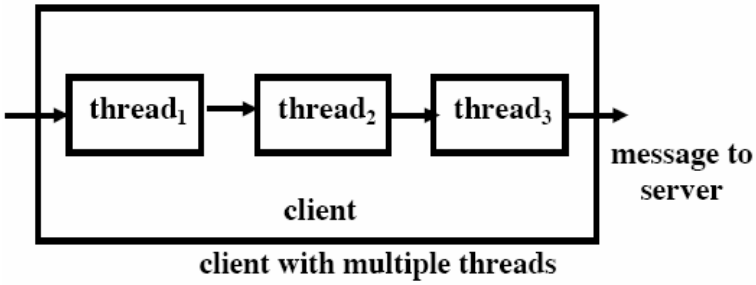
<u>Szerverek több szállal</u>

- a szerver üzenetet kap egy portról.
- minden üzenet feldolgozása azonos eljárással történik.
- amikor egy szál blokkolódik valamilyen input-output miatt, akkor egy másik szál kerülhet ütemezésre.



Kliens több szállal

- egy szál az átküldendő eredményt generálja, ami kommunikációs eljáráshívással kerül át a szerverre. Ekkor a kommunikáció blokkolja a hívót.
- ennek elkerülésére egy második szálat alkalmaznak, amelynek feladata a kommunikáció elvégzése. Ekkor ez a szál fog blokkolódni.
- az első szál az eredményt egy pufferbe rakja be, amelyet a második szál ürít, így az első szálnak lehetősége van a további futásra.



Szálcsomag

Probléma:

- minden szál saját stack-el rendelkezik, így a függvények lokális változói mindig a szál sajátjai. A globális változók esetében azonban nincs védelem arra, hogy a különböző szálak ne tudjanak hozzáférni.
- együttműködés és a koordináció megvalósítása a szálak között

Megoldás:

- **Szálcsomag:** olyan primitívek halmaza, amely támogatja több szál alkalmazását.

Szálcsomag szolgáltatásai:

1. **Szál létrehozás**
   - **statikus létrehozás:** a szálak száma fix marad a processzek egész élete során és minden szál saját stack-el rendelkezik.
   - **dinamikus létrehozás:** egy processz egy szállal indul, az új szálak futás közben jönnek létre, amikor szükség van rájuk.

2. **Szál terminálás**

   Alapvetően két módszerrel terminálható egy szál:

   - egy szál „megöli" magát egy „exit" hívás segítségével, amikor végzett a munkájával.
   - egy szál terminálható kívülről is, a „kill" parancs kívülről való használatával.

3. **Szál szinkronizálás**

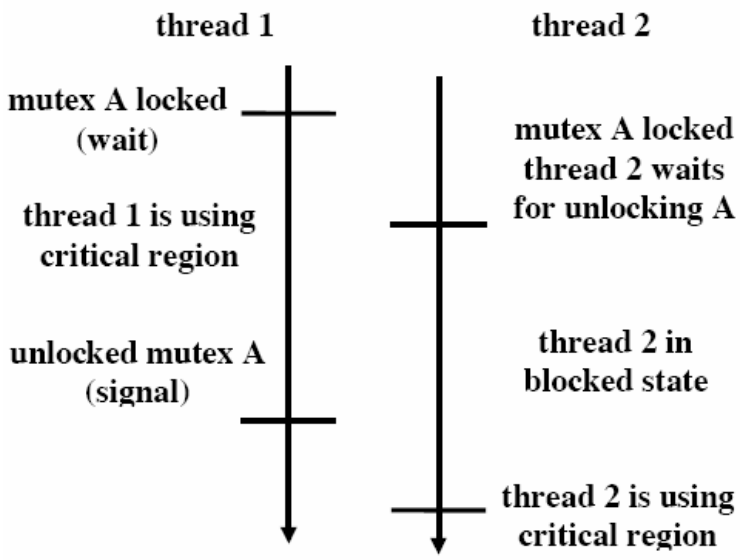
   - az alapvető probléma az, hogy meg kell védeni az adatokat attól, hogy több szál egyszerre próbálja meg elérni.

   - **Megoldás:** *kritikus szakasz* és *kölcsönös kizárás* alkalmazása.
     a. **Mutex *változók*** – bináris szemafor zárolt és nem zárolt állapottal, amelyet rövid távú zárolásra alkalmaznak.

        Műveletek mutex változókon:

        Zárolás (Lock): zárolja az kritikus szakasz hozzáférését más szálak számára
        Feloldás (Un-Lock): feloldja valamely kritikus szakasz zárolását, engedélyezve a hozzáférést más szálaknak is.

     b. **Feltételes változók (condition variable):** egy mutex változó és egy hozzá tartozó „boolean" állapot. Hosszú távú zárolásra alkalmazzák.
        Műveletek feltételes változókon:
        - **várakozás (wait):** a szál addig blokkolódik, amíg valamelyik másik szál elnem végez egy *jelző műveletet* a feltételes változón.
        - **jelzés (signal):** feloldó művelet. Az a szál, amelyik blokkolva volt várva a feltételes változóra, elkezdi futását a kritikus szakaszban.



4. Szál ütemezés

Egy szálcsomag az alábbi ütemezési lehetőségeket támogatja:

a. **Prioritás hozzárendelés:** egy prioritási érték társítása a szálakhoz azért, hogy biztosítva legyen az, hogy a fontosabb szálak gyakrabban futhassanak.
b. **Dinamikus ütemezés:** a szálak időszeleteinek nagysága változhat figyelembe véve a rendszerben található szálak számát.
c. **Kézi ütemezés:** a soron következő szál megnevezése azért, hogy a futó szálak listáját kihagyja. A processzor közvetlenül erre a szálra kapcsol.
d. **Rokonsági ütemezés:** az a szál kerül ütemezésre, amelyik legutoljára futott feltételezve azt, hogy a címterület egy része még mindig a processzor cache-ben van.

5. Kivétel és megszakításkezelés

Egy szálcsomag az alábbi kivétel és megszakításkezelési lehetőségeket támogatja:

a. **a kivétel és megszakítás jelzéseket helyesen kell lekezelni**
   - minden processz tartalmaz egy külön kivételkezelő szálat, amely a processz miden száljának minden kivételes eseményét lekezeli.
   - amikor a szál kap egy jelet valamilyen kivételről, akkor a kivétel üzenetét továbbítja a kezelő szállnak és várakozik, amíg a kivétel kezelve nem lesz.
b. **a kivétel és megszakítás jelzéseket meg kell előzni a veszteség elkerülése miatt**

- minden szál saját globális változóval rendelkezik a kivételes események jelzésére. A szálak e globális változói között nincs konfliktus.

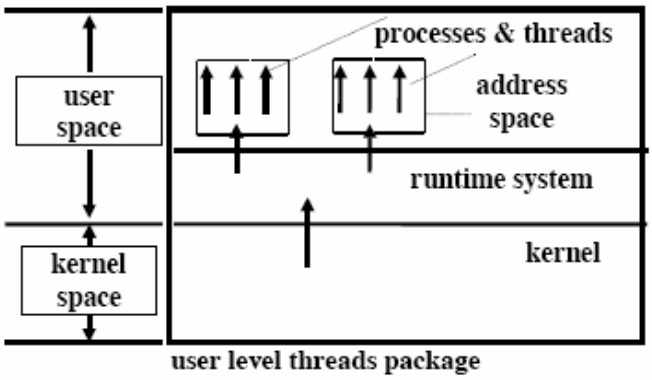
Szálcsomagok megvalósítása

Valamely szálcsomag megvalósításának legfontosabb nehézségei:

- a szálaktól érkező blokkoló rendszerhívások kezelése
- egy processz száljainak ütemezésre futásra

Felhasználói szintű szálcsomagok

- a felhasználói (memória) terült futásidejű rendszerekből áll, amelyek szálkezelő rutinok összessége.
- a szálak a felhasználói memóriaterületen futnak a futásidejű rendszer felett



- a kernel semmit sem tud a processzekről, és egyszálas processzként menedzseli őket.
- a felhasználói processzek minden rendszerhívásai a futásidejű rendszer eljárásaihoz érkezik, amelyek elvégzik a hívásnak megfelelő feladatot. A futásidejű rendszer egy táblázatot tart fent a szálak státusz információinak.
- egy futó száltól érkező blokkoló rendszerhívást is a futásidejű rendszer kezeli
- amikor egy szál egy rendszerhívást végez, akkor a futásidejű rendszer egy függvényét hívja meg, amely ellenőrzi, hogy a szállnak fel kell-e függesztődni-e, vagy nem. Amennyiben felfüggesztés szükséges, akkor a szál futása felfüggesztődik és szálváltás következik be.
- Ilyen esetekben csakis a szál lesz blokkolt állapotú, amely bekerül a futásidejű rendszer egy tárolójába, és a processz futása folytatódik.

- **kétszintű ütemezés van jelen**: a kernel ütemezője processzeket allokál és a futásidejű rendszer ütemezője pedig szálakat kezel.

Felhasználói szintű szálcsomagok előnyei:

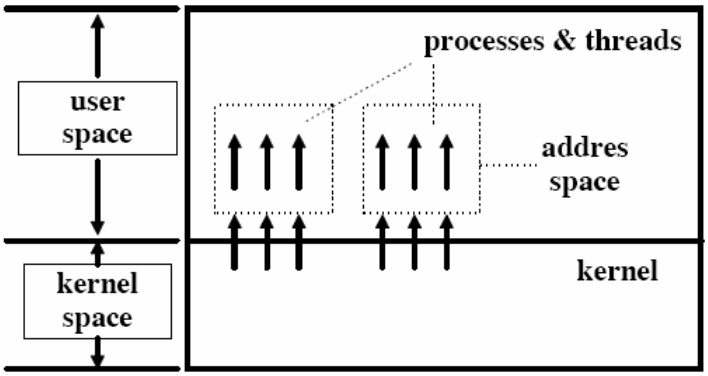
- a szálcsomagok tetszőleges olyan létező OS felső rétegében megvalósíthatók, amely nem támogatja a szálakat.
- a legmegfelelőbb ütemezési politika választható ki, hiszen a saját futásidejű könyvtárral fog futni.
- a szálak közötti kontextus váltás gyorsabb a felhasználói szintű megközelítésben, mert a kapcsolást a futásidejű rendszer fogja elvégezni, így nincs hívás a kernel felé.

Felhasználói szintű szálcsomagok hátrányai:

- a blokkoló rendszerhívások nem engedélyezettek közvetlenül, mert a procressz minden szála le lesz állítva, és a kernel másik processzt ütemez futásra. Megoldás a problémára:
  - amikor a rendszerhívás visszatár az eredménnyel, az eredmény a hívó szálnak adható, ami folytatódhat.
  - a blokkoló rendszerhívások esetleg helyettesíthetők nem blokkoló rendszerhívásokkal. Ha egy hívás felfüggesztett állapottal tér vissza, akkor a hívó szál felfüggesztődik és egy másik szál kerül ütemezésre.
- egy processzen belül megvalósított több szál nem élvez előnyt egy többprocesszoros architektúrán.

Kernel szintű szálcsomagok

Kernel szintű szálcsomagok esetén nem használnak futásidejű rendszert, miden szálat maga a kernel kezel. Pl. a szálak státusz információs tábláját is a kernel kezeli. Minden processz saját táblával rendelkezik, ahol minden szálhoz egy bejegyzés található.



kernel level threads package

- minden felhasználói szintű szálhoz egy kernel szál készül és az ütemezést is a kernel végzi.
- minden olyan hívás, ami blokkolhat egy szálat, rendszerhívásként van megvalósítva, amely a kernelhez érkezik. Pl. a szálak blokkolását és a nem blokkolását is a kernel végzi.
- egy olyan szál esetén, amelyik rendszerhívást végez, csakis maga a szál fog blokkolódni, nem pedig a processz összes szálja.
- amikor egy szál blokkolódik, a kernel vagy választ egy új szálat futásra az adott processztől, vagy választhat akár egy másik processztől is.
- azért hogy a szálak létrehozásának és megszüntetésének költsége ne legyen nagy, néhány rendszer újrahasznosítja a már létrehozott szálakat.

A kernel szintű szálcsomagok előnyei:

- a kernel nyomon tudja követni az egyes szálak processzor ideit folyamatosan, így amikor egy szál kitöltötte az időszeletét, akkor a kernel megszakíthatja futását.
- ez a megközelítés blokkoló hívásokat alkalmaz, mert egy blokkoló hívás után a kernelben ragad a szál, ahol felfüggesztődik és ekkor a kernel egy új szálat indít.

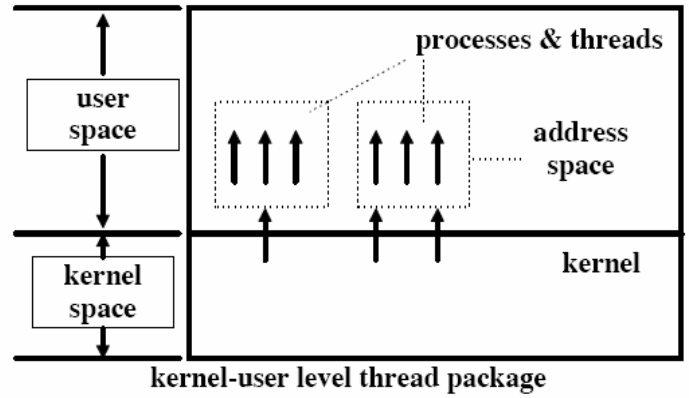
A kernel szintű szálcsomagok hátrányai:

- magát a szálak alkalmazásának koncepcióját az operációs rendszer kerneljének tervezésébe bele kell venni.
- a kontextus váltások gyakrabbak, mint a felhasználó szintű szálcsomagoknál, mert a váltást itt a kernel végzi el.
- Egyszintű ütemezés: a felhasználó rendszerhívás paramétereken keresztül választhatja ki az ütemező algoritmust és társítat prioritást a szálakhoz.
- gyenge skálázhatóság, mert a szálak státuszinformációs tábláját a kernel tárolja.

Kernel-felhasználó szintű szálcsomagok

- A megközelítés kombinálja a kernel szintű szálcsomagokat és a felhasználói szintű szálcsomagokat. A cél a rugalmasság és a maximális lehetséges teljesítmény elérése.
- a felhasználói szálak a kernel szálakhoz társítottak az alábbiak figyelembevételével:
    - mennyi darab processzor áll rendelkezésre, és melyik szálnak érdemes párhuzamosan futni a különböző processzorokon.

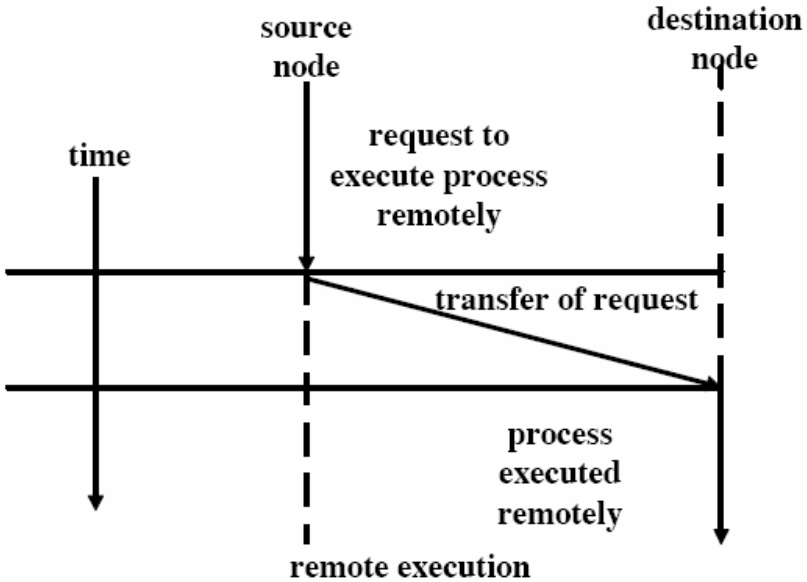
- mely szálak fognak várhatóan blokkoló hívásokat eredményezni



Távoli végrehajtás (Remote Execution)

TÁVOLI VÉGREJTÁS: egy processz áthelyezése és végrehajtása egy távoli gépen.

- Az elosztott rendszerben minden számítógép rendelkezik processz szerverrel, amely a terhelés információkat, a távoli host-okat, a távoli műveleteket és a kliens és szerver összekapcsolását menedzseli. Minden szerver egy listát tart fent olyan regisztrált host-okból, amelyek készen állnak a távoli végrehajtásra.
- a távoli végrehajtást kezdeményezheti a processz vagy a helyi processz szerver.
- a távoli host kiválasztása után egy tárgyalási folyamat indul akkor, amikor a kliens processz szerver jelzi az erőforrásigényt a távoli processz szerver felé.



- amennyiben a klienst sikerül azonosítani a távoli gépen, és a szükséges erőforrások is rendelkezésre állnak, akkor a szerver engedélyezi a távoli végrehajtást.
- ezután a processz kódja átkerül a távoli gépre, majd a szerver létrehozza és inicializálja a processzt a távoli gépen.
- a távoli végrehajtás nem engedélyezi a processzek mozgatását, amikor már létrejött. A processznek ott kell befejeződnie, ahol létrejött.

A távoli végrehajtás előfeltételei

1. Információ terjesztés:

- lennie kell valamilyen módszernek az információ terjesztésére az elosztott rendszeren belül. (A szabad gépek rendelkezésre állása, azonos számítógépek azonosítása a DS-en belül)

2. Megszakíthatóság:

- egy számítógép távoli használatának megszakíthatónak kell lennie, amikor a gép tulajdonosa úgy kívánja.
- a megszakítás meggátolja a teljesítménycsökkenést, amikor egy távoli processz fut egy számítógépen.

3. Helyfüggetlenség

- a távoli processzek ugyanúgy viselkednek mint a helyi processzek, kezelésük azonos.

4. Terhelés megosztás

- a számítási terhelés elosztására van szükség az elosztott rendszeren belül terheléselosztó algoritmusok alkalmazásával.

5. Átlátszóság

- a végrehajtás akkor „átlátszó”, ha egy programot nem távoli végrehajtás specifikusan kell megírni.

6. Rendszer különbözőség

- amennyiben egy programot egy heterogén gépről, távolról hívnak meg, szükséges lehet a program újrafordítása
- a fordítás megvalósításához gépfüggetlen, „középszintű” nyelv használatos. Pl. Java nyelv.

A távoli végrehajtás politikák és mechanizmusok

- a távoli végrehajtás egy terhelésmegosztó és kiegyenlítő módszer, amely alapvetően két részre bontható:

  a. **távoli végrehajtási politika:** egy processz kiválasztása távoli végrehajtásra és annak meghatározása, hogy mikor és hová kell áthelyezni és inicializálni a processzt, valamit a kiválasztott processz áthelyezése a kívánt távoli gépre.
     - a politika megvalósítását helyi adatbázisban tárolt szükséges statisztikai információk összegyűjtésével támogatják

  b. **távoli végrehajtási mechanizmus:** módszer a kiválasztott processzek célgépen való futtatásának kezdeményezésére.

Távoli végrehajtási politika aktivitásai

A távoli végrehajtás politikája az alábbi néhány pontból tevődik össze:

a. **Átviteli politika:** annak meghatározása, hogy egy processz helyben vagy egy távoli gépen hajtódjon végre.
b. **Kiválasztási politika**: annak meghatározása, hogy melyik procesz kerül átvitelre.
c. **Elhelyezési politika:** annak a meghatározása, hogy melyik számítógép kapja meg a kiválasztott processzt.

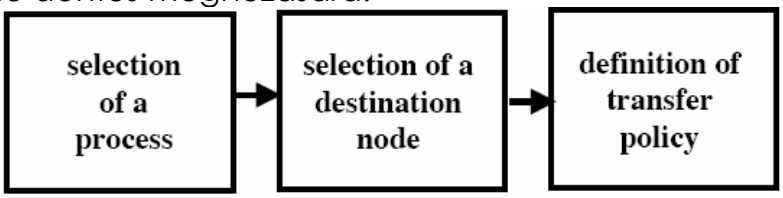
- A távoli végrehajtás politikája bizonyos adatok alapján dolgozik, amelyeket össze kell gyűjteni megfelelő gyakorisággal.
- rendelkezve az adatokkal a távoli végrehajtási politika annak definiálásával foglalkozik, hogy melyik procesz, mikor és hova kell elhelyezni.

  Adatok gyűjtése: az alábbi adatok kerülnek összegyűjtésre a döntések meghozatalához:

  1. **számítógép terhelés:** a proceszek és a kommunikációs kapcsolatok száma, valamint az, hogy a számítógép aktív vagy sem, az átlagos CPU terhelés és az átlagos hálózati terhelés.

2. **proceszek erőforrás kihasználása:** a processzek átlagos CPU és hálózati használtsága, és ezek a helyi és távoli processzekkel való teljes kommunikációja.
3. **kommunikációs adatok:** a processzenkénti legaktívabb kapcsolat meghatározására a kapcsolatok teljes forgalmának mérése a kapott és küldött csomagok segítségével.

Politika végrehajtása: alapvetően háromféle tevékenységet végeznek a megfelelő döntés meghozására:



1. Azon processzek kiválasztása, amelyek a távoli gépen fognak futni.

2. Célcsomópont kiválasztása – végrehajtási oldal – . Két megközelítés erre:
   a. A szabad számítógépek, szerverek, és a kliens kérések információinak felhasználása.
      - ezt az információt minden gép egy helyi adatbázisban tárolja
      - amikor egy szerver úgy dönt, hogy szabaddá teszi magát, akkor üzenetet küld az összes kliensnek, hogy regisztrálják a szerver rendelkezésre állását.
      - amikor egy szerver úgy dönt, hogy „visszavonul", akkor üzenetet küld az összes kliensnek. A kliensek az üzenet hatására leveszik az adott szervert a listájukról.
   b. Az erőforrás menedzserek információinak felhasználása
      - minden számítógép rendelkezik egy hálózati szerver processzel. Amikor egy számítógép szerver, egy távoli végrehajtás elosztó processz jön létre a csomópontion.
      - a távoli végrehajtás elosztó processz információkat tartalmaz az adott szerver terheltségéről.

3. Az átviteli politika definiálása: mikor és hogyan kerüljön át a kiválasztott processz a forrásgépről a célgépre.

Távoli végrehajtási mechanizmus aktivitásai

- a végrehajtási oldal kiválasztása után a processz átkerül a céloldalra.
- a következő lépés az új processz futási környezetének inicializálása. Ehhez szükséges egy kérést küldeni a helyi program menedzsernek, hogy készítsen egy új címterületet a hívó processznek.

- a program menedzser a kernel szervert használja a kívánt címterület létrehozására, és létrehoz egy „kezdeti" processzt.
- az új processz a program menedzser üzenetére vár és a hívó processz vezérlése alá kerül azzal, hogy a program menedzser továbbítja az új processz neki.
- ezután a hívó processz inicializálja az új program területet a program argumentumokkal, az alapvető I/O és környezeti változókkal
- végül elindítja az új processz futását.

Távoli végrehajtás heterogén rendszerben

Heterogén környezetben a távoli végrehajtás megvalósítása nehezebb az architekturális különbségek miatt. Különböző gépek különböző futási környezettel rendelkezhetnek. Más környezetben más névterek lehetnek, az erőforrások elnevezései, elérései mások lehetnek.
A szolgáltatások leíróinak szintaktikája és szemantikája különböző rendszereknél különbözhet. A megoldás mindig egy új, az OS feletti absztrakt régek létrehozásával érhető el, amely elrejti, és kezeli a gépek különbözőségét.

Távoli végrehajtás előnyei

1. **a processzek átlagos válaszidejének csökkentése**
    - a távoli végrehajtás csökkentheti a terhelt gépek processzeinek átlagos válaszidejét.

2. **Egyes feladatok felgyorsítása**
    - a processz átvitele egy gyorsabb, vagy szabad számítógépre.

3. **Magasabb összteljesítmény elérése**
    - a processzorok lehetőségeinek jobb kihasználása válik lehetővé a terheléselosztó algoritmusok alkalmazásával.

4. **Erőforrások hatékony kihasználása**
    - a processzek a feladatnak legmegfelelőbb gépre kerülhetnek, hogy a rendszer erőforrásokat a lehető leghatékonyabban használja ki.

5. **Terhelés redukálása**
    - a processz átvitele az erőforrás közelébe. Ez csökkenti a kommunikációs terhelést.

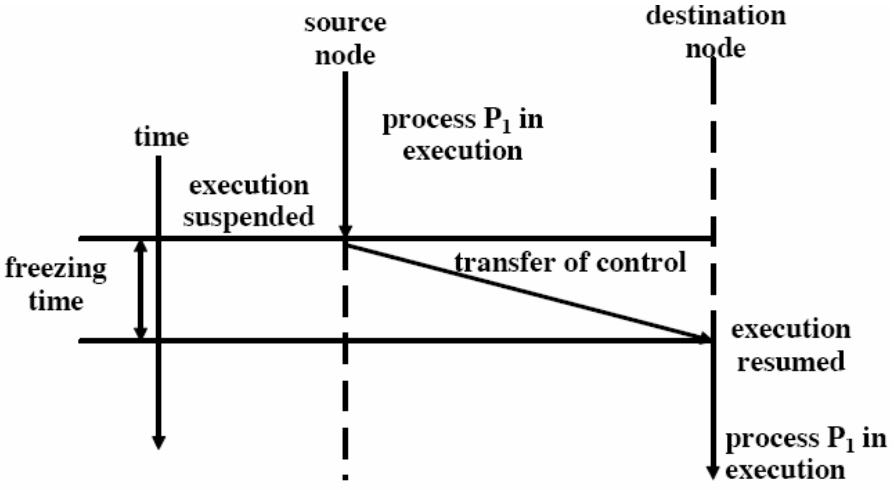
6. **A rendszer megbízhatóságának növelése**
    - a processz átvitele valamely megbízhatóbb csomópontra.

7. **A rendszer biztonságának növelése**
    - a processz olyan csomópontokra is átvihetők és ott futtathatók, amely közvetlenül nem elérhető az átlagos felhasználók számára.

Processz migráció (Processz Migration)

PROCESSZ MIGRÁCIÓ: egy processz áthelyezése az adott csomópontról egy másikra.

- a processz migráció áthelyez egy futó processzt egy másik gépre, ahol a futását folytathatja a gép összes erőforrásának folyamatos hozzáférésével.



process migration

- hasonlóan a távoli végrehajtáshoz, a processz migráció hogy meghatározza és felvegye a kapcsolatot a távoli géppel. A kód átkerül a távoli gépre, és inicializálásra kerül a távoli művelet.
- mivel a procesz átkerül a távoli gépre, a processzhez tartozó státusz információkat is át kell vinni.
- a státusz információ két részből áll: *számítási állapot, és kommunikációs állapot*.

A processz migráció sajátosságai

1. Átlátszóság:
    - Az átlátszóság szintjei:

a. Objektum hozzáférési szint:
    - Támogatja a nem megszakítható processz migrációt.
    - A programok szabad inicializálását teszi lehetővé egy tetszőleges számítógépen.
    - Elhelyezkedés független úton biztosít hozzáférést az objektumokhoz.

b. Rendszerhívások és interprocessz kommunikációs szint:
    - Támogatja a megszakítható processz migrációt.
    - Minden rendszerhívásnak és interprocessz kommunikációnak elhelyezkedés függetlennek kell lennie azért, hogy a migráció során ne függjön attól a csomóponttól, ahonnan származik.

2. Minimális interferencia
    - A migráció befolyásának a processzek feldolgozásában és az egész rendszerben minimálisnak kell lennie.
    - Ezt a migrációs idő minimalizálásával valósítják meg. – freezing time –

3. Minimális maradék függőség
    - A migráció folyamatának nem kell mindenképpen folytatódnia függve a korábbi csomóponttól.

4. Hatékonyság
    - A processz migrációban a nem hatékonyság fő forrásai:
        - Egy folyamat migrációjához szükséges idő
        - Az objektum áthelyezésének ideje
        - A távoli végrehajtás támogatásának költsége, amikor egy processz már migrálva van.

5. Robosztusság
    - Egy adott csomópont (nem az amelyik a processzt futtatja) meghibásodása ne befolyásolja a processz futását és hozzáférhetőségét.

Processz migrációs politikák és mechanizmusok

A processz migráció két fő részre bontható:

a. **processz migrációs politika:** azon processz kiválasztása, amely migrálásra kerül és annak meghatározása, hogy mikor és hová kerül migrálásra.

- a migrációhoz szükséges statisztikai információk begyűjtése és döntés hozás

b. **processz migrációs mechanizmusok:** mechanizmusok, a kiválasztott processz migrálására egy távoli gépre.

- a processz áthelyezés elvégzése figyelembe véve a migrációs politika döntéseit.

A processz migrációs mechanizmusok aktivitásai

- a migráció során átvitt információ a processz állapota és a process címterülete.

  A processz állapota:
  - a processsz futási állapota
  - a processz előrelátható memóriahasználata
  - I/O állapotok
  - processz azonosító
  - processz felhasználói és csoport azonosítói
  - a processz által hozzáférhető objektumok listája
  - a proceszz által megnyitott fájlok listája

  A processz címterülete: a program kód, adat, és stack része
  - a címterület mérete nagyobb, mint a processz állapot információinak mérete

- egy processz állapotának az átvitele általában a migrálandó processz végrehajtásának megállítását igényli, azonban néhány rendszerben a címterült átvitele nem igényli a végrehajtás leállítását.

- a processz migrációs politika során négy feladatot végeznek el:

  1. blokkolása, és befejezése a gyors I/O műveleteknek és a processz befagyasztása ott, ahol a végrehajtásban jár:

     Blokkolás: mielőtt a processz befagyasztódik, a végrehajtását blokkolni kell
     - a blokkolás rögtön elvégezhető, vagy pedig addig kell várni, amíg a processz olyan állapotba ér, amikor blokkolható
     - a blokkolás után a következő lépés az, hogy meg kell várni a processzhez társított gyors I/O műveletek befejeződését. Lassú I/O műveletek esetében azonban nincs várakozás, a műveletek a távoli gépen folytatódnak.

     Gyors I/O műveletek befejezőse: a processz befagyasztásra kerül minden gyors I/O művelet befejeződése után.

- a processz által megnyitott fileok a processz állapotinformációiban is megjelennek. Pl. file pointerek.
- ezen pointereket meg kell őrizni, hogy a migráció után a processz folytathassa működését, a fájlokhoz való hozzáférését.

Fagyasztás: miután a processz blokkolódott és az összes gyors I/O műveletek befejeződtek a processz állapota elmentődik és a kívülről érkező interakciók visszautasítósnak.

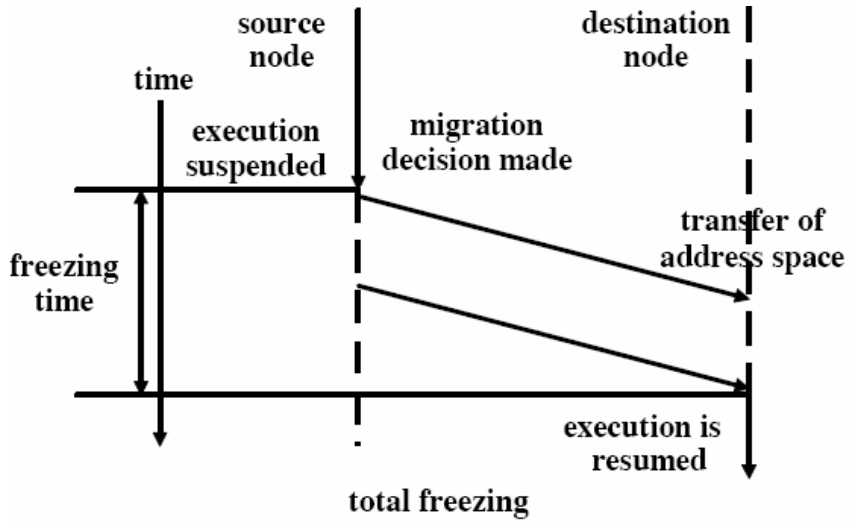
- egy processz állapotának a meghatározása nem könnyű feladat, mert az alábbiak által definiált:

  a. processz címterület
  b. helyi üzenetsor
  c. a távoli processzekkel való kommunikáció állapota

2. A processz címterületének és állapotának átvitele a távoli célgépre:

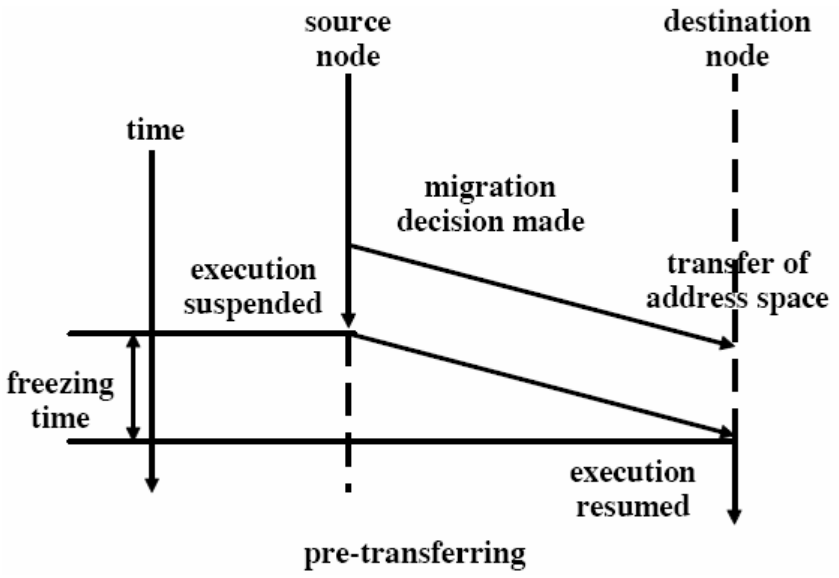
- a processz új helye transzparens marad más processzek számára. Az átvitel nem befolyásolja a migrálandó processz vagy más processz viselkedését.
- címterült átviteli módszerek:

  a. Teljes fagyasztás: a processz végrehajtás felfüggesztődik, amíg a címterület át nem kerül a távoli gépre.



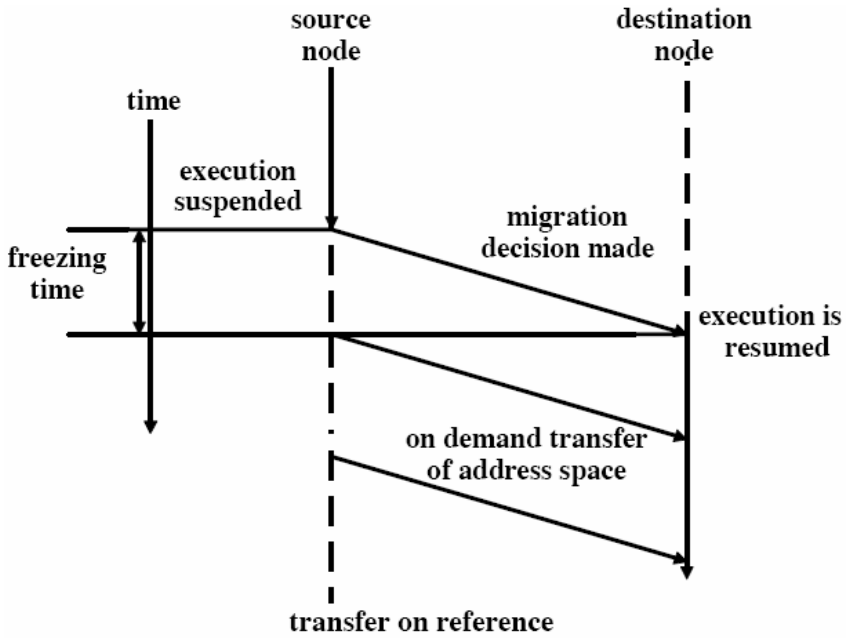
  b. Elő-áthelyezés:
  - először a címterület egy kezdeti átvitelét végzik el, ami alatt a processz tovább fut a csomóponton.
  - másodszor a processz befagyasztódik a módosított állapotának, és az utolsó átvitel alatt módosult címterületi lapok átvitele céljából.

- végül a címterület azon lapjai kerülnek átvitelre, amelyek az első átvitel alatt módosultak.



c. Referencia alapú átvitel

- a processz címterülete a forrás csomóponton marad, és az áthelyezett processz a célterületen fut. A processz megkísérel különböző memóriaterületek elérését, amely azt eredményezi, hogy ezeket a blokkokat a rendszer átmásolja az eredeti helyéről oda, hol a processz fut.



3. Processz újraindítása a kívánt csomóponton

   Újraindítás: a processz addig nem kezdi el a futását a csomóponton, amíg a teljes címterülete és az állapota átvitelre nem kerül.
   - egy üres processz állapot jön létre a processz migráció során a célgépen.
   - két másolat: egy régi és egy új másolat létezik egymás mellett a célgépen, és elérhető a migráció folyamata alatt.
   - a migrálandó folyamat állapota átvitelre kerül a régi másolatból az újba.
   - miután a migrálandó folyamat teljes állapota átkerült, az új befagyasztódik, a régi pedig törlődik.
   - ezután a migráció megtörtént, és a folyamat folytathatja futását a célgépen.

   Probléma: annak az időnek a fűdefiniálása, amikor a migrálandó folyamat elkezdheti futását a célgépen.

4. A migráció során szétválasztott processzek közötti kommunikáció kezelése
   - egy speciális található a forrás processzornál, amely a migrált processz és más processzek közötti üzenetváltást továbbítja.

   A kommunikációs állapothoz társított hibák kezelésének megközelítései:

   a. Üzenettovábbító mechanizmus:

   - Minden függőben lévő, átirányított, vagy jövőbeli üzenetnek a processz új helyére kell érkeznie.
   - amikor egy migrálandó processz felfüggesztődik a forrás csomóponton, a távoli processzek nevei és elhelyezkedései, amellyel a migrálandó processz kommunikál elmentődnek a processz a processz állapotában a célgép címével.
   - a processz átkerül a távoli gépre, de a vele kommunikáló többi processz nem informálódik a migráció tényéről.
   - a migrált processzek kommunikáló többi processz továbbra is a forrás gépre küldi az üzeneteket.
   - így amikor a processz felfüggesztődik és migrálódik, akkor az összes neki címzett üzenet elmentődik.
   - amikor a processz újraindul a célgépen, akkor az elmentett üzenetek továbbítódnak az új helyére.
   - továbbított üzenetek típusai:

Üzenet No.1.: az üzeneteket a forrás gép kapja miután a processz végrehajtása megállt a forrás gépen és még nem indult újra a célgépen.
Üzenet No.2.: az üzenetek a forrás géphez érkeznek miután a processz elkezdte futását a távoli gépen.
Üzenet No.3.: a migrált processznek küldött üzenetek miután a processz futása elindult a célgépen.

Üzenettovábbító módszerek:

1. Üzenet újraküldés

- az 1. és 2. típusú üzenetek visszairányítódnak a feladóhoz mint nem kézbesíthetők, vagy egyszerűen eldobódnak feltételezve azt, hogy a küldőnek van egy másolata az adatokról felkészítve az újraküldésre.
- a küldőt felszólítják, hogy küldje újra az üzenetet a proessz új helyére
- ez a megközelítés nem transzparens a migrált processzel kívánó más processzek kommunikációjának szempontjából.

2. Csomópont eredet (origin node)

- minden node információkat tárol a rajta létrehozott processzek aktuális elhelyezkedéséről.
- az üzenetek minden esetben ahhoz a géphez érkeznek, ahol a processz létrejött.
- az eredeti csomópont továbbítja az üzenetet a processz aktuális helyére.

2. Hivatkozás (link) bejárás

- Az 1. típusú üzenetek számára üzenetsort tartanak fent
- a migráció részeként az üzenetsorban lévő 1. típusú üzenetek továbbítódnak a processz aktuális helyére
- a 2. és 3. típusú üzenetek számára egy linket hagynak a forrás gépen, amely a migrált processz új helyére mutat.
- a link két komponenst tartalmaz: az első a csomópont azonosítója, ahol a processz létrejött, a második pedig a processz utolsó ismert csomópontjának azonosítója.
- az üzenetek továbbítása a migrált processznek ekkor a link-sorozatok bejárásával történik.

3. Hivatkozás módosítás

- a migráció folyamata alatt a forrás csomópont hivatkozás módosítás üzeneteket küld a kernelnek vezérelve a migrálandó processzel való kommunikációt.
- ezen hivatkozás módosító üzenetek tartalmazzák a migrált processz össze hivatkozásának új címét
- így bármely a migrált processznek küldött üzenet a processz aktuális helyére továbbítódik.

b. Üzenet elvesztés megelőzés

- egy a forrás gépen futó migrációs alkalmazás informálja a processz mozgatás szándékáról azon processzeket, amelyek kommunikáció kapcsolatban állnak a migrálandó processzel.
- amikor a processz migrálódik, az üzeneteket eltárolják egy pufferbe.
- miután a migráció lezajlott, a célgép migrációs alkalmazása értesti a többi gép migrációs alkalmazását a migráció megtörténtéről, és átküldi az új címet.

c. Üzenetvesztés visszaállítás

- a migrációs alkalmazás nem menti el a célgép címét.
- a migrálandó processzel kommunikáló többi processz nem értesítődik a migrációról.
- minden üzenet, ami a migráció alatt érkezik, elveszik.
- azonban a migrálandó processzel kommunikáló távoli processzek nevei és címei elmentődnek.
- miután a processz újraindul a célgépen, helyreállítja a kapcsolatot a távoli processzekkel és üzenet visszaállítást kezdeményez.

A processz migráció megvalósítása

A megvalósítás szintje ismert: Operációs rendszer kernel, Kernelen kívül, egy fordítóba és egy futásidejű csomagba ágyazva.

Processz migráció heterogén rendszerekben

Probléma: Különböző csomópontok különböző adatmegvalósítása.
Megoldás: Ezen rendszerekben minden adatot a forrásgép formátumából a célgép formátumába kell transzformálni. Általánosított formátumot használnak az adatok átvitelére. Így minden processzornak konvertálnia kell az adatokat ezen általános formátumból és formátumba.

A processz migráció előnyei

1. A processzek átlagos válaszidejének csökkentése
    - a processz migráció csökkentheti a processzek átlagos válaszidejét a terelt csomópontokon
2. Bizonyos taszkok felgyorsítása
    - processzek migrálása különböző gépekre, és konkurens futtatása
    - migrálás egy gyorsabb processzorra
3. Magasabb teljesítmény elérése
    - a processzorok lehetőségeit jobban ki lehet használni különböző terheléselosztó algoritmusok használatával.
4. Erőforrások hatékony kihasználása
    - a processzek a feladatnak legmegfelelőbb gépre kerülhetnek, hogy a rendszer erőforrásokat a lehető leghatékonyabban használja ki.
    - bizonyos erőforrások nem érhetőek el távolról. A processz migráció elősegíti ezen gépek kihasználását.
5. Kommunikációs terhelés csökkentése
    - egy processz valamely erőforrás közelébe való migrálása csökkentheti a kommunikáció mennyiségét.

6. A rendszer megbízhatóságának növelése
    - a proceszek migrálása olyan gépekre, amelyek megbízhatósága magasabb
    - valamely kritikus proccesszek másolatának migrálása valamely távoli gépre, majd az eredeti és a másolat futtatása az egyes gépeken
    - valamely csomópont hibája esetén a rajta futó processz migrálható egy másik csomópontra, ahol folytathatja a futását.
7. A rendszer biztonságának növelése
    - egy processz migrálható olyan csomópontra is, amely átlagos felhasználók számára nem hozzáférhető.