# Chapter 1

Az operációs rendszer olyan program, amely közvetítőként működik a számítógép felhasználója és a számítógép hardvere között. Nincsen általánosan elfogadott definíciója. Az operációs rendszer kezeli a CPU-t, a memóriát és az I/O eszközöket. Az operációs rendszer vezérli a hardvert, és összehangolja annak használatát a különböző felhasználói programok között.

## Az operációs rendszer egy erőforrás-elosztó és vezérlő program

A számítógép szempontjából az operációs rendszer a hardverrel legtöbbet érintkezett program. Ezért kezelhetjük erőforrás-elosztónak. Ilyen erőforrások a CPU-idő, memóriaterület, tárhely, I/O eszközök stb. Az operációs rendszernek el kell döntenie, hogyan rendelje hozzá azokat meghatározott programokhoz és felhasználókhoz, hogy hatékonyan működjenek.

Az operációs rendszer egy vezérlőprogram. Kezeli a felhasználói programok végrahajtását, hogy megelőzze a hibákat és a számítógép nem megfelelő használatát. Különösen az I/O eszközök működésével és vezérlésével foglalkozik.

## Céljai:

* Felhasználói programok futtatása és felhasználói problémák futtatása
* Kényelmessé tegye a számítógépes rendszer használatát
* Hatékonyan használja a számítógép a hardvert (A felhasználót, csak az eredmény érdekli, a folyamat nem)

## Feladatai:

* A felhasználók kényelmét, egyszerű használatot és jó teljesítményt biztosítani
* A megosztott számítógépek esetén is fenntartani a megfelelő működést
* Prioritások alapján ütemezi, melyik folyamatok a fontosak
* Végrehajtja a programokat, és meggátolja a hibákat és a nem megfelelő használatot

## A számítógépes rendszer négy részre bontható:

1. Hardver – az alapvető erőforrásokat biztosítja
   1. CPU, I/O eszközök
2. Operációs rendszer
   1. Vezérli és koordinálja a hardverek használatát a különböző változatok között alkalmazások és felhasználók
3. Alkalmazási programok
   1. Meghatározzák, hogy a rendszer erőforrásait milyen módon használják fel a számítógép számítási problémáinak megoldására.
   2. Szövegszerkesztők, fordítók, webböngészők, adatbázisok, játékok
4. Felhasználók
   1. A képen szöveg, képernyőkép, tervezés látható

      Automatikusan generált leírásEmberek, egyéb gépek

## Számítógépes rendszer működése

Egy modern általános célú számítógépes rendszer egy vagy több CPU-ból és számos eszközvezérlőből áll, amelyek egy közös buszon keresztül vannak csatlakoztatva, amely hozzáférést biztosít a megosztott memóriához.

Az egyetlen program, ami állandóan fut a számítógépen a kernel. A kernelen alapul a rendszer maga, és azon minden más. Amikor a számítógép elindul, akkor egy **bootstrap** program betölt, amit legtöbbször EPROM-ban tárolnak, és firmware-nek hívják. Először betölti a kernelt, majd a többi rendszerprogramot és alkalmazási programot. A rendszerprogramok, amelyek a rendszerindításkor betöltődnek a memóriába, hogy rendszerfájlokká váljanak, vagy olyan rendszerdémonok, amelyek a kernel teljes futása alatt futnak, ezek kernelen kívül futó szolgáltatások.

## Megszakítások

A képen képernyőkép, diagram, sor, tervezés látható

Automatikusan generált leírásA megszakítások a számítógépes architektúra fontos részét képzik. A megszakításokat a kernel kezeli.

Egy esemény bekövetkezését áltában hardver vagy szoftver megszakítása jelzi. A szoftver megszakítást válthat ki egy speciális művelet, a rendszerhívás. Amikor a CPU megszakad, leállítja, amit csinál, és azonnal áthelyezi a végrehajtást egy rögzített helyre. A rögzített hely általában azt a kezdőcímet tartalmazz, ahol a megszakítás szolgáltatási rutinja található.

A memória első száz helyén van eltárolva a megszakítási rutin mutató táblázata, ahol a címek és a futásukhoz szükséges memória van feltüntetve. Ezt a címtömböt hívjuk megszakítási vektornak. A megszakítási vetkort ezután egy egyedi eszközszám indexeli, amelyet a megszakítási kéréssel együtt adnak meg, hogy megadják a megszakítási szolgálatatás rutinjának a címét.

Példa I/O műveletre:

1. Amikor elkezdődik egy I/O művelet, elvesszük a felhasználótól az irányítást, és csak akkor adjuk vissza, ha végzett az I/O művelet
   1. Wait instruction → CPU Idle
2. Amikor elkezdődik az I/O művelet azonnal visszaadjuk a felhasználónak az irányítást, nem várjuk meg a befejeződést
   1. Rendszerhívások → Kérés az oprendszerhez, hogy a felhasználó várja meg a végét az I/O műveletnej

## Tárolási mértékegységek

A képen szöveg, Betűtípus, képernyőkép látható

Automatikusan generált leírásAlapegység 1 bit, ami 0 vagy 1-es értéket vehet fel. A legkisebb tárolási egység a legtöbb számítógépen a bájt.

A gépi szó az architetctúrától függ, egy egységet jelent az adott architectúrán keresztül.

* 64 biten egy szó 8 bájt
* 32 biten egy szó 4 bájt

A legtöbb gép nagyrészben szavakban gondolkozik, míg a hálózat bitekben, ezzel csalnak, hogy a számok nagyobbnak tűnjenek. Átviteli egységek:

* Hálózat: bit
* Storage: Byte
* Processzor: Gépi szó
* I/O rendszer: Egységek/ blokkok, ami a fájlrendszertől függ.

## Tárolási szerkezet

### Főmemória

A CPU csak a memóriából tud utasításokat betölteni, ezért minden futtatandó programot ott kell tárolni. Az általános célú számítógépek programjaik nagy részét újraírható memóriából, főmemóriából futtatják (RAM).

Egy tipikus utasítás-végrehajtási ciklus, amint az egy Neumann architectúrájú rendszeren hajtják végre, először lekér egy utasítást a memóriából, és eltárolja azt az utasításregiszterben. 2 esetben a programokat nem lehet a főmemóriában tárolni

1. A főmemória általában túl kicsi az összes szükséges program és adat tárolására tartós szinten.
2. A főmemória illékony tárolóeszköz, amely elveszti tartalmát, amikor az áramellátás ki van kapcsolva vagy más módon megszakad.

### Másodlagos memória

Kikapcsolás után is megőrzi az adatokat, emellett nagyméretű. A másodlagos tárolás fő követelménye, hogy nagy mennyiségű adatot tudjon tartósan eltárolni. A legelterjedtebb másodlagos tárolóeszköz a merevlemez-meghajtó (HDD), amely mind a programok, mind az adatok tárolását biztosítja.

### A képen szöveg, képernyőkép, diagram, Párhuzamos látható Automatikusan generált leírásTárolási hierarchia

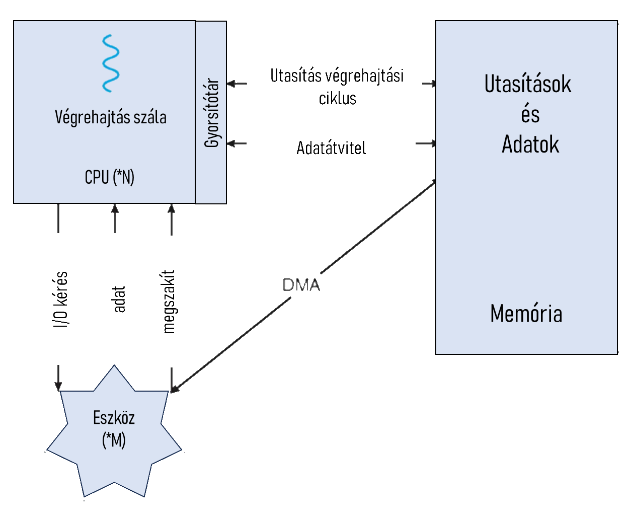
A sokféle tárolási rendszer hierarchiába rendezhető sebesség és költség szerint. A magasabb szintek drágák, de gyorsak. Ahogy haladunk lefelé, a bitenkénti költség általában csökken, míg a hozzáférési idő általában nő. Ez azért van, mert ha az olcsóbb tárolórendszer még gyorsabb is lenne, akkor nem lenne értelme a drágát használni.

## Gyorsítótár

A CPU sokkal gyorsabb, mint egy HDD/SSD vagy az I/O alrendszer, ezért vezették be a Cachet, átmenetileg ide van helyezve az adat a lassabb tárolóról.

## Eszközillesztő

Az operációs rendszerek átalában minden eszközvezérlőhöz rendelkeznek egy eszközillesztővel. EZ az eszköz-illesztőprogram megérti az eszközvezérlőt, és az operációs rendszer többi részét egységes felülettel látja el az eszközhöz.

Az I/O művelet elindításához az eszközillesztő betölti a megfelelő regisztereket az eszközvezérlőbe. Az eszközvezérlő pedig megvizsgálja ezeknek a regisztereknek a tartalmát, hogy meghatározza, milyen műveletet kell végrehajtani (például „karakter beolvasása a billentyűzetről”). A vezérlő elindítja az adatok átvitelét az eszközről a helyi pufferébe. Az adatátvitel befejeztével az eszközvezérlő egy megszakításon keresztül tájékoztatja az eszközillesztőt, hogy befejezte a működését. Az eszközillesztő ezután visszaadja az irányítást az operációs rendszernek, esetleg visszaadja az adatokat vagy egy mutatót az adatokra, ha a művelet olvasás volt.

A megszakítás vezérelt I/O ilyen formája kis mennyiségű adat mozgatására alkalmas, de nagy többletterhelést okozhat, ha tömeges adatmozgatáshoz, például lemez I/O-hoz használják. A probléma megoldására közvetlen memóriaelérést (DMA) használnak. Az I/O eszköz puffereinek, mutatóinak és számlálóinak beállítása után az eszközvezérlő egy teljes adattömböt továbbít közvetlenül a saját puffertárolójába vagy onnan a memóriába, a CPU beavatkozása nélkül. Blokkonként csak egy megszakítás jön létre, hogy jelezze az eszközillesztőnek, hogy a művelet befejeződött, nem pedig a kis sebességű eszközök számára generált bájtonként egy megszakítás. Amíg az eszközvezérlő végrehajtja ezeket a műveleteket, a CPU rendelkezésre áll egyéb munkák elvégzésére.

Nem a processzoron keresztül mozgatjuk az adatot az eszközről a memóriába, hanem közvetlenül az eszközről a memóriába.

## Számítógéprendszer architektúra

### Egyprocesszoros rendszer

A legtöbb rendszer egyetlen általános célú processzort használ. A legtöbb rendszerben speciális processzorok is vannak. A speciális célú processzorok korlátozott utasításkészletet futtatnak, és nem futtatnak felhasználói folyamatokat.

### Többprocesszoros rendszer

Az elmúlt néhány évben a többprocesszoros rendszerek (más néven párhuzamos rendszerek vagy többmagos rendszerek) kezdték uralni a számítástechnikát. Az ilyen rendszereknek két vagy több processzora van, amelyek szorosan kommunikálnak egymással, megosztva a számítógépes buszt és néha az órát, a memóriát és a perifériás eszközöket. A többprocesszoros rendszerek először a szerverekben jelentek meg, és azóta áttértek az asztali és laptop rendszerekre. A közelmúltban számos processzor jelent meg mobil eszközökön, például okostelefonokon és táblagépeken.

#### Előnyei

1. **Megnövelt áteresztőképesség**: A processzorok számának növelésével azt várjuk el hogy kevesebb idő alatt több munkát tudjon elvégezni. Viszont a gyorsítási arány nem egyenlő a processzorok számával. Amikor több processzor működik együtt egy feladaton, bizonyos mértékű többletköltség keletkezik az összes alkatrész megfelelő működéséhez. Ez a többletköltség, valamint a megosztott erőforrásokért folytatott versengés csökkenti a további processzoroktól várható nyereséget.
2. **Méretgazdaság**: A többprocesszoros rendszerek olcsóbbak, mivel megosztják egymással a perifériákat, a háttértárakat és a tápegységeket.
3. **Megbízhatóbb**: Ha a funkciókat megfelelően el lehet osztani több processzoron keresztül, akkor egy processzor meghibásodása, nem vezet a rendszer leállásához, hanem csak lassítja azt.

#### Fajtái

##### Aszimmetrikus multiprocesszor

Minden processzorhoz egy adott feladat tartozik. Egy főnök processzor vezérli a rendszert, a többi processzor pedig tőle várja el a feladatok kiosztását. Ez a séma a főnök-dolgozó sémát határozza meg.

##### A képen képernyőkép, Téglalap, diagram, tér látható Automatikusan generált leírásSzimmetrikus multiprocesszor (SMP)

Minden processzor végrehajt minden feladatot, beleértve az operációs rendszer funkcióit és a felhasználói folyamatokat is. Ebben minden processzor egyenrangú.

Minden processzornak saját regiszterkészlete és gyorsítótára van. Viszont mindegyik megosztja a memóriát. Ennek a modellnek előnye, hogy egyszerre több program is futtatható.

A képen képernyőkép, szöveg, Téglalap, diagram látható

Automatikusan generált leírásA többprocesszoros feldolgozás CPU-kat ad hozzá a számítási teljesítmény növeléséhez. Ha a CPU integrált memóriavezérlővel rendelkezik, akkor a CPU-k hozzáadása is növelheti a mennyiségét.

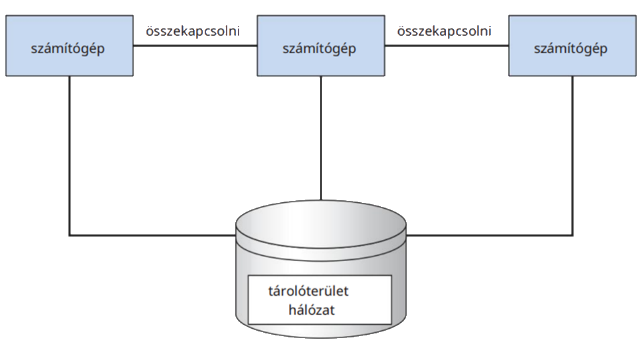
A kétmagos kialakítás két maggal ugyanazon a chipen kevesebb teljesítményt nyújt

## Clustered System

Több CPU-t gyűjt össze. Két vagy több különálló rendszerből – vagy csomópontban – állnak össze. Az ilyen rendszereket lazán összekapcsolt rendszernek nevezik. Az általánosan elfogadott definíció szerint a fürtözött számítógépek megosztják a tárhelyet, és szorosan kapcsolódnak egymáshoz egy helyi hálózati LAN-on vagy egy gyorsabb összeköttetésen, például az InfiniBandon keresztül.

A klaszterezés történhet aszimmetrikusan vagy szimmetrikusan. Az aszimmetrikus fürtözésnél az egyik gép üzem közbeni készenléti módban van, míg a másikon az alkalmazások futnak. A hot-standby gazdagép nem tesz mást, mint az aktív szervert figyeli. Ha a kiszolgáló meghibásodik, a hot-standby gazdagép lesz az aktív kiszolgáló.

A szimmetrikus fürtözésben két vagy több gazdagép futtat alkalmazásokat, és figyeli egymást. Ez a struktúra nyilvánvalóan hatékonyabb, mivel az összes rendelkezésre álló hardvert felhasználja. Ez azonban megköveteli, hogy egynél több alkalmazás legyen elérhető a futtatáshoz.

Egyes fürttermékek több tucat rendszert támogatnak egy fürtben, valamint a mérföldekkel elválasztott fürtözött csomópontokat. E fejlesztések közül sokat a tárolóterületi hálózatok (SAN-ok) tesznek lehetővé. Ha az alkalmazások és azok adatai a SAN-on vannak tárolva, akkor a fürtszoftver hozzárendelheti az alkalmazást, hogy a SAN-hoz csatlakoztatott bármely gazdagépen fusson. Ha a gazdagép meghibásodik, akkor bármelyik másik gazdagép átveheti az irányítást. Egy adatbázis-fürtben több tucat gazdagép oszthatja meg ugyanazt az adatbázist, ami jelentősen növeli a teljesítményt és a megbízhatóságot.

## A képen képernyőkép, szöveg, sor, diagram látható Automatikusan generált leírásMultitasking, dual mode

Az operációs rendszer megfelelő végrehajtása érdekében különbséget kell tudnunk tenni az operációs rendszer kódja és a felhasználó által definiált kód végrehajtása között. Legalább két különböző módra van szükségünk. A felhasználói módra és a kernel módra (amelyet felügyelő módnak, rendszermódnak vagy privilegizált módnak is neveznek.). Egy bit úgynevezett mód bit, hozzáadódik a számítógép hardveréhez, hogy jelezze az aktuális módot. A módbittel különbséget tudunk tenni az operációs rendszer nevében végrehajtott feladat és a felhasználó nevében végrehajtott feladat között.

Amikor a számítógépes rendszer egy felhasználói alkalmazás nevében fut, a rendszer felhasználói módban van. Ha azonban egy felhasználói alkalmazás szolgáltatást kér az operációs rendszertől (rendszerhíváson keresztül), a rendszernek felhasználói módból kernel módba kell váltania a kérés teljesítéséhez.

A rendszerindításakor a hardver kernel módban indul. A kettős működési mód lehetővé teszi számunkra, hogy megvédjük az operációs rendszert a tévedő felhasználóktól. A hardver csak kernel módban teszi lehetővé a privilegizált utasítások végrehajtását. Ha felhasználó módban tesznek ilyen utasítást az illegális.

A virtualizációt támogató CPU-k gyakran rendelkeznek külön üzemmóddal, amely jelzi, ha a virtuálisgép-kezelő (VMM) és a virtualizációkezelő szoftver uralja a rendszert.

## Folyamatkezelés

A végrehajtás alatt álló program, egy folyamat. Egy folyamatnak szüksége van bizonyos erőforrásokra – beleértve a CPU-időt, a memóriát, a fájlokat és az I/O eszközöket – a feladat végzéséhez. Ezeket az erőforrásokat a folyamat a létrehozásakor szerzi meg. A folyamat végezte után fel kell szabadítani minden felszabadítható erőforrást.

A program önmagában nem folyamat. A program egy passzív entitás, mint a lemezen tárolt fájl tartalma, míg egy aktív entitás. Az egyszálú folyamatoknak egy programszámlálója van, amely meghatározza a következő végrehajtandó utasítást. Kezdetben egyszerre egy folyamat megy, és a folyamatokat szekvenciálisan hatjuk végre. Ha van egyszerre több folyamat akkor meg kell valósítani közöttük a kommunikációt, és el kell osztani az erőforrásokat közöttük. Ha két folyamat egymásra vár, akkor sosem fog előrehaladni.

A folyamat a munka egysége a rendszerben. Az operációs rendszer a következő folyamatokért felelős:

* Folyamatok és szálak ütemezése a CPU-kon
* Felhasználói és rendszerfolyamatok létrehozása és törlése
* Folyamatok felfüggesztése és újraindítása
* Mechanizmusok biztosítása a folyamatok szinkronizálásához
* Mechanizmusok biztosítása a folyamatkommunikációhoz és holtpontok kezeléséhez

## Memóriakezelés

A program végrehajtásához az összes utasításnak valamint az összes szükséges adatnak benne kell lennie a memóriában. A memóriakezelés határozza meg, hogy mi és mikor van a memóriában. Ez szükséges a CPU kihasználtságának és a számítógép válaszának optimalizálásához a felhasználók számára. Minden memóriát a program végeztével fel kell szabadítani.

Az operációs rendszer a következő memóriakezelési tevékenységekért felelős:

* Nyomon követni, hogy a memória mely részeit használják és ki
* Eldönteni, hogy melyik folyamathoz és adathoz kell hozzáférés a memóriából
* Memóriaterület felszabadítása és lefoglalása

## Tárhelykezelés

Az operációsrendszer egységes, logikus képet ad az információtárolásról. A fizikai tulajdonságokat kivonja a logikai tároló egységbe a fájlba. A változó tulajdonságok közé tartozik a hozzáférési sebesség, kapacitás, adatátviteli sebesség, hozzáférési mód (szekvenciális vagy véletlenszerű).

A fájlok általában programokat és adatokat jelentenek. Az adatfájlok lehetnek numerikus, alfabetikus, alfanumerikus vagy binárisak. A fájlok lehetnek szabadformájú (pl. szövegesfájl), vagy lehetnek merev formázásúak. Lehet őket szabályozni hozzáférésekkel (írás, olvasás, hozzáfűzés). Mappákba rendezhetjük őket, hogy megkönnyítsük a használatot.

Az operációsrendszer a következőkért felelős:

* Fájlok és könyvtárak létrehozása és törlése
* Primitívek a fájlok és könyvtárak kezeléséhez
* Fájlok hozzárendelése másodlagos tárolóra
* Fájlok biztonsági mentése adathordozóra

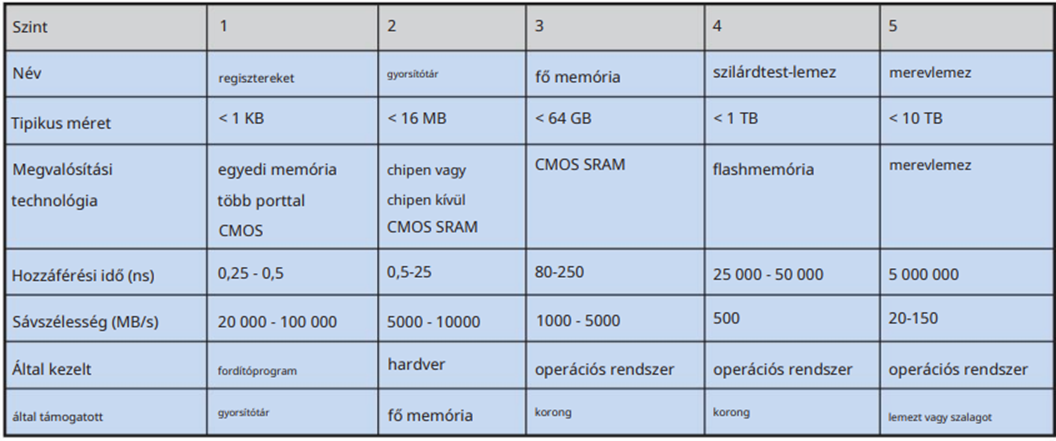
## Háttértár kezelés

Mivel a másodlagos tárolót gyakran használják, hatékonyan kell használni. A számítógép teljes működési sebessége függhet a lemez alrendszer sebességétől és az alrendszert manipuláló algoritmusoktól.

Az operációsrendszer a következőkért felelős:

* Szabadterület-kezelés
* Tárhelykiosztás
* Lemezütemezés

## Gyorsítótár kezelés

A gyorsítótár mérete korlátozott, a gyorsítótár kezelése fontos tervezési probléma. A gyorsítótár méretének és a cserepolitika gondos kiválasztása jelentősen megnövekedett teljesítményt eredményezhet.

A főmemória a másodlagos tárolás gyors gyorsítótárának tekinthető, mivel a másodlagos tárolóról a főmemóriába kell másolni az adatokat a működéshez.

### Az „A” adatok áttelepítése a lemezről a regiszterbe

A képen szöveg, Betűtípus, képernyőkép, Grafika látható

Automatikusan generált leírásA többfeladatos környezetnek ügyelniük kell arra, hogy a legújabb értéket használják, függetlenül attól, hogy a tárolási hierarchiában hol van tárolva. A többprocesszoros környezetnek biztosítania kell a gyorsítótár koherenciáját hardver olyan, hogy az összes CPU gyorsítótárában a legújabb érték legyen. Az elosztott környezeti helyzetben egy dátumnak több másolata is létezhet.

Tegyük fel például, hogy egy A egész szám, amelyet 1-gyel kell növelni, a B fájlban található, és B fájl a merevlemezen található. A növelési művelet úgy folytatódik, hogy először kiad egy I/O műveletet, amely a lemezblokkotm anelyen A található a főmemóriába másolja. Ezt a műveletet követi az A másolása a gyorsítótárba és egy belső regiszterbe. Így A másolata több helyen megjelenik: merevlemez, főmemória, gyorsítótár, regiszter. A értéke el fog térni a különböző tárolószerkezetekben. Az A értéke csak akkor válik azonossá, ha az A új értéke a belső regiszterből visszakerül a merevlemezre.

## I/O alrendszer

Az operációs rendszer egyik célja, hogy elrejtse a hardvereszközök sajátosságait a felhasználók elől. Az I/O alrendszer felelős:

* I/O memóriakezelés, beleértve a pufferelést (adatok tárolása ideiglenesen az átvitel alatt)
* Gyorsítótárazás (az adatok egy részének tárolása gyorsabb tárolórendszerben a teljesítmény érdekében)
* Spooling (egy job kimenetének átfedése más jobok bemenetével)
* Általános eszköz-illesztő program interfész kezelése
* Illesztő programok telepítése meghatározott hardvereszközökhöz

## Védelem és biztonság

### Védelem

A folyamatokhoz való hozzáférést szabályzó bármely mechanizmus illetve, a felhasználókat az operációs rendszer által meghatározott erőforrásokhoz.

A rendszerek általában először megkülönböztetik a felhasználókat, hogy meghatározzák, kik mit tehetnek. Ezeket a következő azonosítókkal teszik meg:

* Felhasználói azonosítók, amelyek tartalmazzák a nevet és a kapcsolódási számot (felhasználónkként egy van belőle). A felhasználó azonosító ezután az adott felhasználó összes fájljához és folyamatához társítva hozzáférést biztosít.
* A csoportazonosító lehetővé teszi a felhasználók halmazának meghatározását és a vezérlők kezelését, majd társítását az egyes folyamatokhoz, fájlokhoz.
* A privilégium eszkaláció lehetővé teszi a felhasználó számára, hogy érvényes azonosítóra váltson több jogot.

### Biztonság

A rendszer védelme a belső és külső támadásokkal szemben. Hatalmas a választék, beleértve a szolgáltatásmegtagadást, férgeket, vírusokat, személyazonosságlopást, szolgáltatáslopást.

## Kernel adatstruktúrák

### Listák, veremek és sorok

A tömb egy egyszerű adatstruktúra amelyben minden elem közvetlenül elérhető. Míg a tömb minden eleme közvetlenül elérhető, a lista elemeit meghatározott sorrendben kell elérni. Azaz egy lista adatértékek gyűjteményét ábrázolja sorozatonként. A leggyakoribb módszer ennek a struktúrának a megvalósítása egy linkelt lista, amelyben az elemek egyhez kapcsolódnak egy másik. A linkelt listának több típusa van:

* Egy egyedileg összekapcsolt listában minden elem az utódjára mutat

A képen képernyőkép, fekete látható

Automatikusan generált leírás

* A duplán linkelt listában egy adott elem hivatkozhat elődjére vagy utódjára

A képen képernyőkép, billentyűzet, fogantyú, számítógép látható

Automatikusan generált leírás

* Egy körkörösen linkelt listában a lista utolsó eleme az elsőre vonatkozik nulla helyett

A képen képernyőkép, szöveg, Betűtípus, szám látható

Automatikusan generált leírás

A linkelt listák különböző méretű elemeket tartalmaznak, és lehetővé teszik a könnyű beillesztést és az elemek törlését. A listák néha közvetlenül a kernel algoritmusai használják.

A verem egy szekvenciálisan rendezett adatstruktúra, amely a legutolsó elsőt használja (LIFO) el az elemek hozzáadásához és eltávolításához, ami azt jelenti, hogy az utolsó elem a halomra helyezett elem az első eltávolítás. A beillesztés műveletei és az elemek veremből való eltávolítása push és pop néven ismert. Az operációs rendszer gyakran használ veremet függvény meghívásakor.

A várakozási sor egy szekvenciálisan rendezett adatstruktúra, amely a FIFO elvet használja. Abban a sorrendben távolítja el a tételeket ahogy bekerültek.

### Bináris keresőfa

A képen kör, óra, művészet látható

Automatikusan generált leírásA fa olyan adatstruktúra, amely az adatok hierarchikus ábrázolására használható. A fastruktúrában lévő adatértékek szülő-gyermek kapcsolatokon keresztül kapcsolódnak össze. Egy általános fában egy szülőnek korlátlan mennyiségű gyereke lehet.

Egy bináris fában egy szülőnek legfeljebb két gyermeke közötti sorrendet, amelyben a bal gyermek <= jobb gyermek. Amikor egy bináris keresési fában keresünk egy elemet, a legrosszabb eset teljesítménye O(n). A helyzet orvoslására létrehozhatunk egy kiegyensúlyozott bináris keresési fát O(lg n).

### Hash-függvény

A hash függvény adatot vesz bementként, numerikus műveletet hajt végre ezen az adatokon, és numerikus értéket ad vissza. Ez a numerikus érték ezután indexként használható egy táblázatba az adatok gyorselérése érdekében. Egyik lehetséges nehézsége az, hogy két bemenet ugyanazt a kimeneti értéket eredményezheti – vagyis ugyanarra a táblahelyre hivatkozhat, ez elkerülhető, ha van egy linkelt lista, ami az összes azonos hash értékkel rendelkező elemet tartalmazza.

A képen képernyőkép, szöveg, sor, Téglalap látható

Automatikusan generált leírásA hash függvény egyik felhasználási módja egy hash leképzés megvalósítása, amely hash függvény segítségével társítja (vagy leképzi) a [kulcs: érték] párokat. Tegyük fel például, hogy egy felhasználónév egy jelszóhoz van társítva. A jelszavas hitelesítés ezután a következőképpen zajlik: a felhasználó megadja a felhasználónevét és jelszavát. A hash funkcióit a rendszer alkalmazza a felhasználónévre, amely ezután a jelszó lekérésére szolgál. A visszakeresett jelszót ezután összehasonlítja a felhasználó által a hitelesítéshez megadott jelszóval.

### Bitmap

A képen képernyőkép, Téglalap, sor, mérőléc látható

Automatikusan generált leírásn bináris számjegyből álló karakterlánc, amely n elem állapotát jelzi. Például van több erőforrásunk és az egyes erőforrások elérhetőségét jelöli egy bináris szám: 0 az erőforrás elérhető, 1 az erőforrás nem elérhető. Egy bittérkép használható az egyes lemezblokkok elérhetőségének jelzésére.

# A képen képernyőkép, szöveg, diagram, szoftver látható Automatikusan generált leírásChapter 2 – Operációs rendszer struktúrák

## Operációs rendszer szolgáltatásai

Az operációs rendszer biztosítja azt a környezetet, amelyben a programok végrehajtásra kerülnek. Bizonyos szolgáltatásokat nyújt a programoknak és e programok felhasználóinak. Megnézhetjük, hogyan kapcsolódnak egymáshoz a szolgáltatások:

### Felhasználói felület

Szinte minden operációs rendszer rendelkezik felhasználói felülettel (UI). Ennek ez interfésznek többféle formája lehet. Az egyik a parancssori felület (CLI), amely szöveges parancsokat és azok beviteli módját (pl. egy billentyűzet a parancsok beírására adott formátumban meghatározott lehetőség). Egy másik kötegelt felület, amelyen parancsok és direktívák találhatók ezeknek a parancsoknak a vezérléséhez fájlokba kell írni, és ezeket végrehajtani. Leggyakrabban grafikus felhasználói felületet használnak (GUI). Itt az interfész egy ablakrendszer egy mutatóeszközzel az I/O irányítására, választani lehet menükből stb. Egyes rendszerek többféle felületet is biztosítanak.

### Programvégrehajtás

A rendszernek be kell tudnia tölteni egy programot a memória és a programfuttatásához. A programnak be kell tudni fejezni a végrehajtást, akár normálisan, akár hibát jelezve.

### I/O műveletek

Egy futó program I/O-t igényelhet, ami magában foglalhatja a fájl vagy I/O eszköz.

### Fájlrendszer-manipuláció

Nyilvánvaló, hogy a programoknak fájlokat és könyvtárakat kell olvasniuk és írniuk. Tudnak létrehozni, törölni fájlokat. Megkeresni egy adott fájlt.

### Kommunikáció

A kommunikáció megvalósítható megosztott memórián keresztül, amelyben két vagy több folyamat olvas és ír a memória megosztott részében, vagy üzenetet továbbit, amely előre meghatározott formátumú információcsomagot mozgat a folyamatok között.

### Hibaészlelés

Az operációs rendszernek minden hibatípus esetén meg kell tennie a megfelelő lépéseket a helyes és következetes számítási mód biztosítása érdekében. Néha nincs más választása, mint leállítani a rendszert. Máskor leállíthat egy hibát okozó folyamatot, vagy hibakódot küldhet vissza egy folyamatnak, hogy a folyamat észlelje és esetleg kijavítsa.

### Forráselosztás

Ha egyszerre több felhasználó vagy több job fut, akkor mindegyikhez erőforrást kell hozzárendelni. Az operációs rendszer sokféle erőforrást kezel.

### Könyvelés

Szeretnénk nyomon követni, hogy mely felhasználók mennyit és milyen számítógépes erőforrásokat használnak. Ez a nyilvántartás felhasználható könyvelésre (a felhasználók számlázására), vagy egyszerűen használati statisztikák összegyűjtésére.

### Védelem és biztonság

A védelem magában foglalja a rendszererőforrásokhoz való minden hozzáférés ellenőrzését. Szintén fontos a rendszer biztonsága a kívülállókkal szemben. Ez a biztonság azzal kezdődik, hogy minden felhasználótól megköveteli a hitelesítést

## A képen szöveg, képernyőkép, szoftver látható Automatikusan generált leírásRendszerhívások

Programozási felület az OS által nyújtott szolgáltatásokhoz. Általában magasszintű nyelven írják (C vagy C++). Többnyire magas szinten keresztül érhetők el a programok Alkalmazási Felület (API) a közvetlen rendszerhívás használata helyett. Rendszerhívási sorrend az egyik fájl tartalmának egy másik fájlba másolásához.

A rendszerek gyakran több ezer rendszerhívást hajtanak végre másodpercenként. Az alkalmazásfejlesztők általában egy alkalmazásnak megfelelően terveznek programokat programozási felület (API).

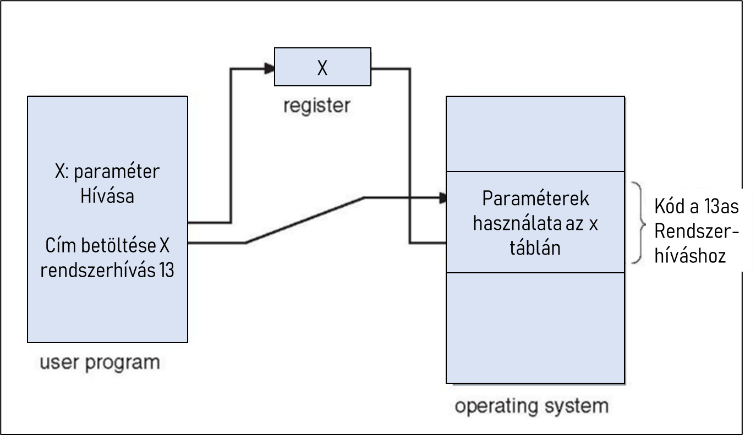
### A rendszerhívás implementációja

A képen szöveg, képernyőkép, diagram, Grafika látható

Automatikusan generált leírásA rendszerhívás interfész elfogja a függvényhívásokat az API-ban, és meghívja a szükséges rendszerhívásokat az operációs rendszeren belül. Általában minden rendszerhíváshoz egy szám tartozik, és a rendszerhívás interfész egy táblázatot tart fenn ezeknek a számoknak megfelelően indexelve. A rendszerhívás interfész ezután meghívja a tervezett rendszerhívást az operációs rendszer kernelében és visszaadja a rendszerhívás állapotát és az esetleges visszatérési értékeket.

### Rendszerhívási paraméter átadása

Gyakran több információra van szükség, mint egyszerűen a kíván személyazonosságára rendszerhívás. Az információ pontos típusa és mennyisége az operációs rendszertől függően változik és hívja. Három általános módszere van a paraméterek átadására az operációs rendszernek:

* Legegyszerűbb: Át adjuk a paramétert a regiszterekben. Bizonyos esetekben több paraméter is lehet, mint regiszter
* Blokkban vagy táblában, a memóriában tárolt paraméterek és a regiszterben paraméterként átadott blokk címe. Ez a Linux és a Solaris megközelítés
* Paraméterek, amelyet a program a verembe helyezett vagy tol, és az operációs rendszer kipattant a veremből. A blokk- és a veremmódszerek nem korlátozzák az átadott paraméterek számát vagy hosszát.

### Rendszerhívások típusai

* Folyamat létrehozása, folyamat befejezése
* Megszakítás
* Betöltés, végrehajtás
* Folyamatattribútumok beszerzése, folyamatattribútumok beállítása
* Várakozás
* Várakozási esemény, esemény jelzése
* Memória felszabadítása és lefoglalása
* Memória kiírása hiba esetén
* Hibák meghatározása, egylépéses végrehajtás
* Zárak a folyamatok közötti megosztott adatokhoz való hozzáférés kezeléséhez
* Fájlkezelés: létrehozás, törlés, megnyitás, bezárás, írás, olvasás, áthelyezés, fájlattribútumok lekérése és beállítása
* Eszközkezelés: Lekérés, feloldás, olvasás, írás, áthelyezés, attribútum lekérés, beállítás, eszközök logikai csatlakoztatása vagy leválasztása
* Információ karbantartása: Idő vagy dátum lekérése, beállítása; rendszeradatok beszerzése, beállítása; folyamat-, fájl- vagy eszközattribútumok lekérése és beállítása
* Kommunikáció: Kapcsolat létrehozása, törlése; Üzenet küldése, fogadása, ha az üzemet átadja a modellt a gazdagépnek név vagy folyamatnév; Megosztott memória modell létrehozása és hozzáférése a memóriához; állapotinformációk átvitele; Távoli eszközök csatlakoztatása és leválsztása
* Védelem: Az erőforrásokhoz való hozzáférés szabályozása; Engedélyek beszerzése és beállítása; Felhasználói hozzáférés engedélyezése és letiltása

# Chapter 3 – Folyamatok

A folyamat a munka egysége egy modern időmegosztási rendszerben. Fő feladata a felhasználói programok végrehajtása, valamint gondoskodni különféle rendszerfeladatatokról, amelyeket jobb a kernelen kívül hagyni. A rendszer tehát folyamatok gyűjteményéből áll.

## A folyamat

A képen képernyőkép, tervezés látható

Automatikusan generált leírásA batch rendszer a feladatokakt hatja végre, míg az időmegosztott rendszer felhasználói programokkal vagy feladatokkal rendelkezik. Például olyan beágyazott eszközön, amely nem támogatja a többfeladatos működést, előfordulhat, hogy az operációs rendszernek támogatnia kell saját belső programozott tevékenységeit, például a memóriakezelést. Sok tekintetben ezek a tevékenységek hasonlóak, ezért mindegyiket folyamatnak nevezzük.

* A folyamat egy futó program.
* A folyamat, több mint egy programkód.
* Tartalmazza az aktuális tevékenységet is, amelyet az utasításszámláló értéke és a processzor regisztereinek tartalma képvisel
* Magába foglalja a folyamatvermet is, amely ideiglenes adatokat tartalmaz
* Magába foglal egy adatszakaszt, amely globális változókat tartalmaz
* Tartalmazhat kupacot is, amely a folyamat futási ideje alatt dinamikusan lefoglalt memória

A program passzív entitás, amely a lemezen van tárolva (futtatható fájl), míg a folyamat aktív. A program akkor lesz folyamat, amikor a végrehajtható fájl betöltődik a memóriába. Egy program több folyamatból is állhat. Egy programot több felhasználó is futtathat egyszerre.

## A képen szöveg, kör, képernyőkép, Betűtípus látható Automatikusan generált leírásA folyamat állapotai

* Létrejön: A folyamat létrehozása
* Fut: Utasításokat hajt végre
* Várakozik: A folyamat valamilyen eseményre vár (pl. I/O befejezése)
* Kész: A folyamat a processzorhoz való hozzárendelésre vár
* Vége: A folyamat végrehajtása befejeződött

## Folyamatvezérlőblokk

Az operációs rendszerben minden folyamatot egy folyamatvezérlőblokk (PCB) képvisel – ezt feladatvezérlőblokknak is nevezik. Részei: folyamatállapota, utasításszámláló (a következő végrehajtási utasítás helye), folyamatazonosító, CPU regiszterek (az összes folyamatközpontú regiszter tartalma), Folyamatütemező (prioritások), memóriakezelő / limit (a folyamathoz lefoglalat memória), nyomkövetési dolgok (elhasznált CPU, indulás óta eltelt idő), I/O állapot (megnyitott fájlokról lista).

Röviden a PCB egyszerűen tárhelyként funkcionál, olyan adatoknak, amik folyamatonként változhatnak

|  |
| --- |
| Folyamtállapota |
| Folyamatazonosító |
| Utasításszámláló |
| Regiszterek |
| Memória limit |
| Lista a megnyitott fájlokról |
| **…** |

Másodpercenként 8-9 folyamatváltás. Multiprogramozottság ára → Idő

Folyamat állapotának elmentése → Másik folyamat betöltése → Megszakítás 🡪Másik folyamat végrehajtása → Elmentése → eredeti folyamat visszatöltése

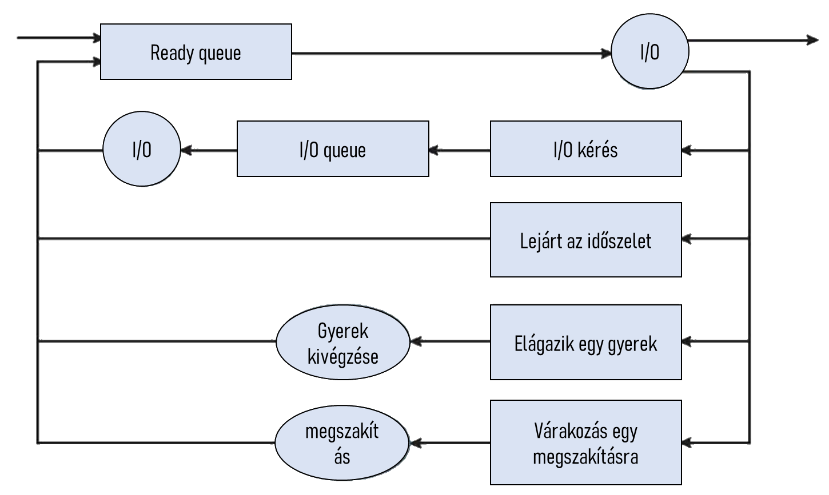
## Szálak

Eddig a PCB csak szekvenciális végrehajtásra volt képes, mert 1 utasításszámláló állt rendelkezésre. Most több „szálon futhatnak”, mert több az utasításszámláló

## Folyamathierarchia

* Gyökér-Szülő-Gyermek → Fa alakú → Gyökér +levelek
* Ősfolyamat → Elindítja az összes többi folyamatot
* 1 folyamatnak 1 szülőfolyamat, bármennyi gyerekfolyamat
* Fork → Folyamat elindít egy másik folyamatot
  + Megvárja a szülő folyamat a gyermek folyamat lefutását
* Ha a szülő folyamat leáll → Ki kell gyalulni az összes alatta lévő gyermek folyamatot
  + Orphan/árva folyamatok → Keresni kell új szülőt →Gyökérfolyamat → init → Időnként elindít egy wait rendszerhívást, az árva folyamatok lefutnak, vagy ha nem kigyalulja a megmaradt folyamatokat a kernel
  + Egész fának valahogy el kell tűnnie
* Ha a szülő nem foglalkozik azzal hogy a gyerek „kész van”, azaz nem hívja meg a waitet, a folyamatból Zombie folyamat lesz
  + Leadja az erőforrást, terminált a folyamat, de megmaradna a PCB blokkja

## Folyamatütemezés

Célja maximalizálni a gépidőt, a folyamatok közötti váltásokkal. E cél elérése érdekében a folyamatütemező választ egy elérhető folyamatot a program CPU-n történő végrehajtásához. A kernel listákat kezel, vagyis a folyamatokból fákat épít.

Ütemező sorok:

* Job queue: Az összes folyamat a rendszerben
* Ready queue: Az összes folyamat, ami készen áll a kivégzésre
* Device queue: A folyamatok, amik I/O eszközre várnak
* A folyamatok ezek között lépkednek

## Ütemezők

### Rövidtávú ütemező

Másnéven CPU ütemező, kiválasztja, hogy melyik folyamatot kell legközelebb végrehajtani, és lefoglalja a CPU-t. Néha ez az egyetlen ütemező a rendszerben. A rövidtávú ütemezőt gyakran használják. (Ezredmásodpercben mérik). Pl. laptop, telefon

### Hosszútávú ütemező

Másnéven feladatütemező, kiválasztja, hogy mely folyamatok kerüljenek készenléti sorba. A hosszútávú ütemezőt ritkán hívják meg. (Másodpercek, percek). A hosszútávú ütemező szabályozza a többprogramozás mértékét. A hosszútávú ütemező törekszik a jó folyamatkeverésre. A folyamatokat kétféleképpen írja le:

* I/O kötött folyamat, több időt tölt I/O-val, mint számításokkal, sok rövid CPU burst
* CPU-hoz kötött folyamat, több időt tölt számításokkal, néhány nagyon hosszú CPU burst

### Középtávú ütemező

Hozzáadható, ha a fokozat többszöröse és a programozást csökkenteni kell. Távolítsa el a folyamatot a memóriából, tárolja lemezen, hozza vissza a lemezről a végrehajtás folytatásához: csere

## Folyamatközi kommunikáció

A rendszeren belüli folyamatok lehetnek függetlenek vagy együttműködőek. Minden olyan folyamat, ami nem oszt meg adatokat más folyamatokkal az független.

Az együttműködési folyamat, más folyamatokat is befolyásolhat beleért az adatok megosztását is. Miért biztosítunk nekik környezetet az együttműködésre?

* Információ megosztás: Több felhasználót is érdekelhet ugyanaz az infó
* Számítási gyorsaság: Feladatok részfeladatokra való bontása
* Modularitás: Egyszerre több felhasználó is dolgozhat a feladaton.
* Kényelem

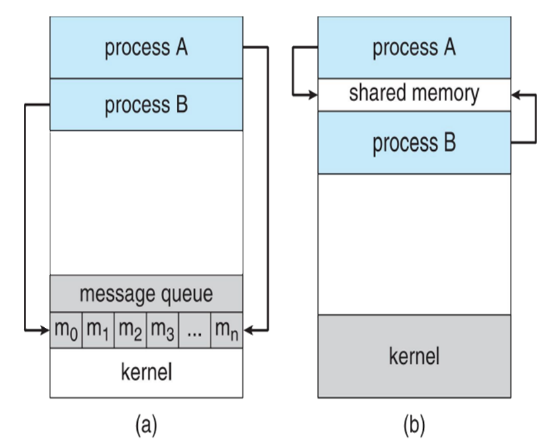
#### Producer-Consumer Process

Az együttműködő folyamatok paradigmája, a termelői folyamat információt állít elő, amelyet a fogyasztó folyamat elfogyaszt. A korlátlan buffer nem szab gyakorlat korlátot a buffer méretének. A bounded-buffer feltételezi, hogy van egy rögzített buffer méret.

#### IPC

Az együttműködési folyamatokhoz folyamatközi kommunikációra (IPC) van szükség. Az IPC-nek két modellje van:

##### Megosztott memória(b)

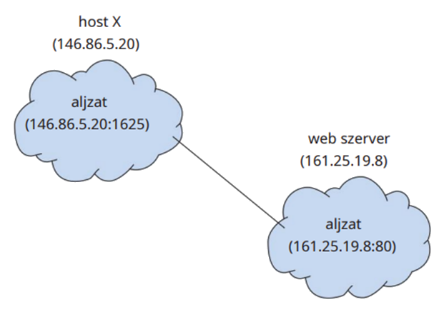
Létrejön egy memória terület, amelyet az együttműködő folyamatok osztanak meg. Ez a kommunikáció felhasználói folyamatok irányítása alatt van, nem az OS irányítása alatt. Fő feladata egy mechanizmus létrehozása arra, hogy felhasználói folyamatok tudjanak szinkronizálni egymás között, amikor osztott memóriát használnak.

##### Üzenetátadás (a)

A kommunikáció üzenetváltásban történik. Osztott változók segítségével kommunikálnak. Két műveletet biztosít: üzenetküldés, és fogadás. Az üzenet mérete lehet fix vagy változó. Ha két folyamat kommunikálni akar egymással, akkor létre kell hozni közöttük a kommunikációs kapcsolatot, üzenet csere segítségével. Problémák:

* Hogyan jönnek létre a kapcsolatok?
  + Közvetlen vagy közvetett kommunikáció
    - Közvetlen kommunikáció esetén minden kommunikálni akaró folyamatnak kifejezetten meg kell nevezni a kommunikáció címzettjét vagy feladóját. Ebben a sémában a kapcsolat automatikusan létrejön minden kommunikálni kívánó folyamatpár között. Egy hivatkozás pontosan két folyamathoz van társítva. Minden folyamatpár között pontosan egy kapcsolat van.
    - Közvetett kommunikáció esetén az üzeneteket postafiókba vagy portokba küldik és fogadják. Minden postafiók egyedi azonosítóval rendelkezik. A postafiók tulajdonosa lehet egy folyamat vagy az operációs rendszer. A kapcsolat csak akkor jön létre, ha a folyamatok közös postafiókon osztoznak. Egy hivatkozás sok folyamathoz kapcsolódhat. Minden folyamatpár több folyamaton osztozhat. A lik lehet egy vagy Kétirányú
  + Szinkron vagy aszinkron kommunikáció
    - A blokkolást szinkronnak tekinthetjük. Küldést (A küldő blokkolva van az üzenet megérkezéséig) és fogadást (a vevő blokkolva van az üzenet megjelenítéséig) is tud blokkolni.
    - A nem blokkolást aszinkronnak tekinthetjük. A feladó elküldi az üzenetet és tudja folytatni. A vevő fogadja a null üzenetet is
  + Automatikus vagy explicit pufferelés
    - A linkhez csatolt üzenetek háromféleképpen valósulhatnak meg
      * Nulla kapacitás: nincs üzenet, a feladónak várnia kell a címzettre
      * Korlátozott kapacitás: n üzenet véges hossza, a feladónak várnia kell, ha a link megtelik
      * Korlátlan kapacitás: Végtelenhosszúság, a feladó soha nem vár

## Socket – a kommunikáció végpontja

IP-cím és port összefűzése – az üzenetcsomag elején szereplő szám (A port egyszerűen egy szám), amely megkülönbözteti a hálózati szolgáltatásokat a gazdagépen. Kell egy nyelv, amiben le lehet írni a végződéseket. Az aljzatot egy portszámmal összefűzött IP- cím azonosítja . Általában a socketek kliens-szerver architektúrát használnak. A szerver egy megadott portra figyelve várja a bejövő kliens kéréseket. Amint egy kérés érkezik, a szerver elfogadja a kapcsolatot a kliens socketről a kapcsolat befejezéséhez. A meghatározott szolgáltatásokat (például telnet, FTP és HTTP) megvalósító szerverek a jól ismert portokra figyelnek (a telnet szerver a 23-as portot figyeli, az FTP- szerver a 21-es portot, a web- vagy HTTP- szerver pedig a 80-as portotfigyeli) . Minden 1024 alatti port jól ismertnek tekinthető ; szabványos szolgáltatásokmegvalósítására használhatjuk őket.Az IPC- üzenetekkel ellentétben az RPC- kommunikáció során kicserélt üzenetek jól strukturáltak, így már nem csupán adatcsomagok. Minden üzenet egy RPC démonnak van címezve, amely a távoli rendszer egyik portját figyeli, és mindegyik tartalmaz egy azonosítót, amely meghatározza a végrehajtandó függvényt és az adott függvénynek átadandó paramétereket. A funkció ezután a kérés szerint végrehajtódik, és minden kimenet külön üzenetben visszaküldésre kerül a kérelmezőnek.

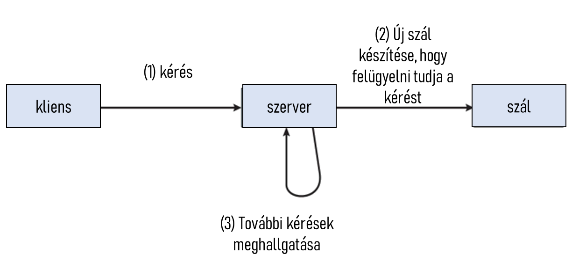
## Csövek

A cső csatornaként működik, amely lehetővé teszi két folyamat kommunikációját. Egy folyamat standard outputját összedrótozza egy folyamat standard inputjával. A csövek voltak az egyik első IPC- mechanizmus a korai UNIX rendszerekben. Jellemzően a folyamatok egymás közötti kommunikációjának egyik egyszerűbb módját biztosítják, bár vannak korlátaik is.

* Közönséges csövek: az azt létrehozó folyamaton kívülről nem érhetőek el. Általában egy szülő folyamat létrehoz egy csővezetéket, és azt használja az általa létrehozott gyermekfolyamattal kapcsolatban
* Nevesített csövek: szülő-gyermek kapcsolat nélkül is elérhetőek

# Chapter 4 – Szálak

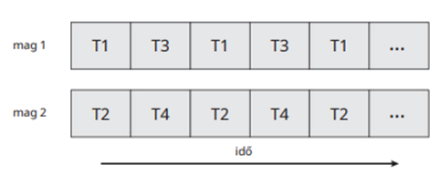
A szála CPU kihasználtságának alapegysége: tartalmaz egy szálazonosítót, egy utasításszámlálót, egy regiszterkészletet és egy vermet. Megoszthatja ugyanazon folyamathoz tartozó más szálakkal a kódszakaszt. Egy hagyományos folyamat egyetlen szálon irányítható. A legtöbb modern alkalmazás többszálú. A szálak az alkalmazáson belül futnak. Az alkalmazással több feladat külön szálon keresztül valósítható meg. Könnyebb létrehozni mint egy folyamatot. Leegyszerűsítheti a kódot és növeli a hatékonyságot. A kernelek általában többszálúak. A szálak használatának köszönhetően töredékére esik az elvesztegetett gépidő

Ha a webszerver folyamat többszálú, a szerver külön szálat hoz létre, amely figyeli a kliens kérését. Amikor kérés érkezik, ahelyett, hogy másik folyamatot hoz létre, a szerver létrehoz egy szálat a kérés kiszolgálásához és a figyelés folytatásához további kéréseket.

## Előnyei

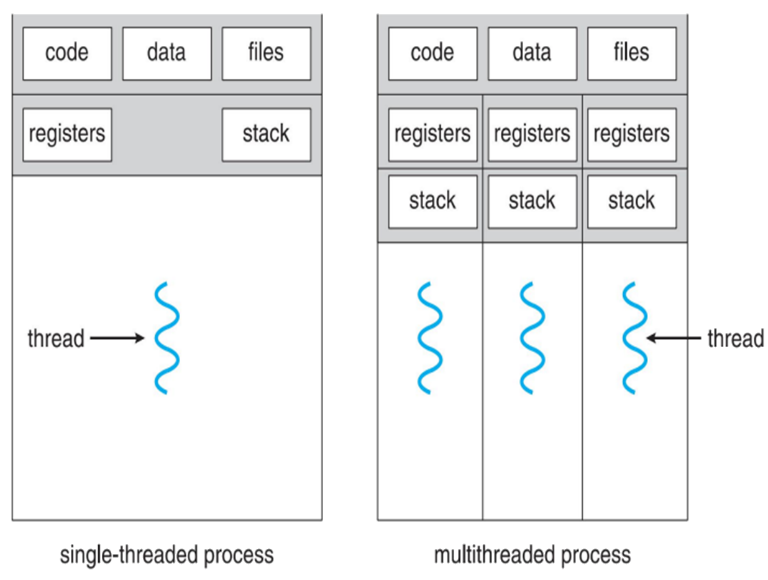
* Reszponzivitás: Lehetővé teheti a folyamatos végrehajtást, ha a folyamat egy része blokkolva van, különösen fontos a felhasználói felületeknél.
* Erőforrásmegosztás: A szálak megosztják a folyamat erőforrásait, könnyebben, mint a megosztott memória vagy az üzenettovábbítás
* Gazdaságos: Olcsóbb, mint a folyamat létrehozása, a szálváltás alacsonyabb rezsiköltséggel jár, mint a kontextusváltás
* Skálázhatóság: A folyamat kihasználhatja a többprocesszoros architektúrák előnyeit

## Multicore Programming

A többszálú programozás mechanizmust biztosít a hatékonyabb használatához ezekből a több számítási magból és jobb párhuzamosságból. Egy többmagos rendszerben a párhuzamosság azt jelenti, hogy a szálak párhuzamosan futhatnak, mert a rendszer hozzá tud rendelni minden maghoz külön menetet. Vagyis egy rendszer egynél több feladatot is képes ellátni egyidejűleg. A párhuzamosság több feladat előrehaladását is támogathatja.

Nem lineáris a teljesítmény növekedésében 2 mag != 2 külön processzorral.

### Programozói kihívások

1. Feladatok meghatározása: Alkalmazások vizsgálata, terület megtalálása, ami párhuzamos feladatokra oszthatók fel.
2. Egyensúly: A feladatok egyenlő értékű munkát végezzenek
3. Adatfelosztás: A feladatok által elért és kezelt adatokat is fel kell osztani, hogy különálló magokon futhassanak
4. Adatfüggőség: A feladatok által elért adatokat meg kell vizsgálni két vagy több feladat közötti függőség szempontjából
5. Tesztelés és hibakeresés: Ha egy program párhuzamosan fut több magon, sok különböző végrehajtási útvonal lehetséges. Az ilyen párhuzamos programok tesztelése és hibakeresése nehezebb.

### Párhuzamosság típusai

A párhuzamosságnak két típusa van: az adatok párhuzamossága és a feladat-párhuzamossága.

* Adatpárhuzamosság: Arra összpontosít, hogy ugyanazon adatok részhalmazait elosztja több számítási mag között, és ugyanazt a műveletet hajtsa végre mindegyik
* Feladat párhuzamosság: Szálak elosztása a magok között egyedi műveletet végző szál

Ahogy a szálak száma növekszik, úgy nő a támogatottsága is.

### Felhasználói és kernelszálak

#### Felhasználói szálak

A kezelést felhasználói szintű szálkönyvtár végzi. Három elsődleges szálkönyvtár van: POSIX P-szálak, Windows szálak, Java szálak

#### Kernelszálak

A kernel támogatja. Gyakorlatileg minden általános célú operációsrendszer, beleértve: Windows, Solaris, Linux, MacOsX

## Többszálú modellek

|  |  |  |
| --- | --- | --- |
| Many to One | One to One | Many to Many |
| Sok felhasználói szintű szál egyetlen kernelszálra van leképezve  Egy szál blokkolása az összes blokkolását okozza  Előfordulhat, hogy több szál nem fut párhuzamosan a Multicore rendszeren, mert egyszerre csak egy lehet a kernelben | Minden felhasználói szintű szál leképződik a kernelszálra  Felhasználói szintű szál létrehozása létrehoz magával egy kernelszálat  Több egyidejűség  A folyamatonkénti szálak száma néha korlátozott a többletterhelés miatt | Lehetővé teszi sok felhasználói szintű szál sok kernelszálhoz való leképzését  Lehetővé teszi az operációs rendszer számára, hogy elegendő számú kernelszálat hozzon létre- |
| Solaris Green Threads  GNU hordozható szálak | Windows  Linux  Solaris 9 és újabb | Solaris 9-es verzió előtti  Windows ThreadFiberrel csomag |
|  |  |  |

A szálkönyvtárak API-t biztosít a programozó számára a szálak létrehozásához és kezeléséhez. A megvalósítás két elsődleges módja:

* A könyvtár teljes mértékben a felhasználói térben van
* Az operációs rendszer által támogatott kernel szintű könyvtár

## Jelkezelés

Az UNIX rendszerekben a jeleket arra használják, hogy értesítsenek egy folyamatot egy adott eseményről. A jelek feldolgozására jelkezelőt használnak:

* A jelet egy adott esemény generálja
* A jel egy folyamatba kerül
* A jelet a két jelkezelő egyike kezeli: Alapértelmezett vagy a felhasználó által meghatározott

Minden jelnek van egy alapértelmezett kezelője, amely a kernel a jelek kezelésekor fut. A felhasználó által definiált jelkezelő felülbírálhatja az alapértelmezést. Egyszálú jelet továbbít a folyamathoz.

### Thread Local Storage

Lehetőséget ad minden szálnak, hogy meglegyen a saját adatpéldánya/memóriája. Látható függvényhívások között is, nem olyan mint egy helyi változó. Hasonlít a static változóhoz (Fix a memóriacíme). A Thread vezérlés hasonlít a folyamatokhoz

# Chapter 5 – folyamat szinkronizáció

A folyamatok párhuzamosan is végrehajthatóak. Bármikor megszakítható, részben befejezve a végrehajtást. A megosztott adatokhoz való egyidejű hozzáférés az adatok következetlenségét eredményezheti. Az adatok konzisztenciájának fenntartásához olyan mechanizmusok szükségesek, amelyek biztosítják az együttműködő folyamatok szabályos lebonyolítását.

Tegyük fel, hogy olyan megoldást akartunk nyújtani a fogyasztó-termelő problémára, amely minden puffert kitölt. Ezt úgy tehetjük meg, hogy van egy egész számláló, amely nyomon követi a megtelt pufferek számát. Kezdetben a számláló 0-ra van állítva. Ezt a gyártó növeli, miután új puffert állított elő, és a fogyasztó csökkenti, miután elfogyasztott egy puffert.

## Versenyhelyzet

Példa adat inkonzisztenciára: Mivel a producer és consumer bármikor megszakítható, ezért hiba lehet, ahogy itt is, keresztül húzza egymást a producer és consumer. Ha mondjuk így egy tömb x.dik elemét keressük, és egy 10 elemű tömbnél a counter 15, az még rosszabb, mert nem vesszük észre a hibát, és kiírhat valami memória szemetet.

## A képen szöveg, képernyőkép, Betűtípus, tervezés látható Automatikusan generált leírásKritikus szakasz probléma

Minden folyamatnak van egy kritikus kódrészlete. A folyamat megváltoztathatja a gyakori változókat, frissítheti a táblát, írhat fájlt, stb. Amikor az egyik folyamat a kritikus szakaszban van, más nem lehet benne. A kritikus szakaszprobléma megoldására protokollt kell tervezni.

1. Kölcsönös kizárás, egyszerre csak egy folyamat lehet kritikus szakaszban
2. Haladás: Ne lehessen egy folyamat kritikus szakaszát végtelenségig halasztani. Ha nincs egy folyamat se kritikus szakaszban, akkor kell egy ami abba akar menni
3. Ütemezett várakozás: Bármely program véges időn belül be tudjon lépni a kritikus szakaszába. Van egy limit egy folyamat mennyiszer kérheti a kritikus szakaszba való belépést.

Minden folyamatnak engedélyt kell kérnie a kritikus szakaszba való belépéshez a belépési szakaszban, követheti a kritikus szakaszt a kilépési szekcióval, majd a maradék szakaszt.

### Kritikus szakasz kezelése

A kernel módban történik. Kétfajta megközelítése van.

#### A kernel nem preemptív

Ha a folyamat Kernel módban van, nem lehet tőle elvenni a gépidőt. Versenyhelyzet lényegében nem fordulhat elő. Ilyen nem nagyon van

#### A kernel preemptív

Nem számít milyen módban van, ha ütemező el akarja elveszi tőle a gépidőt. Ilyen az összes modern operációs rendszer. Gyorsabb reszponzívabb, de bármikor megszakadhat.

#### Peterson megoldása

Load és Store atomi utasítások, nem lehet megszakítani. Két közös változó van a folyamatok között

* Int turn
  + Megmondja kinek a „köre” a kritikus szakaszba lépni
* Boolean flag[i] →alapból hamis
  + Megmondja egy folyamat készen áll-e belépni a kritikus szakaszba
  + Flag[i]=true → I.-dik folyamat készen áll

Teljesül a Kölcsönös kizárás, Haladás, Ütemezett várakozás.

## Hardveres szinkronizáció

### Locking / Zárolás

Sok rendszer hardveres támogatást nyújt a megvalósításhoz, van kritikus szakasz kódja. Az alábbi megoldások mindegyike a zárolás ötletén alapul. A kritikus régiók védelme zárakkal. Folyamata: Belépés a kritikusszakaszba🡪Megkéri a zárat 🡪 Végez 🡪 Elengedi a zárat

* Egyprocesszor esetén leállíthatja a megszakításokat. A jelenleg futó kód előfoglalás nélkül futna
* Többprocesszoros rendszereken általában túlságosan nem hatékony. Az ezt használó operációs rendszerek nem széles körben méretezhetők.

A modern gépek speciális atomi hardver utasításokat adnak. Atomi=nem megszakítható. Vagy tesztelni kell a memóriaszót és a beállított értékeket vagy cserélni kell őket. Leállítja a CPU ütemezőt

### Compare and Swap / Összehasonlítás és csere

Ugyanaz, mint a Lock, csak van benne egy if, hogy olyan állapotban van-e a zár, mint azt vártuk. 2.folyamatra biztos működik.

### Waiting / Kölcsönös kizárás

Megoldás több folyamatra. Van egy segédgörbe hozzá. Folyamata: Egy folyamat van a kritikus szakaszban 🡪 letárolja az összes adatát 🡪 Exit section 🡪 Végigmegy folyamatosan egy tömbön, amiben azt tároljuk melyik folyamat várakozik éppen hogy kritikus szakaszba lépjen. Ha van olyan aki be akar lépni akkor ő jön, ha nincs akkor kinyitja a zárat és keres valakit aki be akar lépni.

## Szoftveres szinkronizáció

### Mutex lock

A legegyszerűbb a mutex zár. Megvédi a kritikus szakaszt úgy, hogy először szerez egy zárat, majd felengedi azt. Lokális változó jelöli azt, hogy a zárolás elérhető vagy nem. A kritikus szakasz kezdete az acquire lock és vége a release lock ezek atomi jellegű hívások. Ez a megoldás sok várakozást igényel, ezt a zárat ezért nevezik spinlocknak.

### Szemafor

Szinkronizáló eszköz, amely kifinomultabb módokat biztosít, mint a Mutex zárak, a tevékenységek szinkronizálásához. Csak két oszthatatlan atomi művelettel érhetők el: wait (meg akarja várni míg erőforrást kap) és signal (le akarja adni az erőforrást). Egy s egész számban látjuk, hogy hány folyamat lehet egyszerre a kritikus szakaszban. Ha 0, akkor maximumon van a jelenleg futó kritikus folyamatok száma. Kétféle szemafort használhatunk:

1. Számláló szemafor: az egész szám korlátlan érték felett mozoghat
2. Bináris szemafor: Egész szám csak 0 és 1 lehet, ezáltal ugyanaz, mint a mutex zár.

Különféle szinkronizálási problémákat tud megoldani. Ehhez garantálni kell, hogy egy folyamat nem hajthatja végre egyszerre a wait-et és a signalt-t, amik a kritikus szakaszban vannak. Így növekedhet a várakozási idő, az alkalmazások pedig sok időt tölthetnek a kritikus szakaszban.

Minden szemafornak van egy waiting sora, ez tartalmaz egy értéket és egy mutatót a sor következő rekordjára. A block művelet a folyamatot ami meghívta a megfelelő waiting queue-ba teszi. A wakeup művelet kivesz egy folyamatot a waiting queue-ból és beteszi readybe.

### Deadlock / Holdpont és Starvation / Éhezés

#### Holtpont

Kettő vagy több folyamat határozatlan ideig vár egy eseményre, ami az egyik másik várakozó folyamat által keletkezhetne, tehát nem jutunk sehova

#### Starvation

Egy folyamat soha nem távolítható el abból a szemaforsorból, amelyben van felfüggesztett.

#### Prioritás megfordítása

Ütemezési probléma, amikor az alacsonyabb prioritású folyamat zárolást tartalmaz, a magasabb prioritású folyamatokhoz. Erre megoldás a prioritás-öröklési protokoll

## A szinkronizálás klasszikus problémái

### Bounded buffer

### Readers Writers Problem

Ha egy adatbázist több párhuzamos folyamat között kell megosztani, akkor a folyamat olvashatja vagy frissítheti. Ha két olvasó egyszerre fér hozzá a megosztott adatokhoz, akkor nem lesz káros hatás, viszont ha többen akarják egyszerre frissíteni akkor igen.

Annak érdekében, hogy ezek a nehézségek ne merüljenek fel, megköveteljük, hogy az írók kizárólagos hozzáféréssel rendelkezzenek a megosztott adatbázishoz, miközben az adatbázisba írnak

### Dining Philosphers Problem

Tekintsünk öt filozófust, akik gondolkodással és evéssel töltik az életüket. A filozófusok egy kör alakú asztalon osztoznak, amelyet öt szék veszi körül, mindegyik egy filozófusé. Az asztal közepén egy tál rizs található, az asztalon pedig öt pálcika van megterítve (5.13. ábra). Amikor egy filozófus gondolkodik, nem érintkezik kollégáival. Időről időre egy filozófus megéhezik, és megpróbálja felvenni a hozzá legközelebb eső két pálcikát (azt a pálcikát, amely a közte és a jobb és bal szomszédja között van). Egy filozófus egyszerre csak egy pálcikát vehet fel. Nyilvánvalóan nem tud felvenni egy pálcikát, amely már a szomszéd kezében van. Amikor egy éhes filozófusnak egyszerre van a két pálcikája, úgy eszik, hogy nem engedi el a pálcikát. Amikor befejezi az evést, leteszi mindkét pálcikát, és újra gondolkodni kezd.

Az étkező-filozófusok problémáját klasszikus szinkronizációs problémának tekintik, sem gyakorlati jelentősége miatt, sem azért, mert az informatikusok nem szeretik a filozófusokat, hanem azért, mert a párhuzamosság-szabályozási problémák nagy csoportjának példája. Ez egy egyszerű reprezentációja annak az igénynek, hogy több erőforrást kell több folyamat között holtpont- és éhezésmentesen allokálni. Az egyik egyszerű megoldás az, hogy minden pálcikát szemaforral ábrázolunk. Egy filozófus úgy próbálja megragadni a pálcikát, hogy egy wait() műveletet hajt végre azon a szemaforon. A megfelelő szemaforokon a signal() művelet végrehajtásával elengedi a pálcikáját .

## Monitorok

Egy magasszintű absztrakció lehetővé teszi kényelmesen és hatékonyan a folyamatok szinkronizációját. Absztrakt adattípus, tehát a belső változókat csak a belső kód érheti el, úgy kell kezelni mint egy classt. Egyszerre csak egy folyamat lehet aktív a monitoron

# Chapter 6 – CPU ütemezés

A CPU ütemezés a multiprogramozott operációs rendszerek alapja. A CPU folyamatok közötti váltással az operációs rendszer termelékenyebbé teheti a számítógépet. A CPU a maximális kihasználtságot a multiprogramozással kapott.

A CPU – I/O sorozatfelvételi ciklus: A folyamat végrehajtása egy ciklusból áll CPU végrehajtás és I/O várakozás 🡪 CPU burst majd I/O burst.

Az első 8 miliszekundumban van a folyamatok legintenzívebb része, utána csak vár. Erre a megoldás az, hogy 8ms után másik folyamat indul. Lényegében ez a CPU ütemezés. A rövidtávú ütemező kiválaszt a készenléti sorban lévő folyamatok közül, és az egyiket a CPU-hoz rendeli, ez a sor többféleképpen is megtörténhet.

A folyamatoknak 4 állapota lehetséges:

1. Amikor egy futó folyamat várakozó állapotba vált (például I/O kérés vagy a wait() atomi függvény használata)
2. Amikor egy folyamat futó állapotból kész állapotba vált (pl. megszakítás esetén)
3. Amikor egy folyamat a várakozási állapotból kész állapotba vált (pl. I/O befejezésekor)
4. Amikor egy folyamat befejeződik

Ha az ütemezés csak az 1. és 4. körülmények között történik azt mondjuk, hogy az ütemezési séma nem megelőző vagy együttműködő. Ellenkező esetben megelőző jellegű. Nem megelőző ütemezés esetén, ha a CPU-t lefoglalták egy folyamathoz, a folyamat megtartja a CPU-t mindaddig, amíg fel nem engedi a CPU-t akár leállással, akár várakozási állapotba váltással.

Ma minden ütemező preemptív tehát kapsz gépidőt, de bármikor elvehető. A Windows 95-nél váltott a Microsoft preemptívre. Ebben az esetben nagyon fontos a reszponzivitás. Kell valami ami elvégzi a váltást két folyamat között. Ez lehet a diszpécser modul

* A diszpécsermodul a CPU vezérlését adja a rövidtávú ütemező által kiválasztott folyamathoz, ez magába foglalja a kontextusváltást, a felhasználói módba váltást, és azt ugrást a megfelelő helyre a felhasználói programban a program újraindításához
* A diszpécsernek a lehető leggyorsabbnak kell lennie, mivel minden folyamatváltáskor meghívódik. Azt a folyamatot, amely alatt a diszpécser leállít egy folyamatot és elindít egy másikat elküldési késleltetésnek (Dispatch Latency) nevezzük.

## Ütemezés szempontjai

A különböző CPU ütemezési algoritmikusok eltérő tulajdonságokkal rendelkeznek, és egy adott algoritmus kiválasztása előnyben részesítheti a folyamatok egyik osztályát a másikkal szemben. A tulajdonságok összehasonlításával lényeges különbséget tudunk tenni közöttük.

* Processzor kihasztnáltság: A CPU-t a legelfoglaltabban kell tartani. Valós rendszerekben 40 és 90% között kell tartani
* Kibocsátó képesség: A munka egyik mércéje az időegységenként végrehajtott folyamatok száma. Hosszú folyamatok esetén ez a sebesség óránként egy folyamat lehet; rövid folyamat esetén másodpercenként 10 folyamat is lehet
* Idő: A folyamat kezdetétől a befejezésig eltelt idő az átfutási idő. A várakozási idő az az idő amíg a folyamat a várakozási sorban várakozik.

A legjobb CPU kihasználtságának és az átviteli sebességnek a maximalizálása és a várakozási idő, valamint az átfutási idő minimalizálása.

## Ütemezési algoritmusok

### First Come First Served – FCFS

Ez a legegyszerűbb CPU ütemezési algoritmus, ami először érkezik az van először végrehajtva, tehát érkezési sorrend van. FCFS ütemezés esetén felléphet a konvoj hatás, ha egy hosszú folyamat van egy rövid folyamat előtt. Tehát megnő a rövid folyamat várakozási ideje.

### Shortest Job First – SJF

Ez a legrövidebb feladatot rakja az első helyre. Ezáltal minimumra csökken a várakozási idő. Viszont itt is fenn áll a konvoj hatás lehetősége, hiszen egy hosszú feladatot megelőzhet sok picike. A CPU burst hosszát becsülgeti

### Priority Scheduling – PS

Minden folyamathoz prioritás tartozik, és a folyamathoz hozzárendelt CPU-k a legmagasabb prioritásúak. Az egyenlő prioritással rendelkező folyamatok esetén FCFS sorrend történik. A belsőleg meghatározott prioritások valamilyen mérhető mennyiséget használnak fel a folyamat prioritásának kiszámításához (pl. memória igény, megnyitott fájlok száma, CPU burst aránya). A külső prioritásokat az operációs rendszeren kívüli kritériumok határozzák meg, mint például a folyamat fontossága, a számítógép használatért fizetett forrás típusa és összege.

Az elsőbbségi ütemezési algoritmus egyik fő problémája a határozatlan ideig tartó blokkolás (Starvation) vagy éhezés. A futásra kész, de a CPU-ra váró folyamat blokkoltnak tekinthető. Az alacsony prioritású folyamatok határozatlan ideig várakozhatnak. Erre megoldás az Aging, amíg nem jutnak gépidőhöz addig növekszik a prioritásuk

### Round Robin

Kifejezetten időmegosztó rendszerekhez tervezték, tehát PC-khez tökéletes. Meg van határozva egy kis időegység, amelyet időkvantumnak vagy időszeletnek neveznek. Az időkvantum hossza általában 10-100 milliszekundum. A készenléti sort körkörös sorként kezeli a rendszer. A CPU ütemező körbejárja a készenléti sort, és mindegyikhez hozzárendeli a CPU-t legfeljebb 1 időkvantum erejéig.

### Multilevel queue (MQ)

|  |  |  |
| --- | --- | --- |
|  | Előtér | Háttér |
| Ütemezése | RR | FCFS |
| Prioritás | Előtérben lévő folyamatok  abszolút prioritás a háttérben  futó folyamatokkal szemben | |
| CPU idő | 80% | 20% |

A folyamatok különböző csoportokba oszthatóak. A készenléti sor külön van particionálva előtérben lévő folyamatok és háttérben lévő folyamatok. Egy adott folyamat csak 1-ben lehet egyszerre.

### Multilevel Feedback Queue (MFQ)

A folyamatok véglegesen hozzá vannak rendelve egy sorhoz, amikor belépnek a rendszerbe. Ennek az ütemezésnek az előnye, hogy alacsony az ütemezési többletköltség, de rugalmatlan. Ezzel szemben ebben az ütemezésben a többszintű visszacsatolás lehetővé teszi, hogy a folyamatok mozogjanak a sorok között. Ezt a következő paraméterek határozzák meg:

* Sorok száma
* Ütemezési algoritmusok minden egyes sorhoz külön
* A folyamat frissítésének időpontjának meghatározása
* Egy folyamat lefokozásának időpontjának meghatározása
* Módszer, amellyel megállapítható, hogy egy folyamat melyik sorba kerüljön.

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírásA készenléti sorba belépő folyamat a 0. sorba kerül. A 0. sorban lévő folyamat 8 ezredmásodperces időkvantumot kap. Ha ezen időn belül nem fejeződik be, akkor az 1. sor végére kerül. Ha a 0. sor üres, az 1. sor elején lévő folyamat 16 ezredmásodperces kvantumot kap. Ha nem fejeződik be, akkor a rendszer megelőzi és a 2. sorba kerül. A 2. sorban lévő folyamatok FCFS alapon futnak, de csak akkor futnak, ha a 0 és 1. sor üres.

## Szálütemezés

A felhasználói szintű és a kernel szintű szálak megkülönböztetése. Ha a szálak támogatottak, a szálak ütemezve vannak, nem a folyamatok. Az ütemezés óta folyamat-tartalom (PCS) néven ismert folyamatban van a verseny. Általában a programozó által beállított prioritás van. Az elérhető CPU-ra ütemezett kernelszál a rendszertartalom-tartomány (SCS) verseny a rendszerben lévő összes szál között

Processzor affinitiás → Mennyire „ragaszkodjon” egy folyamat a processzorhoz amin fut. Általában folyamatoknál ez az érték nagy. Hogy ne tegye át másik procira a folyamatot, ezzel időt veszítve

* Soft affinitás
* Hard affinitás

Folyamat egy procin belül szeretne maradni de ez nem mindig megoldható

### Processing típusok

* Homogén processzorok→ Egy multiprocesszoron belül
* Assymetric multiprocessing
  + Master CPU-k és Slave CPU-k
* Symmetric Multiprocessing – SMP
  + CPU-k egyenrangúak

### Load Balancing

* SMP esetén összes processzornak kell „adni feladatot”, hogy hatékony legyen
* Load Balancing → Megpróbálja elosztani a terhet CPU-k között
* Pull migration
  + Ha vannak vegetatív magok/processzorok, áthúzza a terhelt prociról
  + Békés
* Push migration
  + Terhelt processzorról letolja a terhelést
  + Nem a legbékésebb

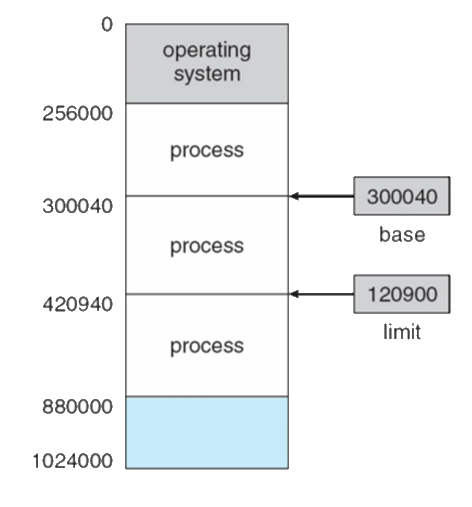
### Multicore Processzorok

Egy fizikai chipen belül több processzor. Nem olyan hatékony 2 mag mint 2 processzor. Elveszted a plusz Cache-t. Az ALU-t → Aritmetikai és Logikai egység. De mindenképpen javulás egy maghoz képest, viszont közel nem lineáris a javulás

### Valósidejű Processzorütemezés

X időn belül kell valami response/feedback a folyamattól. Nincs olyan hogy szimplán „lefagy”. Ha nem készül el időn belül, hibának veszi, elkezdi kiértékelni, javítani 🡪 Hibakezelő. Ez PC-kre egyáltalán nem jellemző

# Chapter 7 – Memóriakezelés

A memória bájtok nagy tömbjéből áll, mindegyiknek saját címe van. A CPU programszámláló értékének megfelelően lekéri a memóriából az utasításokat. Ezek az utasítások a további betöltést és tárolást okozhatnak az adott memóriacímekből. A főmemória és a regiszterek csak a CPU által közvetlenül elérhető tárolóeszközök. A programnak a storageból a memóriába kell kerülnie hogy le tudjon futni. A memory unit csak címek sokaságát, illetve olvasás írás requestet lát. A gyorsítótár a memória és a CPU között van. A memóriát védeni kell hogy megfelelően tudjon működni. A Base és a limit meghatározza a memóriacímek tartományát, ami elérhető az usernek. A CPU-nak ellenőriznie kell, hogy az user módban generált memória eléréshez joga van-e az usernek.

A memória területek elkülönítéséhez meg kell határoznunk a jogi címek tartományát, amelyekhez a folyamat hozzáférhet, és biztosítani kell, hogy a folyamat csak ezekhez a jogcímekhez férjen hozzá. Ezt a védelmet két regiszter használatával tudjuk elérni, általában egy bázis és egy limit.

Az alapregiszter tartalmazza a legkisebb fizikai memóriacímet; a háttérregiszter adja meg a tartomány méretét. A memóriaterület védelme a CPU hardverével valósul meg. Az alap és a limit regisztereket csak az operációs rendszer tudja betölteni, amely különleges kiváltságos utasítást használ. Mivel a privilegizált utasítások képesek csak kernelmódban hajthatóak végre, és mivel az operációs rendszer hajtja végre kernel módban csak az operációs rendszer tudja betölteni az alap és a limit regisztereket.

## Címek összerendelése

Általában egy program bináris futtatható fájlként található a lemezen. A végrehajtáshoz a programot be kell vinni a memóriába és el kell helyezni egy folyamaton belül. A használt memóriakezeléstől függően a folyamat a lemez és a memória között mozgatható a végrehajtás során. A lemezen lévő folyamatok, amelyek végrehajtásra a memóriába helyezésre várnak, a beviteli sort alkotják.

A legtöbb rendszer lehetővé teszi, hogy egy felhasználói folyamat a fizikai memória bármely részében tartózkodjon, így nem kell a címtérnek 000000 kezdődnie. A forrásprogramban szereplő címek általában szimbolikusak. A fordító általában ezeket a címeket áthelyezi és leköti (pl. 14 bájt a modul elejétől).

Címekhez kötés fajtái:

* Fordítási idő: Ha a fordítási időben tudja, hogy a folyamat hol fog elhelyezkedni a memóriában, akkor abszolút kód generálható.
* Betöltési idő: Ha a fordításkor nem ismert, hogy a folyamat hol fog elhelyezkedni a memóriában, akkor a fordítónak áthelyezhető kódot kell generálnia.
* Végrehajtási idő: Ha a folyamat a végrehajtása során áthelyezhető innen egyik memóriaszegmensből a másikba, akkor a kötést késleltetni kell a futásig.

## Logikai és fizikai címtér

A CPU által generált címet általában logikai címnek nevezik, míg a memóriaegység által látott cím – vagyis a betöltött cím a memória memória-címregisztere – általában valódi cím. A fordítási és a betöltési idejű címkötési módszerek azonos logikai és fizikai címeket állítanak elő.

A program által generált összes logikai cím halmaza egy logikai címtér. Az ezeknek a logikai címeknek megfelelő összes fizikai cím halmaza egy fizikai címtér. Így a végrehajtási idejű cím-kötési sémában a logikai és a fizikai címtér különbözik.

A futásidejű leképzést a virtuális címekről a fizikai címekre a memóriakezelő egységnek (MMU) nevezett hardveregység végzi. Az alapregiszter áthelyező regiszternek nevezik. Az áthelyezési regiszterben lévő érték hozzáadódik minden olyan címhez, amelyet egy felhasználói folyamat generál a cím memóriába. A felhasználói program soha nem látja a fizikai valós címeket. CPU által van generálva, lassabb de megbízhatóbb.

### Dinamikus áthelyezés

A rutin nem töltődik be, amíg meg nem hívják. Jobb memóriaterület kihasználás, a fen nem használt rutin soha nem töltődik be. Minden rutin a lemezen tárolható, és áthelyezhető betöltési formátumba. Hasznos, ha nagy mennyiségű kódra van szükség a ritkán előforduló esetek kezeléséhez. Nincs szükség speciális támogatásra az operációs rendszertől. Programtervezéssel valósul meg. Az operációs rendszer segíthet azáltal, hogy a könyvtárakat biztosít a dinamikus betöltés megvalósításához.

#### Dinamikus linkelés

Statikus linkelésnek hívjuk a rendszerkönyvtárak és programkódok a betöltő által a bináris képbe való kombinálást. Ez beleépíti a headert lényegében a programba. Ehhez több memória kell, ezáltal nem olyan hatékony.

Dinamikusnak pedig azt amikor a végrehajtási ideig el van halasztva a betöltés. A Stub/csonk megnézi az adott függvénykönyvtár be van-e már töltve. Ezáltal csökken az induló memóriahasználat, viszont idővel ahogy használjuk vissza magasodhat. A DLL azaz dynamic link libraryt a memória maximum 1 példányban tárolja el.

## Swapping / csere

Egy folyamat átmenetileg áthelyezhető a memóriából a háttértárba, majd vissza a memóriába. A folyamatok teljes fizikai memóriaterülete meghaladja a fizikai memóriát.

Háttértár: Elég gyors ahhoz, hogy minden felhasználó számára elférjen az összes memóriakép másolata, valamint közvetlen hozzáférést kell biztosítania ezekhez a memóriaképekhez.

Roll out, roll in: prioritás alapúra használt csereváltozat ütemező algoritmus. Az alacsonyabb prioritású folyamatok kicserélődnek így a magasabb prioritású folyamatok betölthetők és végrehajthatók. A csereidő nagy része az átadási idő; A teljes átviteli idő egyenesen arányos a felcserélt memória mennyiségével.

A főmemória két partícióra oszlik: A rezindes operációs rendszer, általában kevés memóriával rendelkezik a megszakításkor. A felhasználási folyamatok nagy memóriában vannak tárolva. Minden egyes folyamat egyetlen összefüggő szakaszában memória. Az általános regiszterek, amelyek megvédik a felhasználói folyamatokat egymástól, illetve az operációs rendszer kódjainak és adatainak változásaitól. Az alapregiszter a legkisebb fizikai címet tartalmazza. A háttérregiszter a logikai címek tartományát tartalmazza, minden logikai címnek kisebbnek kell lennie a háttérregisztertől. Az MMU dinamikusan képezi le a logikai címét. Ezután engedélyezheti az olyan műveleteket, mint például a kernelkód átmeneti állapota és a kernel méretének megváltoztatása.

### Memóriavédelem

Minden logikai címek a háttérregiszter által meghatározott tartományon belül kell lennie. AZ MMU dinamikusan képezi le a logikai címet azáltal, hogy hozzáadja az értéket az áthelyezési regiszterben. Ez a leképzett cím elküldésre kerül a memóriába. Ezt hívjuk hardveres támogatásnak a limit és az áthelyezés regiszternek.

## Többpartíciós allokálás

A multiprogramozás foka, amelyet a partíciók száma korlátoz. Változó partícióméretek a hatékonyság érdekében (egy adott folyamat igényeihez igazítva). Hole a rendelkezésre álló memória blokkja, különböző méretű lyukak vannak szétszórva a memóriába. Amikor egy folyamat megérkezik, memóriát foglal le egy elég nagy lyukból, ahhoz, hogy befogadja. A folyamat kilépése felszabadítja a partícióját, a szomszédos szabad partíciókat kombinálva. Az operációs rendszer a következő adatokat tárolja: lefoglalt partíciók, szabad partíciók

### Dinamikus memóriaallokálás

* **First** fit: Megyünk végig a memórián, és az első összefüggő üres helyre ahova tudjuk „berakjuk”, ott allokálunk neki memóriát
* **Best** fit: Oda rakjuk, ahol a legkisebb a lyuk, ami még elég a programnak. Ez a legkevesebb a pazarlást adja
* **Worst** fit: A lehető legnagyobb összefüggő memóriaterületen allokáljunk a programnak memóriát
* First-fit és Best-fit válthasználttá ez viszont feldarabolja a szabad helyet ami Külső töredezettséghez vezet.

#### Külső töredezettség

Van elég memória, de nem összefüggő terület. Kell egy „töredezettségmentesítő” .Egy gondolat a megoldásra → Compaction/Tömörítés:

Rendezzük újra az összes memória blokkot, hogy egy nagy üres memóriaterület maradjon. Csak gondolat maradt, mert ha egy program ugyanarra a memóriacímre hivatkozik, mint tömörítés előtt, nem fog működni, illetve az I/O műveletek sem szeretnék

##### Megoldások a töredezettségre:

###### Szegmentálás

A szegmentálás egy memóriakezelési séma, amely támogatja a memóriának ezt a programozói nézetét. A logikai címtér szegmensek gyűjteménye.Daraboljuk fel a programok, hogy a memóriában több kisebb darab legyen és ne egy nagyobb. Ez a megoldás már elavult, következménye a Paging. Minden szegmensnek van neve és hossza. A címek megadják a szegmens nevét és a szegmensen belüli eltolást.