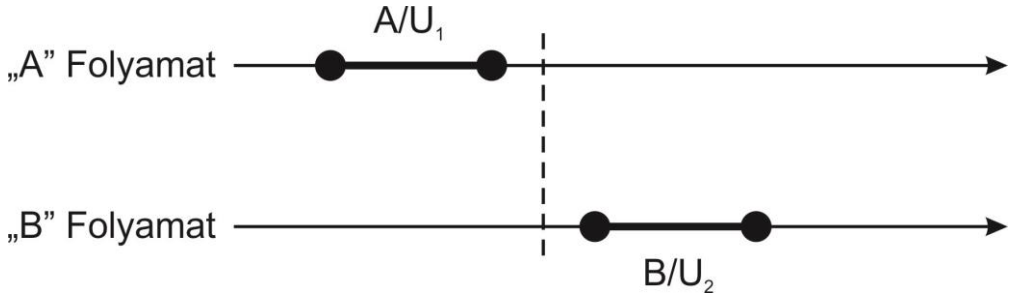
Előadás_#05.

1. Szinkronizáció

   Azok a folyamatok, melyek egymástól nem függetlenek, azaz valamilyen függőség fennáll közöttük, ahhoz, hogy a legkevesebb késedelemmel legyenek képesek adatot cserélni egymással összehangoltan kell együttműködniük. A másik folyamathoz való alkalmazkodás kompromisszumokkal és korlátozásokkal jár, de több a járulékos előny ebből, mint a hátrány. A folyamatok műveleteinek összehangolását nevezzük tehát szinkronizációnak.

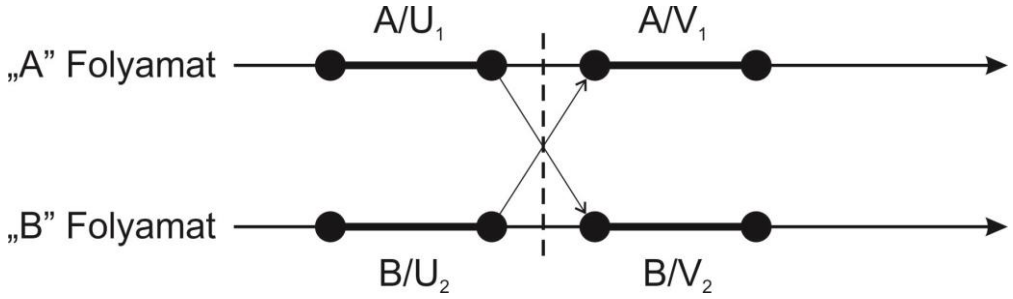
   Az összehangolt folyamatok lefutása a következők szerint történhet:

   - Előidejűség (Precedencia)



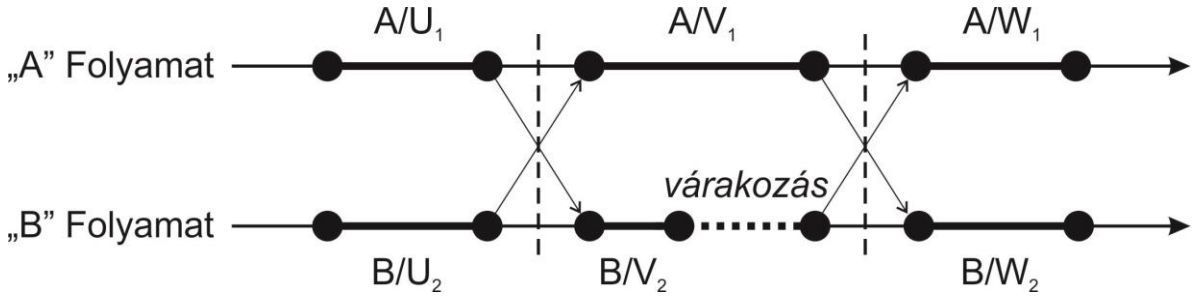
   Az egyes folyamatok meghatározott műveletei között előírt végrehajtási sorrend van meghatározva. Egy azt jelenti, hogy ez egyik folyamat adott utasításának feltétlenül meg kell előznie a másik folyamat adott utasítását. Nem csak puszta megelőzésről van szó, hanem arról is, hogy a második folyamat adott utasítása csak az első folyamat adott utasításának a befejezése után kezdődhet el, azaz semmiképpen nem lehet időbeli átfedés. Legjellemzőbb eset az, amikor a második folyamatnak szüksége van egy, az első folyamat által előállított adatra.

   - Egyidejűség (Randevú)



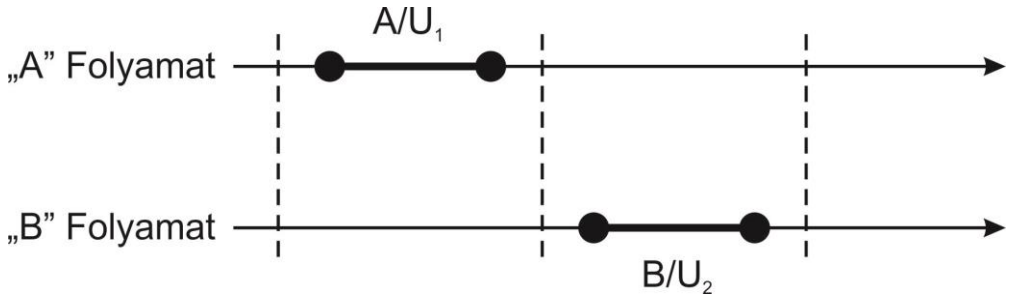
   Az egyes folyamatok meghatározott műveletei között előírt végrehajtási mód lehet az, hogy a műveleteknek be kell várniuk egymást, és egyidejűleg kell végrehajtódniuk. Az egyidejű végrehajtás az után kezdődhet meg, hogy az azt megelőző folyamatok közül leghosszabb is befejeződött.

A randevú speciális esete a meghosszabbított randevú.



Ez gyakorlatilag randevúk egymás utáni sorozatát jelenti. Jellemző, hogy ugyan a folyamatok egyszerre indulnak, szükségképpen mégsem biztos, hogy egyszerre fejeződnek be. Az ez után következő randevúra csak a lassabban lefutó folyamat befejeződése után kerülhet sor.

- Kölcsönös kizárás (Mutual Exclusion)



Az egyes folyamatok meghatározott műveletei között előírt végrehajtási mód lehet az, hogy a műveletek semmiképpen se egyszerre hajtódjanak végre. Sőt a műveletei még részlegesen sem fedhetik át egymást.

Könnyen belátható, hogy több együtt futó folyamat esetében léteznek olyan program szakaszok, melyek semmi esetre sem hajtódhatnak végre együtt. Legjellemzőbb eset egy közösen használt erőforráshoz való hozzáférés. A folyamatoknak azon utasítás sorozatait, melyek végrehajtása egyidejűleg nem megengedett, kritikus szakasznak nevezzük. Egy folyamat csak akkor léphet be a kritikus szakaszba, azaz csak akkor kezdheti meg a kritikus szakaszként kijelölt utasítássorozat végrehajtását, ha a kritikus szakaszban másik folyamat nem tartózkodik. Amennyiben egy másik folyamat foglalja a kritikus szakaszt, meg kell várni annak a (rész)folyamatnak a befejeződését, azaz a kritikus szakasz felszabadulását. Párhuzamosan futó folyamatok összetartozó kritikus szakaszai tehát kölcsönösen kizárják egymást.

A kölcsönös kizárás eredeti tipikus esete a nyomtatás egy megosztott nyomtatón, (fontos megjegyezni, hogy ez esetben a Windows előtti, azaz a nyomtatási sorkezelő nélküli, közvetlen elérésű sornyomtató vagy mátrixnyomtató szerepel a példában) hiszen nem lenne szerencsés, ha egy

lap szöveges tartalmának nyomtatása közben az egyik sor az egyik, a másik sor pedig egy másik folyamatból származna. Kölcsönös kizárással a második folyamat csak az első folyamat nyomtatásának befejezése után kezdhet maga is nyomtatni.

A kölcsönös kizárás ma is az egyik leggyakrabban használt szinkronizációs eszköz mind szoftveres, mind hardveres erőforrások közös használata esetén. Jelentősége azonban már nem a nyomtatás, hanem a fájl és port műveletek esetében van.

[term_fogy_modell.xlsx]

A kritikus szakasz gyakorlati megvalósításánál ügyelni kell arra, hogy a következő három feltételt teljesíteni tudjuk.

- A kölcsönös kizárás biztosítása
  A kritikus szakaszban egyszerre csak egy folyamat tartózkodhat.
- A haladás biztosítása
  Amennyiben a kritikus szakasz nem foglalt, és van legalább egy olyan folyamat, amely be szeretne lépni, akkor ezek közül egyet be kell engedni,
- A várakozási idő elviselhető szinten tartása
  Törekedni kell az éheztetés elkerülésére.

A kritikus szakaszra való belépés engedélyezése történhet szoftveres vagy hardveres úton. A szoftveres út az egyszerűbb, de időigényesebb megoldás. Az operációs rendszernek ismernie kell a kritikus szakasz elejét (Entry) és végét (Exit). Az érintett folyamatoknak képesnek kell lenni egyrészt a belépési szándék jelezésére, másrészt a kritikus szakasz elhagyásának jelezésére. Feltétlenül szükség van legalább egy változóra, amely jelzi a kritikus szakasz effektív foglaltságát.
A hardveres megoldásnak a processzor utasításkészletébe épülve kell képesnek lennie a foglaltság ellenőrzésének az elvégzésére. Ehhez speciális, megszakíthatatlan, egyidejűleg több művelet végrehajtására is alkalmas megoldásokra van szükség.

- TestAndSet
  A megnevezés azért van egybeírva, hogy ez is utaljon arra, hogy ez esetben egy lépésről van szó. Amennyiben egy érintett folyamat azt olvassa ki ebből a változóból, hogy szabadon beléphet, akkor belép és ezzel a TestAndSet logikai változó értékét szabadról foglaltra állítja. A megoldás tehát összesen egy darab logikai változót használ, viszont igényli azt, hogy a PRAM modell mindenképpen az "OlvasÉsÍr"oszthatatlan műveleti sorrendet kövesse.

- Swap

  Az elnevezés találó, hiszen ez esetben valóban egy cseréről van szó. Minden érintett folyamat rendelkezik egy kulcsnak nevezett logikai változóval. A kulcs 1-es értéke azt jelzi, hogy az adott folyamat be szándékozik lépni a kritikus szakaszba. Maga a kritikus szakasz szintén rendelkezik egy logikai változóval, aminek lakat az elnevezése. Amennyiben a kritikus szakaszban nem tartózkodik folyamat, a lakat értéke 0.

  A kritikus szakaszba való belépéshez az összes érintett program meg kell, hogy vizsgálja a lakat változó értékét. Amennyiben egy folyamat a lakat értékét 0-nak találja, akkor a lakat értékét lecseréli (Swap) saját kulcs változójának értékére (ami 1 volt), és belép a kritikus szakaszba. A többi folyamat a lakat vizsgálata során már 1-es értékkel találkozik, azaz nem léphet be. A kritikus szakasz elhagyása után a folyamat visszacseréli a lakat és a kulcs változók értékét, ezzel lehetővé téve egy másik folyamat belépését a kritikus szakaszba.

  Meg kell jegyezni, hogy a fent ismertetett két megoldás, a három teljesítendő feltétel közül csak az első kettőt tudja maradéktalanul megvalósítani. A harmadik feltétel (a várakozási idő elviselhető szinten tartása) csak lényegesen bonyolultabb eljárással valósítható meg.

- Szemafor

  A szemafor algoritmust Edsger Wybe Dijkstra (1930-2002, holland matematikus-informatikus) dolgozta ki, 1968-ban. A megoldás működési mechanizmusa egy egyvágányos vasúti sínszakasz védelmének modelljét vette alapul. A szemafor a szinkronizációs problémák megoldásának univerzális eszköze, hiszen a kölcsönös kizárás megvalósításán túl az előidejűség és az egyidejűség esetében is használható.

  A működés megvalósításához szükségünk van:

  - egy nem negatív értékeket hordozó változóra, ennek neve "s"
    (létezik bináris szemafor is ez esetben az "s" értéke csak 0 vagy 1 lehet)
  - három atomi, azaz nem megszakítható műveletre, melyek elegendőek a modell működéséhez:
    - "Init" [beállít] a szemafor inicializálása, alaphelyzetbe állítása, azaz "s"=1 (kölcsönös kizárás) vagy "s"=0 (előidejűség) illetve "s" 1-nél nagyobb értékre állítása, mely érték a kezelt erőforrások számát mutatja meg.

- "Wait(s)" [vár] Ez a művelet "s" értékét csökkent 1-el, amennyiben "s" értéke nagyobb mint 0. Nevével ellentétben a wait művelet nem minden esetben várakoztat, csak az "s"=0 esetben történik várakoztatás. (Eredeti elnevezése "P" a *proberen* holland szó után [kipróbálni])
- "Signal(s)" [jelez] Ez a művelet "s" értékét 1-el megnöveli. (Eredeti elnevezése "V" a *verhogen* holland szó után [megnövelni])

A működés mechanizmusa a következő.
Amennyiben "s" értéke 1, ez azt jelenti, hogy a modell a kölcsönös kizárást valósítja meg. A kritikus szakaszba belépni szándékozó folyamat egy "P" műveletet hajt végre, aminek hatására "s" értéke 0 lesz, és belép a kritikus szakaszba. A kritikus szakaszban végrehajtott programrész befejezése után egy "V" műveletet hajt végre a folyamat, azaz "s" értékét 1-el megnöveli, jelen esetben visszaállítja 1-re. A folyamat elhagyja a kritikus szakasz, lehetőséget teremtve másik folyamatok számára a belépésre.
Amennyiben "s" értéke 0, ez azt jelenti, hogy a kritikus szakasz használatban van. A kritikus szakasz felszabadulása után "s" értéke 1 lesz. Fontos megjegyezni, hogy nem mindegy, hogy az "s" értéke az inicializálást követően lesz 1 vagy egy "V" művelet után.
Amennyiben "s" értéke 2 (vagy ennél is több), akkor több folyamat egymás mellett képes belépni, mert az "s" értéke ez esetben azt mutatja meg, hogy a szemaforhoz tartozó erőforrásból mennyi áll rendelkezésre. Belépéskor a folyamatok sorban egymás után csökkentik (P), kilépéskor pedig növelik (V) "s" értékét.
[Szemafor.xlsx]

Működés közben sok folyamat teszteli az "s" értékét, ami legtöbbször felesleges aktív processzor használatot jelent. Lassú folyamatokat nem célszerű futásra kész állapotban tartani, hiszen a processzor, mint erőforrás ezen folyamatoknak még hiányzik, de számos más erőforrást viszont feleslegesen tartanak foglalásban. Célszerűbb a lassú folyamatok esetében altatni, azaz passzív várakozást alkalmazni, hogy elkerüljük az erőforrások pazarlását.

2. A "Monitor" típusú programszerkezet

   A szemaforral ellentétben a monitor egy magasabb szintű, objektum elvű, objektum orientált művelet. Ezt a programszerkezetet C.A.R Hoare angol programfejlesztő dolgozta ki 1974-ben. A monitor objektum egy több szál által használt eljárás nem párhuzamos végrehajtását teszi lehetővé, azaz nem folyamat szintű művelet. Ötvözi az objektum orientált programozást a szinkronizációs metódusokkal.

   A monitor objektum a következőkből áll:

   - osztott adat
   - ezeket az adatokat feldolgozó eljárások
   - monitort inicializáló metódusok

   A megvalósításhoz ez esetben is három dolog kell, egy feltételváltozó és két művelet (Wait és Signal), de a kritikus adatokhoz csak szinkronizált műveleteken keresztül lehet hozzáférni.
   Minden eljárás halmazt egy monitor kontrolál. A többszálas alkalmazás futásakor, a monitor egyetlen szálnak engedélyezi egy adott időpontban az eljárás végrehajtását. Ha egy szál éppen egy monitor által kontrolált eljárást akar futtatni akkor az lefoglalja a monitort. Abban az esetben ha a monitor már foglalt, akkor a belépni szándékozó szál várakozik amíg a monitort lefoglaló szál befejezi az adott eljárás végrehajtását és felszabadítja a monitort. Előnye, hogy a közösen használandó erőforrások és az erőforrásokon végrehajtható műveletek egyetlen szintaktikus egységben helyezkednek el.

   A monitor és a szemafor összehasonlítása:

   - A monitor magasabb szintű eszköz. Magas szintű, objektumorientált nyelvek esetében (pl. Java) roppant egyszerűen használható.
   - A szemaforok segítségével rugalmasabban kezelhetők a kritikus szakaszok, hiszen nem kell beágyazott módon, előre elkészíteni az összes kritikus szakaszt.
   - A szemaforokkal több kritikus erőforrás használatát könnyebb általában összekombinálni egy kritikus szakaszon belül.