Feladatok (task) együttműködése

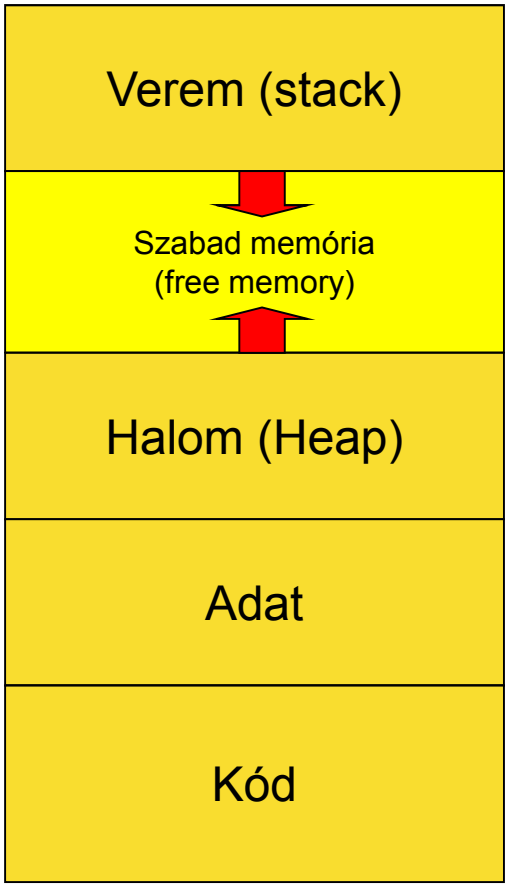
dr. Kovácsházy Tamás

4. anyagrész,

Feladatok implementációja, folyamatok és szálak

Feladat (task) fogalom megvalósítása...

- Feladat fogalom eredetileg folyamat (process) értelemben került használatra.
- A folyamat a végrehajtás alatt álló program.
  - Ugyanabból a programból több folyamat is létrehozható.
  - Saját kód, adat, halom (heap) és verem memória területtel rendelkezik.
  - Védettek a többi folyamattól.
    - Szeparáció, virtuális gép, sandbox.



Folyamatok szeparációja

- Virtuális CPU-n futnak:
  - Nem férhetnek hozzá a többi folyamat és az operációs rendszer futása során előálló processzorállapothoz.
  - Kontextusváltás történik, ha más folyamat kerül futásra.
- Saját virtuális memóriaterületük van (később).
  - Nem férhetnek hozzá más folyamatok virtuális memóriájához vagy direkt módon a fizikai memóriához.
  - A processzor MMU-ja oldja ezt meg.
    - Lehetséges azonos fizikai memóriaterületek megosztása például olvasási joggal (pl. kód terület).
    - A modern MMU-k ezt akár írási joggal is meg tudják tenni...

Folyamatok létrehozása

- OS specifikus rendszerhívás (pl. CreateProcess(), fork(), stb.).
- Szülő/gyermek viszony a létrehozó és a létrehozott között.
  - Process fa (process tree)
  - A szülő erőforrásaihoz hozzáférés többnyire konfigurálható (mindenhez – semmihez).
  - A szülő megvárhatja a gyermek terminálódását, vagy futhat vele párhuzamosan.
  - Paraméterezhető a gyermek (command line).
- UNIX fork() részletesen tárgyalásra kerül később.
- Sok adminisztráció, erőforrásigényes.

Folyamatok kommunikációja

- A folyamatoknak együtt kell működniük (később lesz róla részletesen szó).
  - Ehhez kommunikálniuk kell.
- Két tetszőleges folyamat nem tud közös memórián keresztül kommunikálni.
  - Az MMU és a virtuális memória éppen ezt kívánja lehetetlenné tenni (szeparáció).
  - Csak OS rendszerhívásokon keresztül tudnak kommunikálni, ami erőforrás igényes.
- Hatékony a védelem/szeparáció szempontjából.
- Nem hatékony módja a párhuzamos, erősen összefüggő feladatok megoldásának.
  - Pl. akár GUI + számításigényes feladat (WORD „újraszedés”)

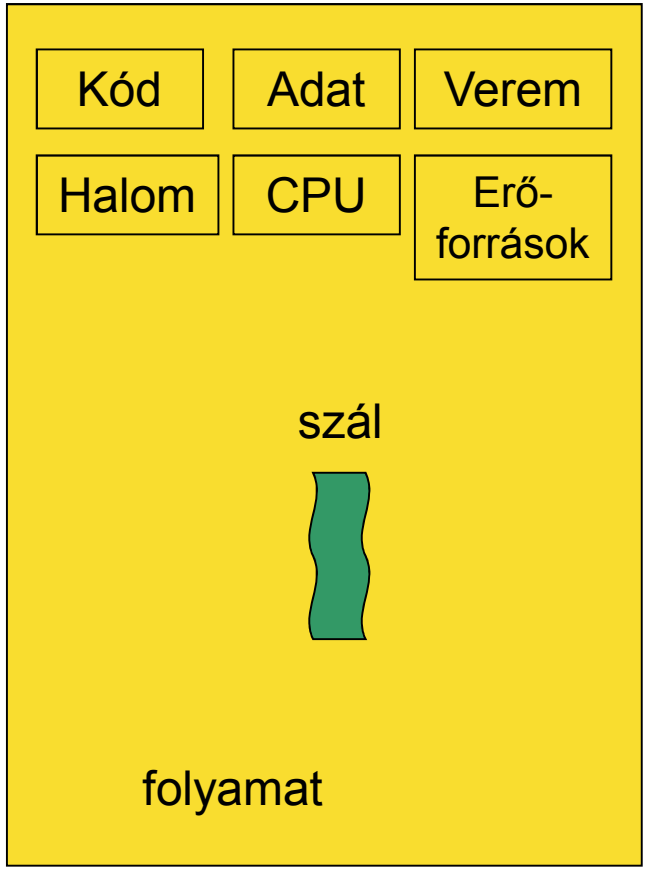
Folyamatok befejezése

- OS specifikus rendszerhívás (pl. TerminateProcess(), exit(), stb.).
- Nyitott, használatban lévő erőforrásokat le kell zárni.
  - Pl. nyitott file-ok, stb.
- A szülő megkapja a visszatérési értéket, többnyire egy egész értékű változó (integer) formájában.
- Mi történik, ha a szülő folyamat befejeződik, de a gyermek nem?
  - OS függő megvalósítás, tipikus megoldások:
    - Alapértelmezett szülő folyamat (pl. UNIX init folyamat).
    - A gyermek automatikus befejezése (cascading termination).
- Sok adminisztráció, erőforrásigényes.

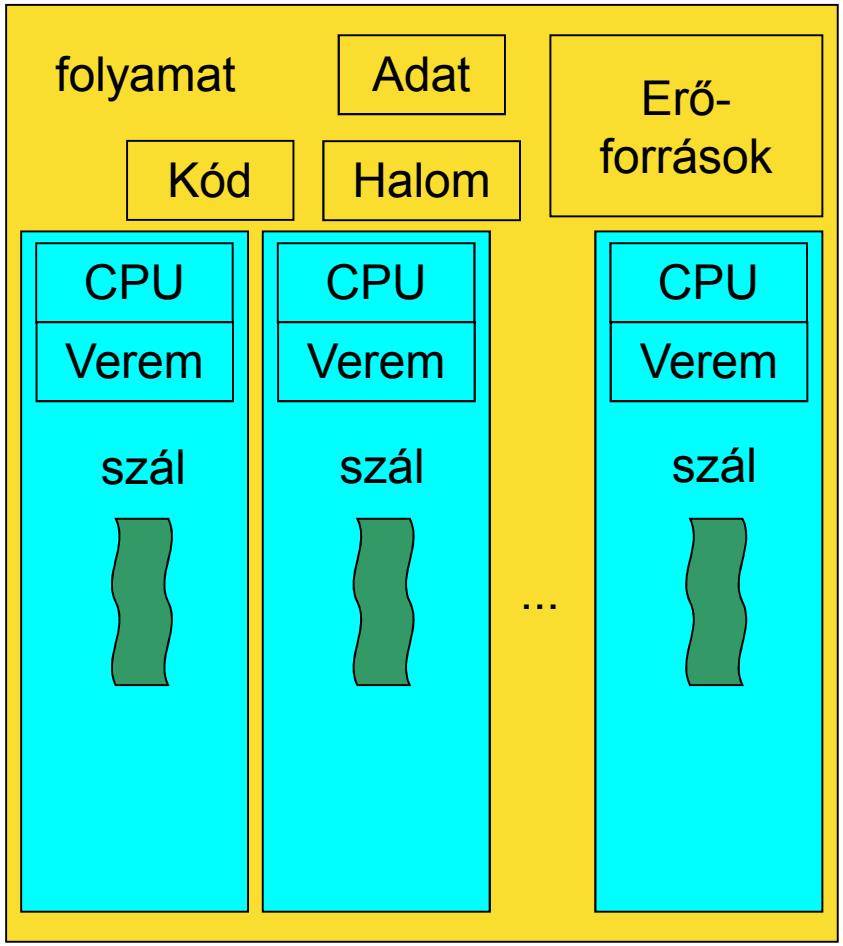
Folyamatok értékelése

- Védelmi/szeparációs szempontból jó megoldás, de erőforrásigényes:
  - Folyamat létrehozása és megszüntetése.
  - Folyamatok közötti kommunikáció és erőforrásmegosztás.
- Megoldás: Szál (thread) bevezetése:
  - A szál a CPU-használat alapértelmezett egysége, magában szekvenciális kód.
  - Saját virtuális CPU-ja van, és saját verem áll rendelkezésre.
  - A kód, adat, halom és egyéb erőforrások (pl. file) tekintetében osztozik azokkal a további szálakkal, amelyekkel ***azonos folyamat kontextusában*** fut.
- Folyamat = nehézsúlyú folyamat (heavyweight process)
- Szál = pehelysúlyú folyamat (lightweight process)

Folyamat és szál eltérése ábrán



Egyszálas
tiszta folyamat alapú rendszer



Többszálú rendszer
folyamat alapon

Szálak támogatása

- Jelenleg a modern operációs rendszerek natív módon támogatják szálak létrehozását.
- Windows:
  - Program v. szolgáltatás = folyamat, folyamaton belül szálak.
  - Az ütemező szálakat ütemez.
- Linux:
  - Program v. daemon = folyamat, folyamaton belül szálak.
  - Az ütemező task-okat ütemez, amik lehetnek folyamatok vagy szálak.

Felhasználó módú szálak

- Korábban a UNIX alatt (Linux alatt is).
  - green threads
- Az OS csak folyamat szintet ismer.
- Szükség van szálakra.
  - Felhasználói módú szál könyvtárak...
- Az OS csak a folyamatot tudja ütemezni, ha az fut, akkor azon belül a felhasználói módú szál könyvtár saját ütemezője fut.
  - Több szál felel meg egyetlen ütemezési egységnek!
    - Nem tudja kihasználni a több processzoros rendszerek előnyeit.

Szálak támogatása (létrehozása)

- Pl. Win32 API, Pthreads, JAVA thread
- Win32: CreateThread() bonyolult paraméterezéssel.
- Pthreads: POSIX threads pl. Linux és más UNIX variánsok kernel vagy akár user szinten (csak viselkedést ad meg).
- JAVA (VM a folyamat, VM-en belül szál):
  - Thread osztályból származtatva
  - Runnable interface megvalósítása
  - A JAVA platform-specifikusan valósítja meg a szálat:
    - Natív OS specifikus szál (one-to-one, tipikus).
    - JAVA specifikus szálak (many-to-one) egy natív OS szálra vagy folyamatra leképezve.
    - many-to-many leképzés (kevesebb erőforrás kell mint a one-to-one esetben, de lehet párhuzamosan futtatni a szálak közül néhányat, amit a many-to-one nem tud).

Szálak alkalmazásának előnyei

- Kis erőforrás igényű a létrehozásuk és megszüntetésük.
  - Kb. egy nagyságrenddel gyorsabb mérések szerint.
- Alkalmazáson belüli többszálúság támogatása.
  - A GUI válaszol, még ha háttérben csinál is valamit az alkalmazás (pl. számol).
- Gyors kommunikáció közös memóriában az azonos folyamat kontextusában futó szálakra.
  - Csak a verem szál specifikus, a többi osztott.
- Skálázhatóság.
  - Alkalmazáson belül kihasználható több CPU.

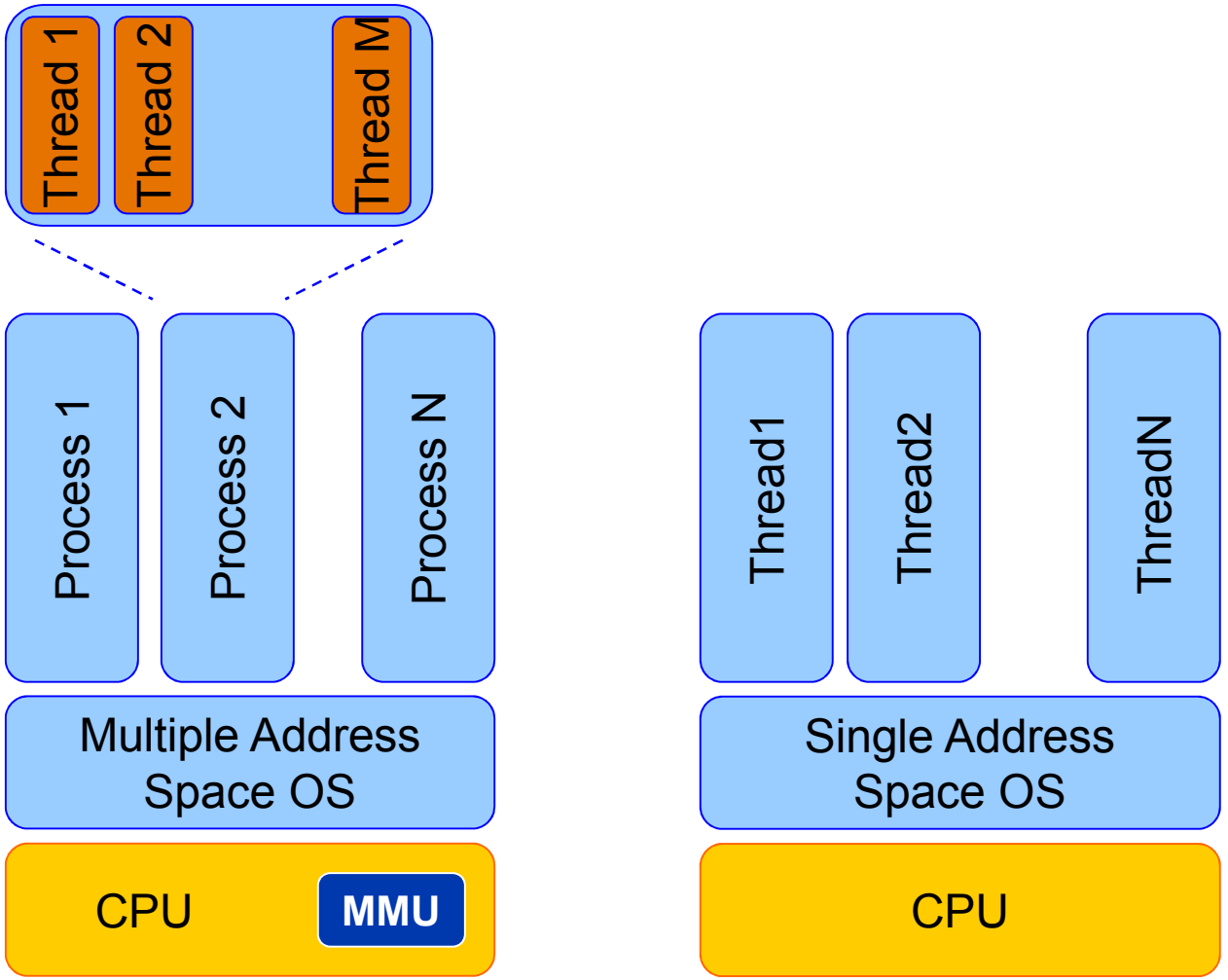
Szálak alkalmazásának következményei

- A közös memórián keresztüli kommunikáció veszélyes.
  - A kommunikációra használt memóriaterületek (struktúrák vagy objektumok) konzisztenciája sérülhet.
  - Több előadás szól majd erről a témáról (kölcsönös kizárásnak fogjuk majd a megoldást hívni).
  - Eltérő folyamatok kontextusában futó szálak kommunikációja az OS-en keresztül történik.
    - Erre ritkábban van szükség, mivel a szorosan összekapcsolódó (sokat kommunikáló) feladatok megvalósíthatóak egy folyamaton belül

HW támogatás

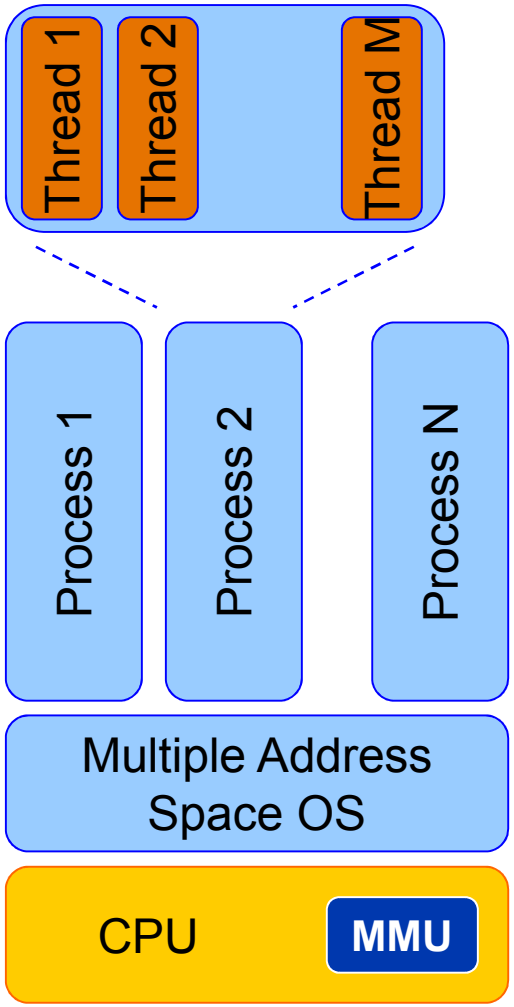
Virtuális memória MMU-val

Csak fizikai memória, MMU nélkül
(pl. egyes beágyazott OS-ek)

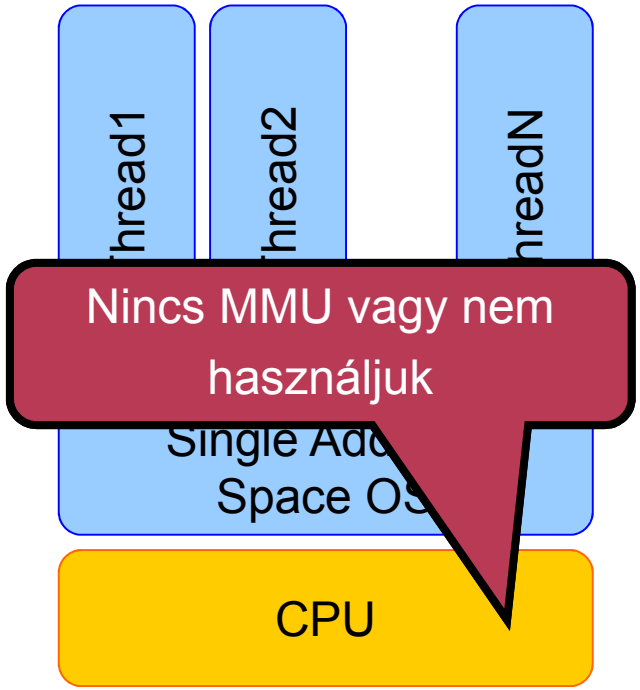


HW támogatás

Virtuális memória MMU-val



Csak fizikai memória, MMU nélkül (pl. egyes beágyazott OS-ek)



Coroutine és fiber (rost?)

- Kooperatív multitasking
  - Folyamaton vagy szálon belül.
    - OS támogatással vagy programnyelvi megvalósítás.
    - Az OS szinten a coroutine-eket vagy fiber-eket tartalmazó folyamat vagy szál kerül ütemezésre
    - Az ütemezési algoritmus a programozó kezében van, neki kell implementálnia (kooperatív ütemezés).
  - Coroutine: programnyelvi szintű elem
    - Haskell, JavaScript, Modula-2, Perl, Python, Ruby, etc.
  - Fiber: rendszerszintű eszköz
    - Win32 API (ConvertThreadToFiber and CreateFiber).
    - Symbian

Coroutine

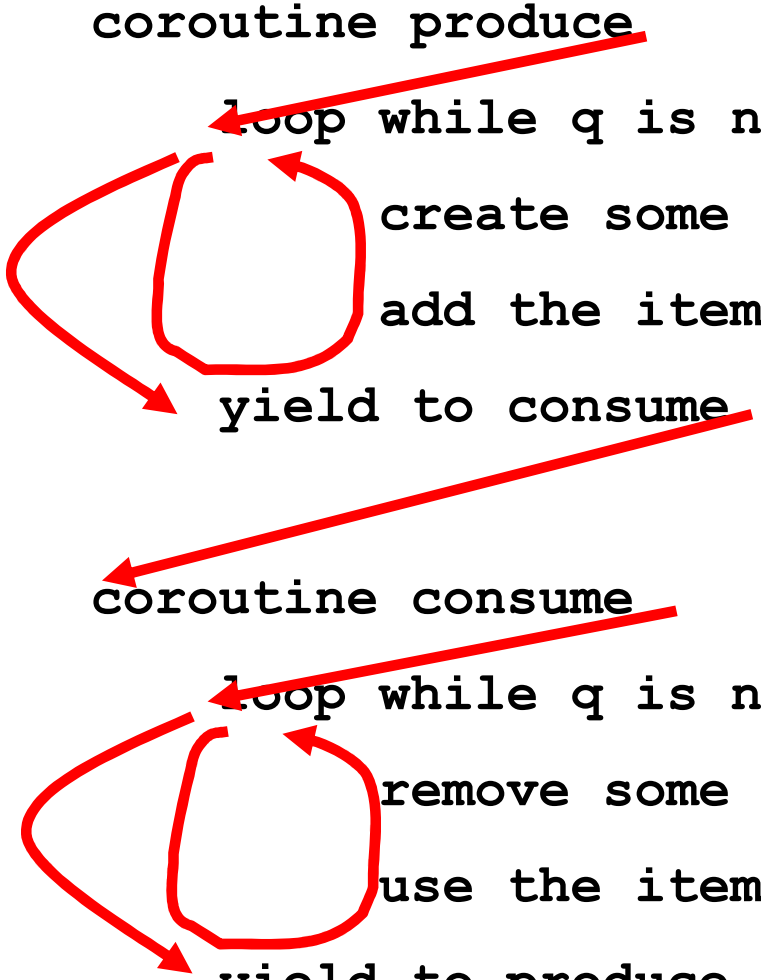
- Subroutine általánosítása
  - Subroutine:
    - LIFO (Last In/called, First Out/returns).
    - Egy belépési pont, és több kilépési pont (return/exit)
    - A vermet használja a paraméterek és a visszatérési érték átadására.
  - Coroutine:
    - Az első belépési pont azonos a subroutine-nal.
    - Utána viszont a legutolsó kilépési pontra tér vissza!
    - Átlépés a „yield to Coroutine_id” utasítással lehet.
    - ***Nem használhat vermet***, hiszen az megtelne (ide-oda lépked, igazából nem tér vissza).

Coroutine példa, 1. „hívás”

```
var q := new queue

coroutine produce
    loop while q is not full
        create some new items
        add the items to q
    yield to consume


coroutine consume
    loop while q is not empty
        remove some items from q
        use the items
    yield to produce
```

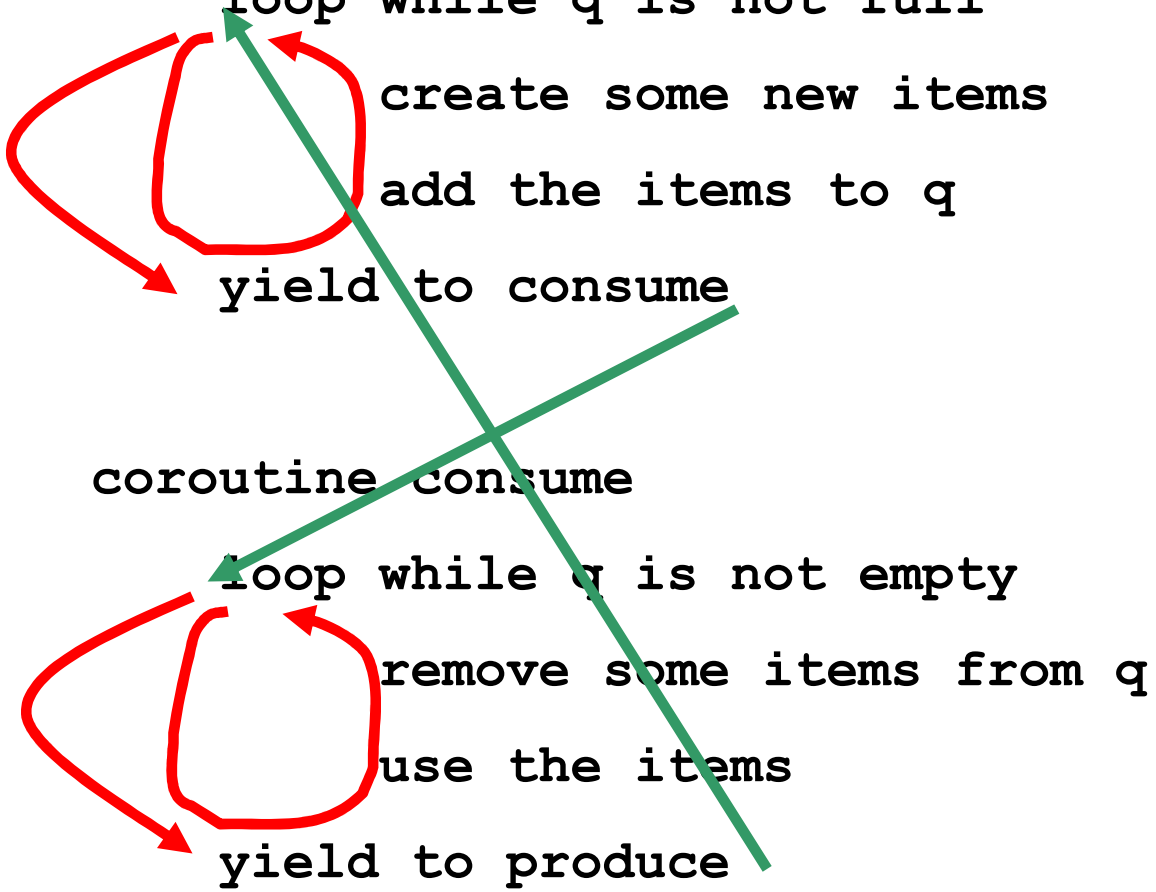


Coroutine példa, „hívás” később

```
var q := new queue

coroutine produce
    loop while q is not full
        create some new items
        add the items to q
        yield to consume


coroutine consume
    loop while q is not empty
        remove some items from q
        use the items
        yield to produce
```



Méréstechnika és Információs Rendszerek Tanszék

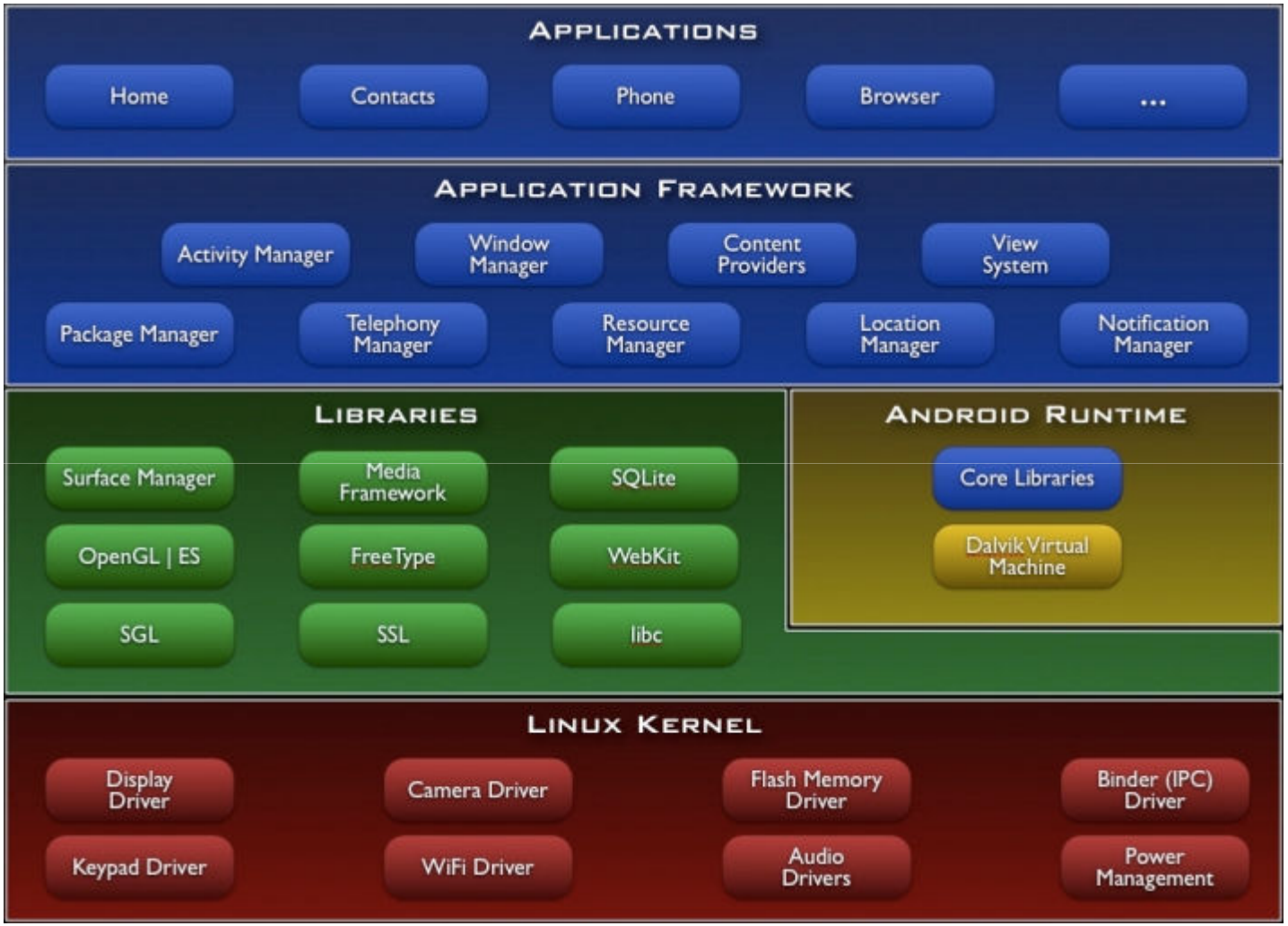
Coroutine és fiber értékelése

- Kooperatív multitasking-gal megoldható problémák esetén.
- Verem alapú környezetben (pl. C/C++) nehéz az implementációja a coroutine-nak (fiber rendszerszintű).
- Egyes kooperatív együttműködést igénylő feladatok jól megvalósíthatóak vele.
- Nem szükséges osztozni az erőforrásokon (kooperatív):
  - Nem kellenek OS hívások.
  - Kisebb erőforrás igény.
- Az OS ütemezi egy szálban/folyamatban őket, vagyis egyetlen végrehajtó egységet tudnak csak kihasználni.

Egyéb érdekes megoldások…

- Az Android Linux alapon (folyamat és szál támogatás) felépít egy érdekes keretrendszert, ami túllép a Linux-on
- Nézzük meg ezt részletesen…

Android



Android alkalmazás és a Linux

- Application Security Sandbox
- Az Android alkalmazások egy alkalmazás specifikus UNIX folyamatban futó Dalvik VM-ben futnak.
  - Saját folyamatban a Dalvik VM saját példánya:
    - Szál és alacsony szintű memória menedzsment standard Linux módon.
  - A Dalvik speciálisan mobil eszközökre lett kifejlesztve:
    - Alacsony memória használat.
    - Nem verem, hanem regiszter alapú gép.
- Minden alkalmazás külön Linux UID-et kap, hogy még véletlenül sem tudjanak egymáshoz férni.
  - Ennek megfelelően állítódnak be a hozzáférési jogosultságok mindenki más hozzáférését tiltva az alkalmazás által kezelt erőforrásokhoz
  - A legkisebb jogosultság alapelve (*principle of least privilege*)
- Az alkalmazás tulajdonságait a Manifest File adja meg.
- Az alkalmazás erőforrás hiány miatt bármikor terminálható
  - Ezt az operációs rendszer automatikusan meg is teszi
  - Vannak erre eszközök is

Android alkalmazás komponensek 1.

- Aktivitás (activity)
  - Egy képernyő felhasználói felülettel
  - Egy alkalmazáson belül több lehet/van
  - Egy alkalmazáson belül az aktivitások függetlenek egymástól, de együttműködve teszik lehetővé az alkalmazás használatát
  - 3 féle állapot:
    - Resumed (képernyőn aktív), Paused (képernyőn inakatív, az aktív részben takarja), Stopped (teljesen takart, leállított)
    - Mindig az éppen a képernyőn lévő aktivitások futnak, de a többi állapota is tárolásra kerül
    - Lifecycle callbacks
  - Külső alkalmazások elindíthatják az aktivitást, ha az engedélyezve van

Android alkalmazás komponensek 2.

- Szolgáltatás (service)
  - Háttérben futó funkció felhasználó felület nélkül
    - Folyamatosan futó háttérfeladatok megvalósítására (pl. mp3 lejátszóban a fájl tényleges lejátszására)
    - Nem csinál saját szálat az alkalmazásban, ha CPU intenzív, akkor csinálni kell neki egyet
  - Started szolgáltatás
    - Addig fut, amíg nem végzi el a dolgát, túlélheti az alkalmazást
    - Pl. nagy fájl letöltése
  - Bound szolgáltatás
    - Együtt él a szolgáltatással
    - Egy jól meghatározott interfészen keresztül érhető el az alkalmazásból
    - P. MP3 lejátszó háttér komponens play, stop, előre- és hátratekerés, lejátszási sebesség, hangerő, stb. beállítási lehetőségekkel
  - Lifecycle callbacks it is megtalálgatóak