Operációs rendszerek

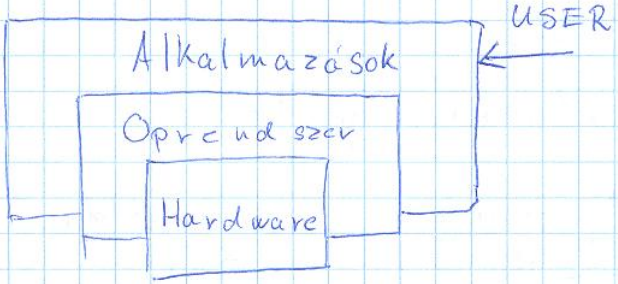
kondor@iit.bme.hu IB 323
www.iit.bme.hu/~kondor

Az informatika az a tudomány amely a következő dologgal foglalkozik:

- információ
- információval végezhető műveletek (tárolás, továbbítás...)
- információn végzett műveletekkel segítő rendszerek (feldolgozás)
- információ feldolg. rendszerek, műveletei (tervezés, létrehozás, üzemeltetés...)

→ Informatika I. : HW, Opr.
Infó 2: Hálózat, Adatbázisok, Programozási nyelvek

Számítógépes rendszer



operációs rendszer: a hardware-t használhatóvá tevő, az ott a legkülönfélébb hozzá. egységes felület biztosításával a futtatandó programok számára.

→ hatékonyságjavító rendszerek (hardware jól kihasználása)

- mamut operációs rendszerek (integrálva rengeteg funkció)
  → problémák: nem fedhetők uralni, sok hiba (softwarekrízis)

-> csak áttekinthető keretben tudjuk uralni

-> külön kezelhető egységek
(leváltak pl.: adatbáziskezelő, stb.)

ma megint integrálás felé ment el az oper. rendszer
(amit megvesz a boltban ráteszem és tudja kezelni)

CPU kihaszn.: $\frac{\text{hasznos idő}}{\text{összes idő}} \cdot 100$ [%]
[CPU utilisation]

Átereszőképesség: job/óra
[throughput]

Korai rendszer:

egyszerű monitor:

offline - spooling:

multiprogramozás:

Azóta ...

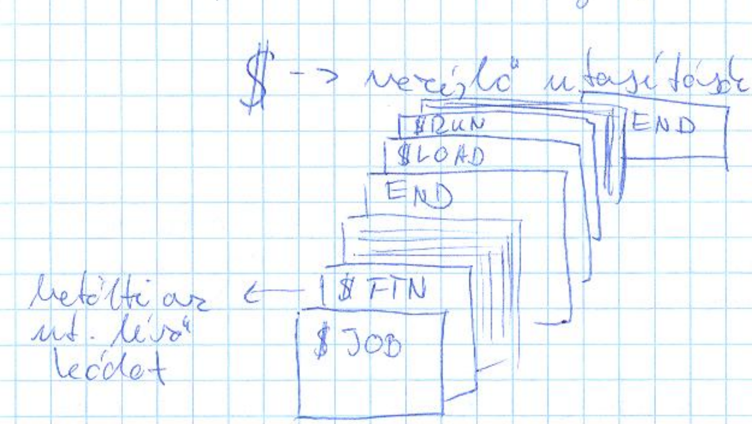
Korai rendszer: CPU_K: 7% Á: 4 job/óra

open shop (gépteremben van a progr.)

closed shop (programozó - operátor)

Egyszerű monitoros rendsz.:
(operátort „belehelyezték" a gépbe)

$ -> vezérlő utasítások ezzel kezdődnek



CPU_K: 55%

Á: 33 Job/óra

Mem.

IT vezérlő
perif. kezel.
Job. vez.
felh. felület

- automatizálta hogy mikor van vége az adott Job-nak
- nem kell operátori munka
- különböző védelmi problémák.

OPERÁCIÓS RENDSZEREK

2008. 10. 30

2 mérőszám / (processzor kihasználtsága (annak a progr. a [illegible] futtatása amelyet a felh. akar futtatni)

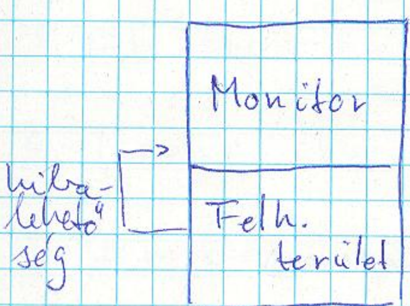
L> ez a hasznos idő / az oprendsz. futtatása nem tart. bele /

throuput: időegység alatt mennyi munkát képes elvégezni a processzor

Ezzel a 2 mérősz. megtudjuk mérni a teljesítményt, összehasonlítási alapot ad.

A korai rendsz. 7% kihasználtság és 4-5 job/óra, ezt sikerült az egysz. monitoros rendszerrel növelni 50%-ra és 35-40 job/órára

M



- fellép a memória védelem igénye
- a különböző jobokat egymás után pakolták (ha a felh. maga kezeli a kártyaolvasót „megkerüli" a kártyakezelő felhasználó jogait)
  L> I/O védelem kialakítása is felmerül

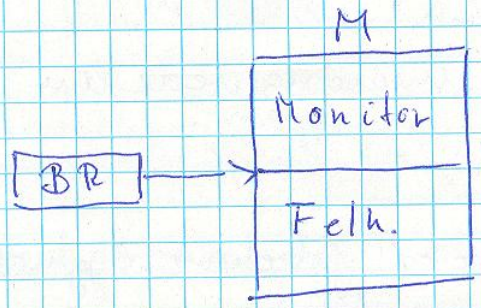
Hardware tám. kell!

(1) Nem szabad mindent megeng. a felhasználónak, de mindent meg kell enged. az operációs rendszernek

↳ 2 különböző üzemmód
- felhasználói mód
  / korlátozott utasítás készlet lehetősége /
- rendszer mód
  / minden megengedett /

Minden rendszerprogr. utasítást privilégizáltá kell tenni
↳ a felh. ne kapcsolhassa át az üzemmódot

Egy bázisregisztert a memóriában beállítom a felh. területet.



/ a bázisreg. átírása privilégizált /

Ha felh. módban privileg. utasítás jön akkor megszakítás.

A periféria kezelő utas. a privil. utasítások közé kell hogy tartozzanak
- ha I/O utasítás jön akkor megszakítás, vagyis egy vektortérbeli címre ahol a periféria kezelő utas. van

↳ a vektortérképet is Monitor területre kell tenni

- a processzor alapvető utasításainak köre privilégizált

pl.: MASK ; HLT
↓
csak megszakítással lehet továbblépni, de ha maszkolva van akkor csak a RESET gomb használatos

/ezt nem engedhetem meg hogy egy felhasználói programba kerüljön/

CPU_k = 55 %
A' = 33 job/óra

/operátori munka minimális, nem lassítja manuális munka/
↳ periféria lassítja a működést

Sebességkategóriák:

1. Elektronikus : μs tartomány (ma már ns)

2. Mechanikus : ms

3. Ember : s → ∞

a program általános szerkezete:

beolvas /mechanikus eszköz/ — t_b
feldolgoz /elektronikus/ — t_f
kiír /mechanikus eszköz/ — t_k

1 input utas. alatt 1000 másik utas. végrehajtandó lenne hajtani

- kedvezőtlen időben a CPU kihasználatlan

$$CPU_k = \frac{t_f}{t_b + t_f + t_k}$$

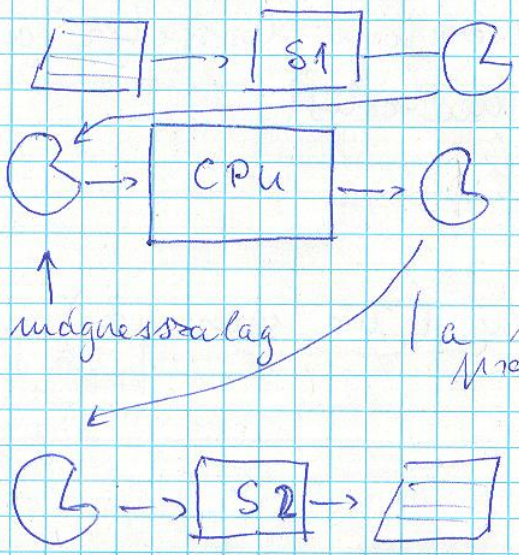
↳ mechanikai időkat párhuz. tesszem a proc. feladatrégek. idejével

| DE! nem biztos hogy minden programot megtehet így írni |

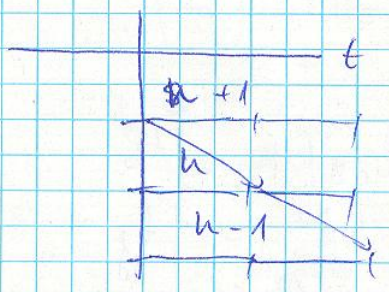
↓

Offline I/O műveletek — új módszer

- fut egy program, közben a kimenetkezőt kiolvasom és az előz. kiírom



(a lassú I/O-kat kivesszem a processzorral)



=> pipeline szerű

$CPU_k = 90\%$

$A' = 55$ j/óra

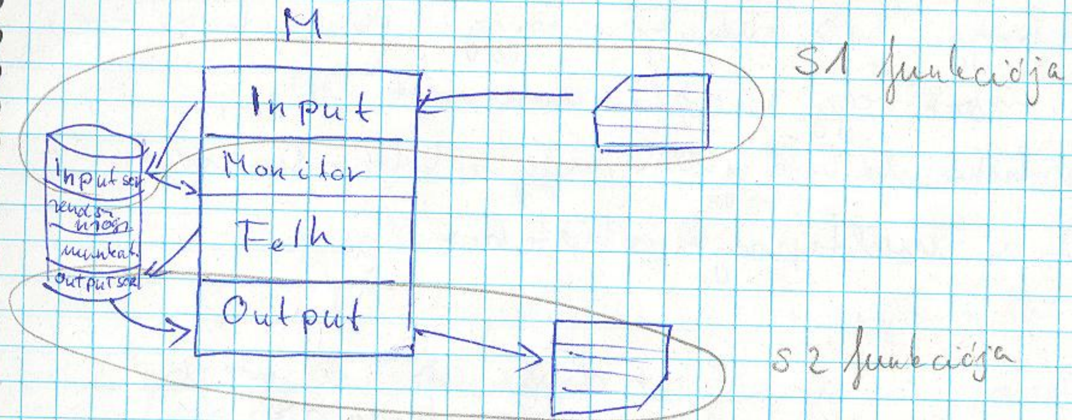
- készülékfüggetlen programozás
/logikai perifériák mögött tetszőleges fizikai készülék/

uniformizálása a kiírható és bevihető paramétereknek

egységes I/O-felület létrehozása

SPOOLING:

- eltudtunk indítani nyomtatást úgy hogy közben visszakaptuk a vezérlést



A lassú periféria megszakításkezeléssel működik

/csak annyi időre veszi el a proc. amennyi az 1 sor vagy 1 sor kiírásához kell/

Ezt a rendszer 1 gépen belül valósították meg

Multiprogramozott operációs rendsz. kialakulása

- a memória gyorsabbá vált mint a lemez.
- bővült a perifériakészlet

CPU lökést => jobok apróbb lépésű felépítése
I/O lökést

• egyes progr. sok CPU löketet hajtanak végre míg mások sokkal több I/O műveletet

pl. egyenletmegoldás, konvertáló

-> CPU intenzív és I/O intenzív jobok

(a SPOOLING rendezi ki egy. a jobokat

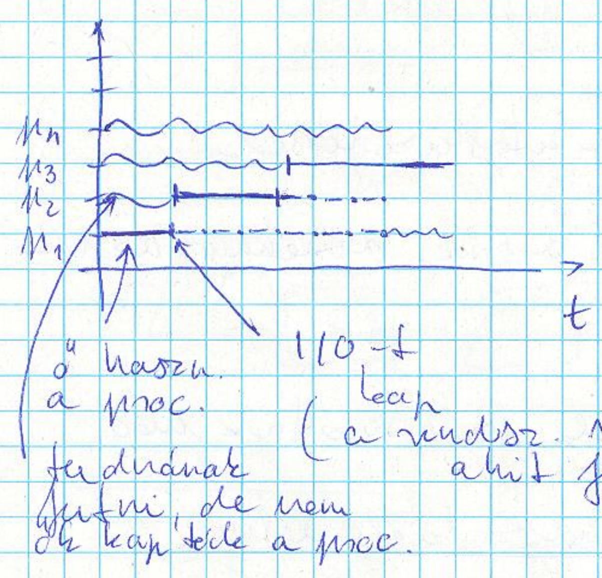
L> kiegyenlített jobunk)

Az oprendszer kérhessen el a sorrendtől, találja ki hogy I/O intenzív vagy CPU intenzív-e, és állítson össze egy ideális sorrendet.

/ egyenként elérhető jobok, így kell tárolni és /

↓

ez vezetett a multiprogr. ötletéhez



- a rendsz. a megsz. tudja hogy az I/O művelet befejez.

PRE EMPTY - idő előtti kiürítéses rendsz. az alsóban ha P_1 I/O művel. befejeződik akkor azt fogja folytatni, nem a P_3-at

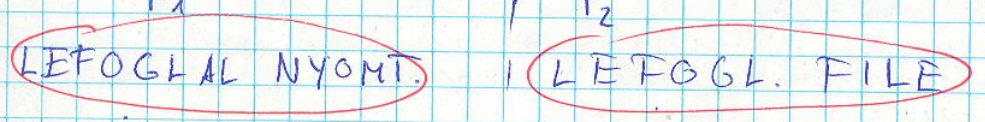
új problémák:

- emlékeztetnem kell a programmegszakítás állapotára
- áthelyezhetőségi megoldás
- programok amelyek párhuzamosan futnak, közöttük leíró elv szigetelés megoldása, összeakadásmentesítés (memóriagazdálkodás)

2008. 11. 04

- perifériakezelésben is új helyzet, egy másik progr. is elindíthatja a futó perifériát (periféria-várakozási sor kialakítása)
- koordináció az együtt működő programok között
  pl.: 2 program akar nyomtatni és az egyik lefoglalja a perif. amedolig nem megy el

használunk ilyen lefogl. eszk.



LEFOGLAL FILE ← itt várnak. kell a P_2-t amíg fel nem szabadul

: a P_1-nek kell várnia.

u. lesz. a file => BEAKADNAK <=

FELSZABADÍT FILE | FELSZAB. FILE

FELSZABADÍT NYOMT. | FELSZAB. NYOMT.

:

Ez a holtpont helyzet!

MULTICS

célkitűzés: multiprogr. számítógép csinálása, és végigvitte ki a házakhoz

mai INTERNET ötlete

- kötegelt feldolgozásnak nevezik a JGB-es feldolgozás formát (BATCH)

- időoszt. rendszer: a programozót ülteti le a számítógép elé, de több programozót ültetnek le elé

  ↳ a számítóg. majd összehangolja

  a programozó közvetlenül haszn. a számítógépet

  • parancssoros interfész (amíg írom addig minim. terhelem a gépet, az enter leütésére indít el parancsvégrehajtást)

  • minden felh. kap egy fix időintervallumot, és addig fut a programja, ha lejárt az idő másik progr. programját futtatja

  • problémák:

    -> kis memória, kevés program a tárban, lassú

- Real-time rendszerek: a számítógépet általában használják, érzékelni tud jeleket és válaszolni tud rájuk

  pl.: automatikai rendszerek

  /a számítógép többféle automatát megvalósíthat/

Mindhárom rendszer a multiprogr. révén tud hatékonyan működni

JOB: lépésekből áll és egymás után kötetlenül, a végrehajtás szekvenciális.
a job lépés végrehajtásának megfelelhető egy programnak

INTERAKTÍV: több szolg. ke pdk, a különböző felhasználók programjai futnak párhuzamosan

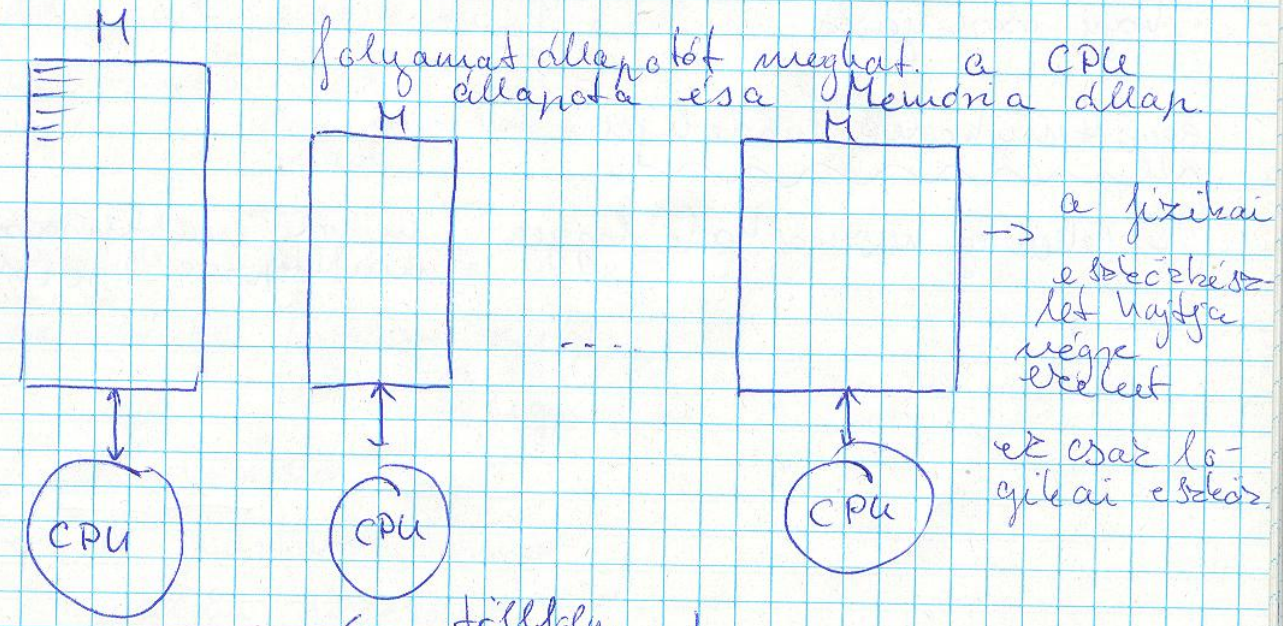
REAL-TIME: a külön programokat futtatjuk párhuzamosan (task)

// ↙

közös név folyamat
(felbontható apróbb lépésekre)

operác. rendszernél szükségünk a lépésekre
↳ programokat nézünk és hagyunk végre párhuzamosan
(a proc. átkapcsolgat a progr. között a várások alatt)

végrehajtás alatt álló folyamatnak nevezik az operációs rendszernél



- minden folyam párhuzamosan fut, asszimetri-kusan

2 féle kapcsolat:
- versengés (pl erőforrásért)
- együttműk. (közös feladat megold.)

Az oprendszer elosztja az erőforrásokat (jelen esetben memória, processzor)

-> memóriának fizikai blokkokban osztja szét

-> a processzort időosztással képezi le

a logikai proc. mincsenek benne a privilegizált utasítások

a folyamatok meghívhatják az oprendszer rendszerhívással elérhető funkcióit

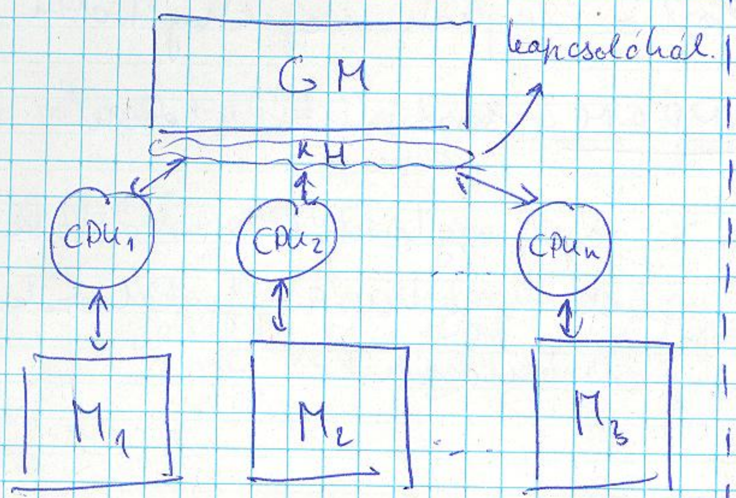
A folyamatok versengenek egymással, nem is ismerik egymást.

Ha együttműködnek akkor információt kell hogy cseréljenek

együttműködés modelljei

- lehetőleg használható legyen (mind multiprogr. mind multiproc. rendsz.)

KÖZÖS MEMÓRIA



- a címtart. egy részében ugyanazt a térrat látják
  ↳ ezen kereszt. információcs.

read (cím, adat)
write (cím, adat)

„primitív művelet"

→ ütközhetnek az írás olvasás műveletek
  ↳ meg kell mondanunk hogy mi sebbeelek

(kapcsolóhálózat felad. hogy az interferenciákat kivédje)

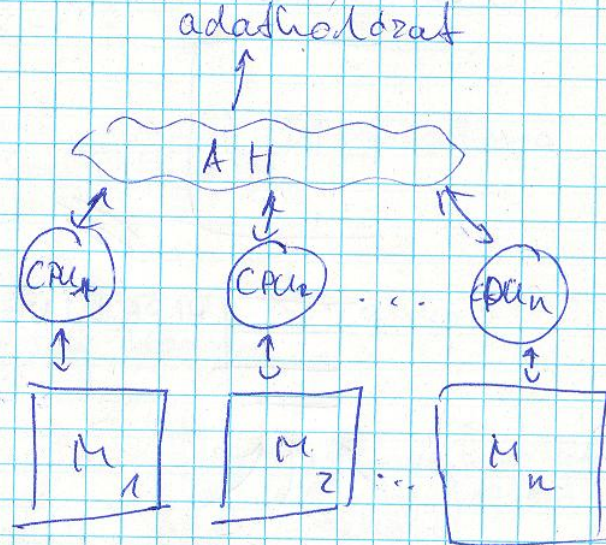
read - read | egyértelmű |

write - read | az a kérd. hogy mit vár a read |
↳ vagy az előz. v. az újat, de közbelsőt ne

write - write | valam. egyért. győztes legyen |
↳ ne keveredjenek!

megold. a párhoz „támad" valam. sorrendbe állítsa

ÜZENET - TOVÁBBÍTÁSOS

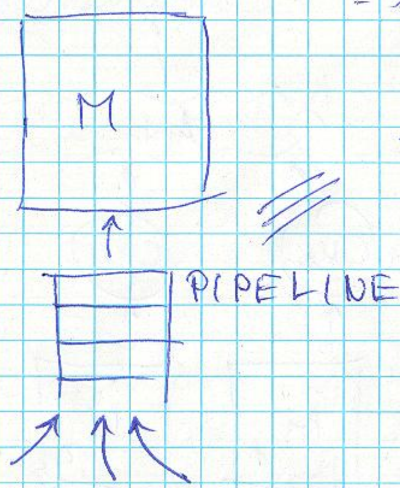


- egymást egyfajta I/O egys. látják

send
receive

↳ nem egyértelmű a paraméterezése

Sorbaállítás:



- sorba fognak végrehajtódni

PRAM: sorosított eléréssel ram

- végén egy kontroller döntötte el hogy kivel foglalkozik a memória

- most arbitráció, multimasteres sín

Adatcsere bonyodalma, nem biztos hogy az az adat van a kiolvasandó rekeszben amire szükségem van
↳ jelzőrekesz alkalmazása
(időbeli összehang. a folyam.)

2008. 11. 06

Szinkronizáció lényeges esetei

- precedencia (sorrendiség felállítása)

P_i P_j

~~write(j, "1")~~
write(m címre, x adat)
write(j, "1")
⋮

olyan prec. hogy a write előbb történj

⋮
read(j, r_j)
read(m, y)

Addig teszteld amíg 1-es nem lesz a j egy regiszt.

Egyik folyam. egy műveletet előbb hajtódjon végre mint a másik

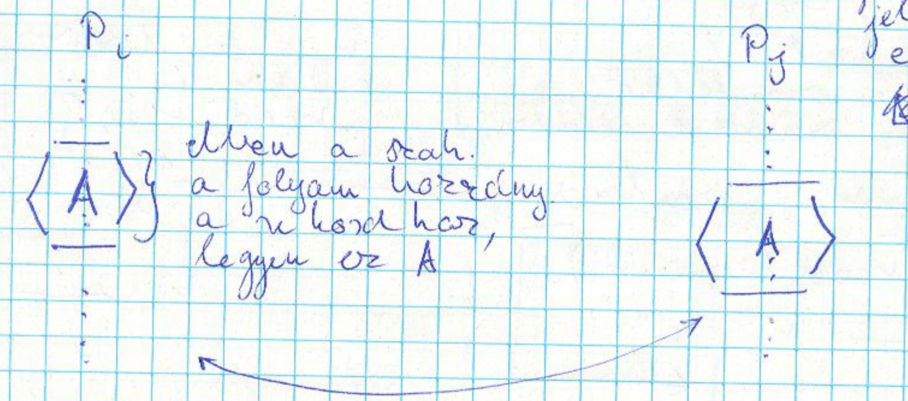
• szoftveresen megcsináltunk egy handshaking kapcsolatot egy kitüntetett regiszt. keresztül

A rekord akkor értelmes ha konzisztens a tartalma (összetartozó)

- kölcsönös kizárás

  • ha valaki elkezd hozzáférni a rekordhoz akkor addig más ne nyúlhasson hozzá amíg be nem fejezte

flag létrehozása amely jelzi e hogy foglalt-e a rekord



ezek a futási szakaszok párhuzamosan nem végre-hajthatók

- egyidejűség (randevú)

semmi nem védi meg attól hogy ez párh. hajt. végre

! összeakad!!

Dijkstra algoritmus mely a problémát megoldja (úgy hogy kiéheztetés sincs)

n → read (f, r_j)	n → read (f, r_i)
r_j = szab.	r_i = szabad
write (f, „fogl.")	write (f, „foglalt")
⟨kotor. (A)⟩	⟨kotorász. (A)⟩
⋮	⋮
write (f, „szab.")	write (f, „szabad")

2 folyamat kölcsönös kizárása **Peterson**

- a közös memóriában mindegy. van egy külön flagje
- belépési protokoll, illetve kilépési protokoll hogy ne mehessen és majd utána a többi is

f_i, f_j : flag
turn {i, j}

	P_i	P_j
Be:	write (f_i, "fogl.")	write (f_j, "fogl.")
	write (turn, "j")	write (turn, "i")
	read (f_j, r_i)	read (f_i, r_j)
	ha ez a felírő" szabad rendb.	r_j = szabad → i
	r_i = szabad → i	↓ n: read (turn, r_j)
	read (turn, r_i)	r_j = "j"
	n ← ; r_i = "i"	

Kilépés:

write (f_i, "szabad")
f_j

a 2. javaslat egyhézős vált. állít,
aki később billent az fog várni
→ 2 folyam. működik

2008. 11. 10

- hardware támogatás kell a közös flag-hez
↳ összetett, oszthatatlan utasítás
(nem ékelődhet közbe semmi)

Test and Set(F) utasítás

- flagkezelés
- kiolvassa az ered. értékét a flagnek és visszaírja egyben
  (csak a 1 -> foglalt
  Ø -> szabad)

használhatjuk függvényként
visszaadja a rekesz előző tartalmát

```
            begin
programozói     Test and Set := F;     } ez oszthatatlan
                F := FFh                 └> olyan mint a read és a write
            end
```

```
hardware    read(F, r)             ] oszthatatlanul
            write(F, "FFh")        ] hajtódjon végre
```

oszthatatlanság elérése a megszakítás maszkolásával

INTEL-nél LOCK-al könnyen megvalósítható
(másik proc nem tud hozzáférni a sínhez)

XCHG

- oszthatatlan exchange utasítás
- 2 paraméter (memória cím)
  - globális flag
  - lokális flag

  } értéket oszthatatlanul megcseréli

XCHG (LF, GF)

var TEMP

begin

    TEMP := GF;
    GF := LF;
    LF := TEMP

end

] oszthatatlan

F → közös flag

P_1 P_2

r := TestandSet(F)

r $\overset{?}{=}$ ∅ / ha ∅ mi még szabadnak láttuk /

(i)

⟨ matat ⟩

write (F, "∅")

- hardware - es támogatás!

LF_1 GF → ezt látja mindkettő LF_2

P_1 P_2

write (LF_1, "FFh")

XCHG (LF_1, GF)

LF_1 = ∅

(i) ⟨matat⟩

write (GF, "∅")

Ezzel megoldhatom a kölcsönös kizárást

A egyhelyben várós rontja a többi proc. helyzetét

↳ az egyh. járás a sín sávszéless. is foglalja

⇓

- folyamatok várakoztatása, aktív várakozás kiküszöbölésének megoldása

  / pl: op.rendszer /

- véletlen dönti el ki lép be a ciklusba

  ↳ fair ütemezés kezelése

  / véges időn belül mind hasznudja /

3 féle szinkronizációs igény

↳ ezek egy eszköz kitalálása

⟹ SZEMAFOR

Szemafor

- 2 értéke, vagy szabad vagy tilos.

var S: integer := k; ← kezdeti érték / hanyan férnek bel

P: begin

↑ P művelet

while S < 1 do skip;
S = S - 1;

] / ez egy ciklus, elválaszthatatlan! /

end

V: $S := S + 1$, oszthatatlan!

↑ V művelet

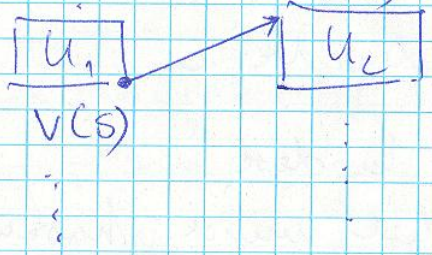
V-vel jelezhetjük a továbbhaladást hogy mi átmentünk

Bináris szemafor $S: \{\emptyset, 1\}$ ← kezdőértéke $\emptyset$ v. 1 lehet

- Precedencia

P_1 P_2

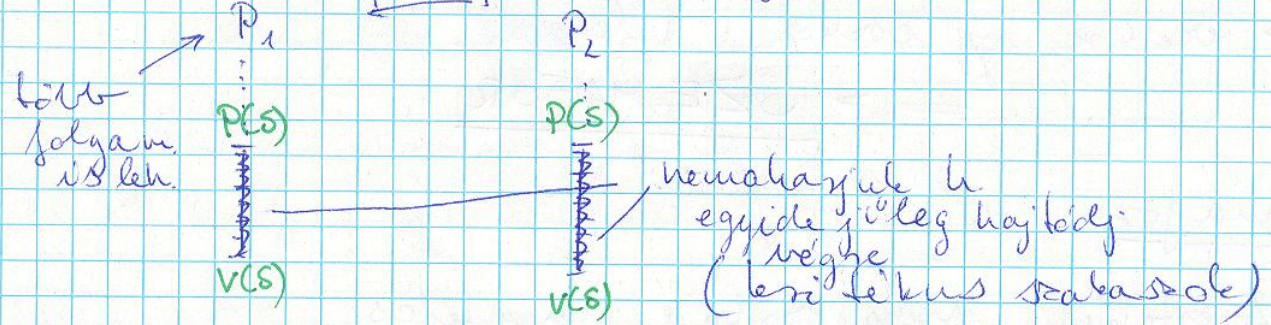
$S := \emptyset$ — szemafor



ha a P_2 hamar ér az U_2 hez akkor vár amíg a V(S) végre nem hajtódik

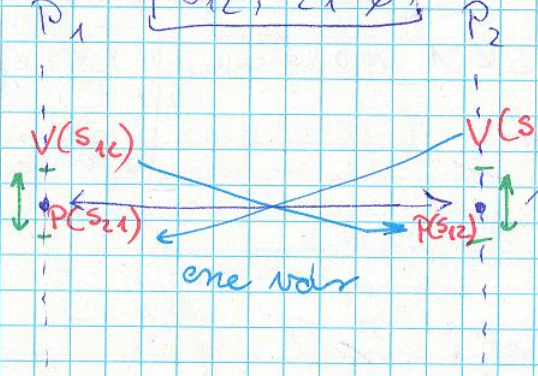
- Kölcsönös kizárás:

$S := 1$ — szemafor



- Randevú

$S_{12}; S_{21} := \emptyset$



intervallumon belül legyen. egyid. végeh.

ha megérk. az inter. akkor szólok a másiknak és várok

Séma for megvalósítás multiprogr. rendszerben:

- ha multiprogr. akkor vannak benne folyamatok
  - → vannak olyan rendszerh. amik „elvégzik" a folyamatokat

folyamatkezelő

sleep;
wake up (p_ID);

lista: list of ..

jelző (l, p_ID)
lefűz (l); p_ID

↑ függvény amely egy értéket visszaad

var s: record of érték: integer; (=1 → ha bináris)
l: list of p_id;
end

procedure P (s):

osztható-tatlan!

begin
  s.érték := s.érték - 1;
  if s.érték < 0 then begin jelző (s.l, <magam>); sleep end
end

procedure V (s):

osztható-tatlan!

begin
  s.érték := s.érték + 1;
  if s.érték < 1 then wake up (lefűz (s.l));
end

2008. 11.11

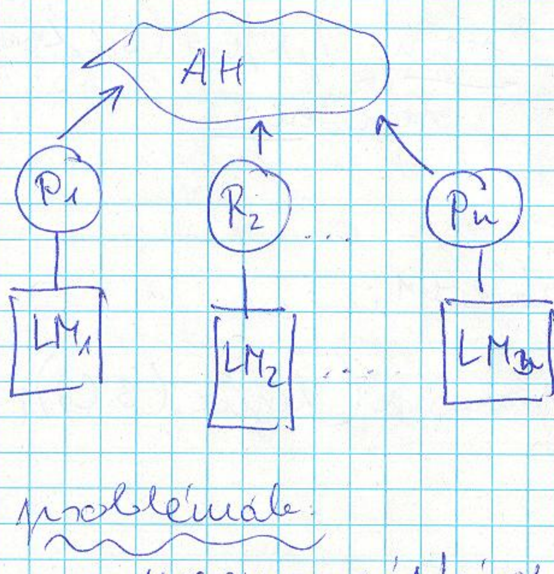
Esemény

- jelzem h. megtörtént
- várakozok rá hogy megtörténjen

Lehet várni egy szemafora is, de csak egyvalaki mehet tovább. Az eseményt mindenki egyszerre érzékeli, egyszerre indulnak tovább.

Ha egy esem. elk. várakozni akkor a kezdeményező aktiválására várok, míg szemaforon, ha szabad akkor egytől elhaladok rajta menni

Üzenet - továbbítás



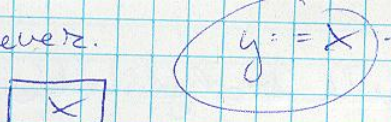
send (...)

recieve (...)

problémák:

- megnevezés kérdése / hogyan szólítják meg egym. /
- szinkronizációs kihatások
- műveletek szemantikája / mit is jelent hogy küldtem üzenetet... /

1. Megnevez.



ha egy memóriaterületre lenne

x

Pi

y

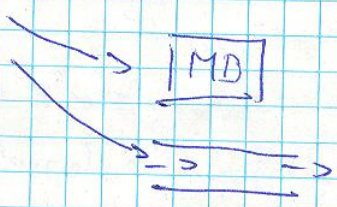
Pj

Direkt send(Pj, x) receive(Pi, y)

megnevezem közvetlenül

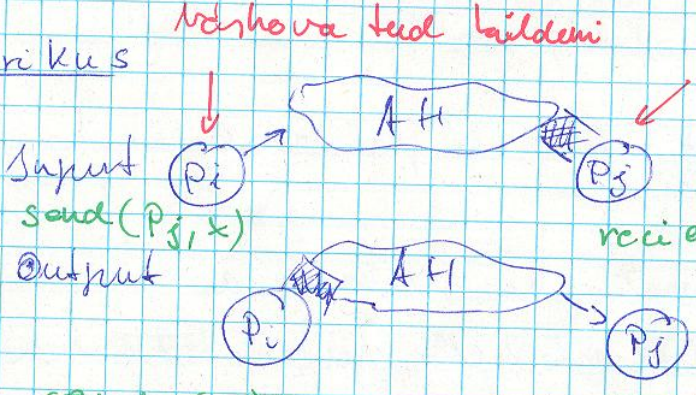
- direktbe megmondom kinek küldok, és kitől vegyek
  - L> „Marconi effektus" (nem vehetem át másik alkalmazásba)

Indirekt send(mailbox csatorna, x) receive(mailbox csatorna, y)



- Pi nem látja a másik oldalt, nem kötve magával a többit, átvehető másik alkalom.

Aszimmetrikus



output portos eset: mindenféle szolgáltatás
(pl. internet)

kliens → szerver

output portos eset: nem érzékeli hogy
ki veszi el

Csoportcímzés

- nem egy partnernek akarunk üzenetet küldeni hanem többnek
  (speciális eset ha mindenkinek, ez a Broadcast)

2. Szinkronizációs hatások

- együttműködés esete hogy üzenetet küldözgetnek
  (egy üzenet hamar kiküldődik el mint a fogadás megtörténjen)

- pufferelésen alapuló szinkronizáció kényszerít ki a küldő és a fogadó között

- pufferelt eset lesz a precedencia teljesül

- véges puffernél ha betelt a küldőnek várnia kell, blokkolhat a küldő

- végtelen puffer, csak a fejlesztők ← ez kényelmesebb
  gondolják hogy van, az üzenetküldőnek nem kell várni → adatvesztés kockázata!

3. Szemantika

Mi is tört. a művelet során?

- lehet károsodott az adat, tranziens
  ↳ újra elküldés

Broadcastos send:

- mindent elküldöm szinte egymenetben, várom a nyugtát
  ↳ pl. h. kap esol. az üzenetvételek

Nyelvi konstrukciók → letölth. alva sorány.

Holtpont helyzet

P_i	P_j
RQST E_1	RQST E_2
RQST E_2	RQST E_1

Fennakadnak nem tudnak ~~tovább~~ menni

P_i	P_{i+1}	P_{i+2}
recieve (P_{i+1})	recieve (P_{i+2})	recieve (P_i)
send (P_{i+2})	send (P_i)	send (P_{i+1})

- üzenetváltással is össze l. akasztani folyamatokat

def: A rendszerben vannak folyamatok.

A rendsz. alkot holtp. egy részhalm. ha minden halmazbeli elem olyan eseményre vár amit csak kizárólag az adott részhalmazban lévő elem hozhat létre

RQST → erőforr. lefoglalása
(addig vár amíg ki nem tudják szolgálni)

RLSE → lefoglal. feloldása

erőforrások tulajdonságai

- menthető-e az állapota és visszaállítható-e automatikusan

  pl.: processzor, memória

- egypéldányos v. többpéldányos

  memória

  a kérése egyszerű (megnevezem)

  RQST E_1

  fajtánként kérhetek tetsz. mennyis.

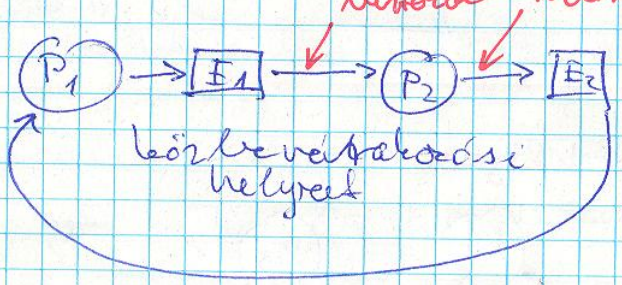
  RQST(R_1 X, R_2 Y...)

  - csak akkor hajtható végre ha valamennyi erőforrást megkaphatom

Holtpont kialakulásának

Szükséges feltételek:

- legyen kölcsönös kizárás
  (olyan erőf. ami nem megosztható állap.)
- Hold + wait
  (olyan folyamat, amely erőforrásokat erőf. és vár egy másikra)
- erőforrásokat nem vesznek el senkitől erőforrást
- Circular wait



+ TK-ból:
- erőforrások nem preemptálhatók jellege

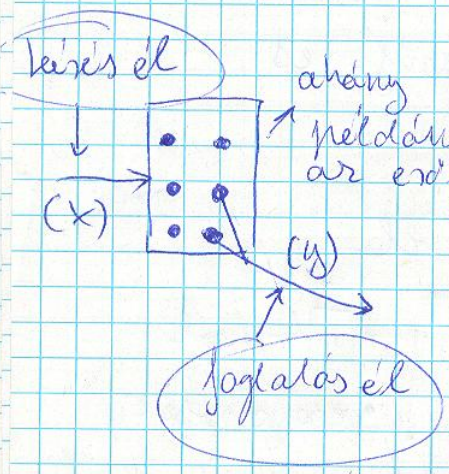
Holtpont kezelő lehetős. stratégiák

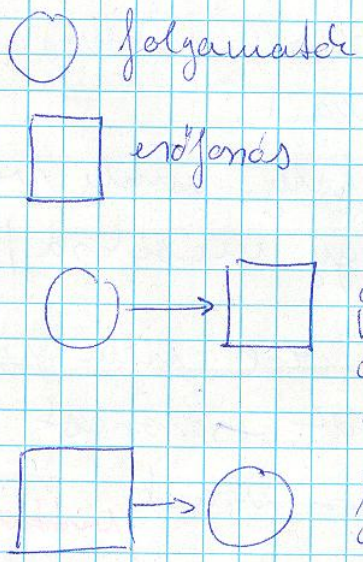
1. Strucc-algoritmus
   - nem vesszük tudomásul a problémákat
     ↳ kár csökkentő mint a megelőz. fordított összeg
2. holtpont megelőzése
   - előrelátó időben elkerül, védekezés, futási időben nem kell vele törődni holtpont monitorozás tervezés
3. holtpont elkerülése
   - dinamikus erőforrás kiosztás, futási időben történik
     ↳ dinamikus
4. holtpont észlelése és feloldása
   - „esemény utáni"

2008.11.13

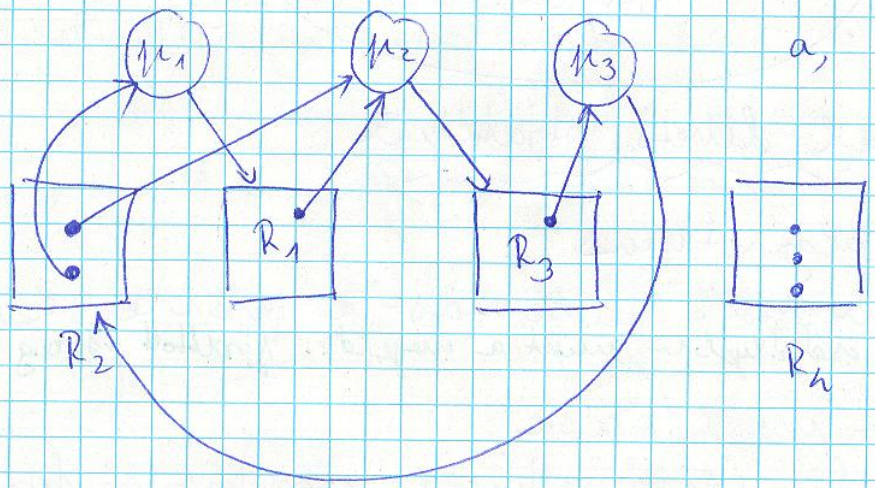
Erőforrás-foglalási gráf

- irányított gráf





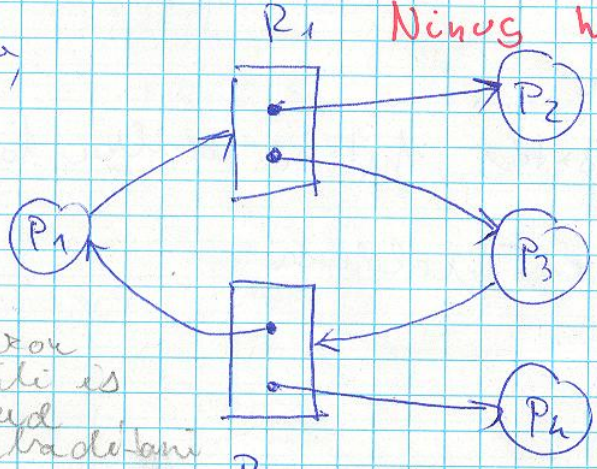
pl.: **Holtpont helyzet**



a,

Nincs holtpont

b,



halmazok kérési és felszabadítási

- ahogy a rendszer működik folyamatosan változnak a gráf adott pillanatnyi képe ha a rendszert

egypéldányos esetben a kör megléte szükséges és elégséges feltétele a holtpontnak

2. Holtpont megelőzése

- tervez. fázisban akarunk olyan rendszert létrehozni amiben nincsen holtpont
  - kölcsönös kizárás jelenléte
    ↳ ha ez száműzhető akkor nincs holtpont
      / ilyen rendszert nemnagyon tudunk tervezni /
  - Hold and wait
    ↳ egyszerre kér minden erőf., és egyszerre az összeset akkor holtpont helyzet
      / ám ez erőforrás hatékonyság vesztést jelent /
  - No preemption - nincs erőszak
    ↳ ha nem tartom be kiküszöbölhető a holtpont
    ↳ menthető állapotúnak kell lennie az erőf.
      / ha nem menthető állapotú, akkor meg kell tudni oldanom hogy újrakezdje a munkát
      Roll back pontok /
  - Körkevárakozás
    ↳ ha [redacted] teszteléskor kilőhető akkor jó
      / addig lehet várakozás amíg nincs visszacsatolás /
      - minden folyam. [redacted] erőforrás sorrend
      - kérés csak nagy. sorsz. erőforrásra lehetséges mint ami a folyamatnál van

3. Holtpont elkerülése

- futási időben történő megoldás

a, többpéldányos

	Max	Birtokol	Kérés
P_1	6	4	–
P_2	3	2	1
P_3	9	2	1
Szabad		2	

10 db

	Birtokol 1	Kérés
P_1	4	4
P_2	3	
P_3	3	6
	~~0~~	

	Birtokol 2	Kérés
P_1	4	1
P_2	3	✓
P_3	2	~~7~~ |1+6|
Szabad	1	4

→ tudjuk úgy osztog. az erőf. hogyne alakulhasson ki holtpont

biztonságos - a legrosszabb esetre is van olyan lefutási eset hogy minden be tud fejeződni

nem biztonsági állapot

Az erőf. kiosztó. az a feladata hogy biztonságos állapotban tartsa a rendszert

Bankár algoritmus | Dijkstra |

P . n — a folyamatként kiadott mátrix
R . m erőforrások (FOGLAL : n × m)

FOGLAL$_i$: m vektor

MAX : n × m mátrix
(megjelentett maxim. igények)
MAX$_i$: m vektor
SZABAD : m vektor

MÉG = MAX − FOGLAL

ezekből már megtudunk jósolni egy legrosszabb esetet
KÉR n × m mátrix
KÉR$_i$: m vektor

KÉR$_i$ kiszolgálható-e?

① érvényes kérés-e? (nem lép-e ki a lejelentett max. mennyiség.)

KÉR$_i$ ≤ MAX$_i$ − FOGLAL$_i$ —(n)→ valami hiba (ABORT)

↓ (i) (minden esetben igaznak kell lennie)

KÉR$_i$ ≤ SZABAD —(n)→ várni kell

↓ (i)

② szimulálom a kiszolgálást
FOGLAL$_i$:= FOGLAL$_i$ + KÉR$_i$
SZABAD := SZABAD − KÉR$_i$
biztonságos-e? —(n)→ visszacsinál:
FOGLAL := FOGLAL − KÉR$_i$
SZABAD := SZABAD + KÉR$_i$
(várj)

(i) ↓
ODA AD

Biztonságos-e?

GYŰJT : n vektor

LEFUT : n vektor (Boole)

① GYŰJT := SZABAD

LEFUT := FALSE

② Keres i-t

MÉG i ≤ GYŰJT ∧ LEFUT[i] = FALSE (és)

van

LEFUT[i] = TRUE

GYŰJT = GYŰJT + FOGLAL_i

nincs

igaz minden i-re

LEFUT[i] ∧ i-re?

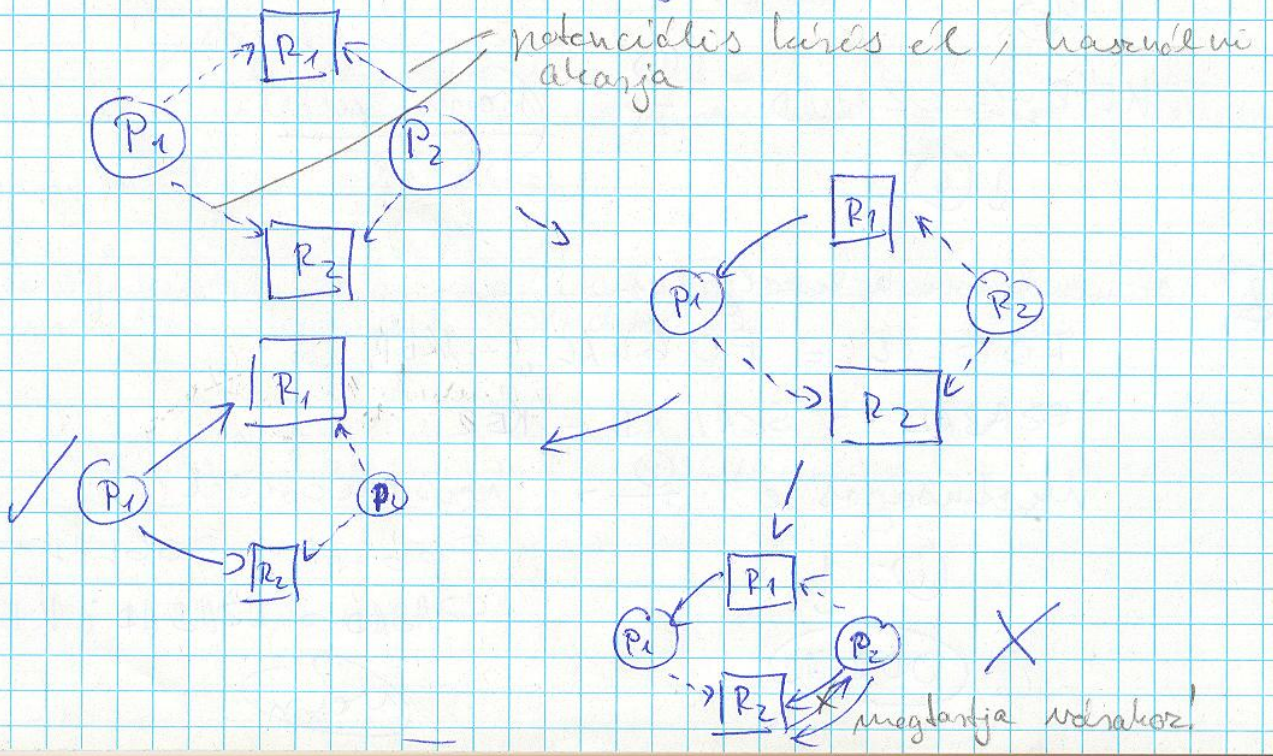
nem → nem biztons.

igaz → biztons.

N, egypéldányos

- elég ha vesszük az erőf. gráfot, és ott a körökből kikeresjük

potenciális kérés él, használni akarja



2008. 11. 18

Shoshan - Coffman

- itt nem kell maximális igény kiszámítani

FOGLAL n x m

KÉR n x m

SZABAD n

1. Feltesszük egy folyamra, ami abból elégíthető ki az erőforr.

Inicializálás

TOVABB n Boole

GYŰJT

1, TOVABB := false

GYŰJT := SZABAD

2, Keres i-t

KÉR$_i$ ≤ GYŰJT ∧ TOVABB[i] = false

(mindenkit csak egyszer lehet megfaládni)

van

TOVABB[i] = false? (ez itt szerintem TRUE!)

GYŰJT := GYŰJT + FOGLAL$_i$

nem

TOVABB[i] ∀ i-re?

i

NINCS HOLTPONT

n

VAN HOLTPONT

↳ azok számnak holtpontban akiket nem próbáltunk ki

Mikor érdemes végrehajtani az algoritmust

(1) minden erőforrás adás után
↳ nem célszerű

(2) automatikusan x időnként

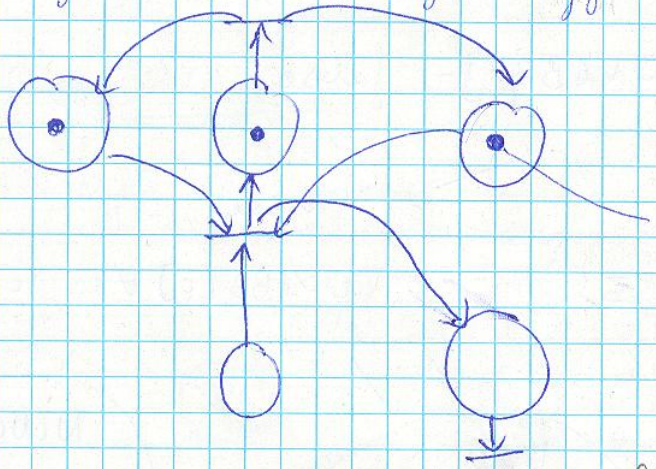
(3) manuális indítás

Ha a rendszer holtpontban van nem oldható meg roncsolás nélkül (minimális veszteséggel járhat.)

→ nem menthető állapotú erőforrások kerülése
(roll-back módszerrel érdemes megoldani)

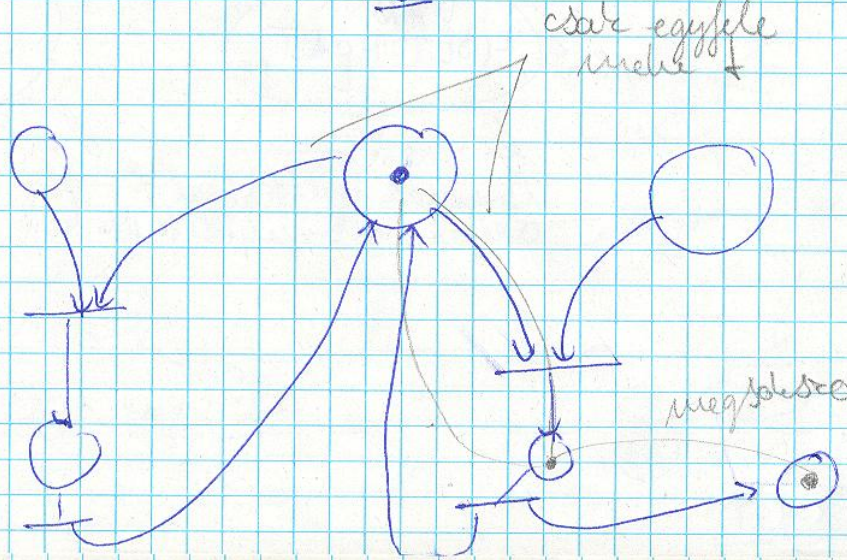
Petri hálók

- olyan modell amelyik egy irány. gráf



- lehetséges változásait átmenetek

Ha akkor a bemenete eltűnnek és a kimeneten megjelennek



Feladat:

$R_1 = 12$ $R_2 = 35$ $R_3 = 8$

	R_1	R_2	R_3
P_1	1	2	0
P_2	1	8	1
P_3	1	4	2
P_4	2	2	2

MAX		
4	13	2
6	21	6
5	11	3
8	28	4

MEG		
3	11	2
5	13	5
4	7	1
2	23	1

(1) (2)

R_1 R_2 R_3 kéthez még

HOLT PONT

$P_4(4, 3, 1) \Rightarrow$ kéréssel elkerülé

FOGLALT	5	16	5
SZABAD	7	19	3
SZABAD'	3	16	2

→ ez lenne ha kiszolg

(1) kérés lehetsége?

(2) kiszolg.-e a kérés

(3) eddemes-e vele foglalk. kérésig marad -e!

Van-e holtpont?

$R_1 = 24$ $R_2 = 35$ $R_3 = 8$ HOLT PONT

	FOGLAL R_1	R_2	R_3	KÉR R_1	R_2	R_3	
P_1	3	2	2	7	16	3	(2)
P_2	2	8	1	4	26	1	X
P_3	21	2	1	0	0	0	(1)
P_4	2	4	2	7	24	3	X
ÖFOGL.	8	16	6				
SZAB.	16	19	2				

ez semelyik sem lesz elég!

20 23 5

17 21 3

ennyi lesz ha jólesz

↳ csak a P_3-nak elég

- 1 példány. elvj.

P_1	P_2	P_3
RQST (R_3)	RQST (R_2)	RQST (R_3)
RQST (R_2)	RLSE (R_2)	RQST (R_2)
RQST (R_1)	RQST (R_1)	RLSE (R_2)
RELEASE ($R_1 R_2 R_3$)	RQST (R_2)	RQST (R_1)
	RLSE ($R_1 R_2$)	RLSE (R_1, R_3)

juthat-e holtpontba?

foglal. → sorrend

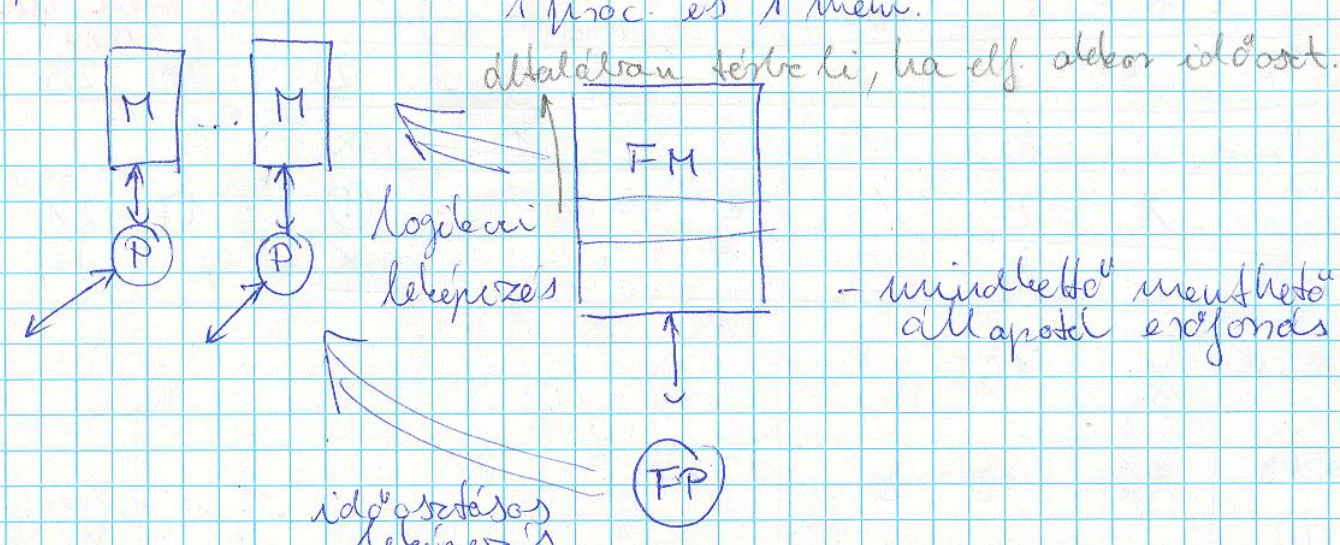
3 - 2 - 1 — 1 - 2 — 3 - 2 — 3 - 1

ciklikus várakozás

2008. 11. 20

Multiprogram. operációs rendszerek

állapotmodell



→ a memória állapotmentése a hőkötés segítségével történik

→ processzor állapotmentése
(utasításszámláló + egyéb regiszterek)

hatékony folyamatváltás ha több folyamat egyszerre helyezhető el a tárban, nem kell újra betölteni a memóriába.

folyamat szintű proc. → nem hajthatja végre a privát utasításokat
de hozzátartozik az operációs különböző szolgáltatásai
↳ ezeket az állapotjelző-
ket is mentenie kell

kontextus
("context switching")

CPU ütemezési problémája

- mikor váltunk kontextust

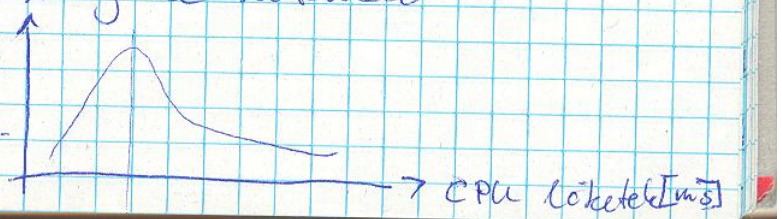
CPU löket (új program indít. ezzel jár)
I/O löket

(exit rendszerhívás, végzett a program)

ha a CPU löket befej. jelen egy ütemezési feladat
↳ az I/O művelet a proc. nélkül végbemenet, itt kell kontextust váltani addig egy CPU löketes feladatot ütemezhetünk

Kinek adjuk oda a CPU-t?
↳ ezt a döntést hozza meg az ütemező

vizsgáljuk a CPU lökethosszakat

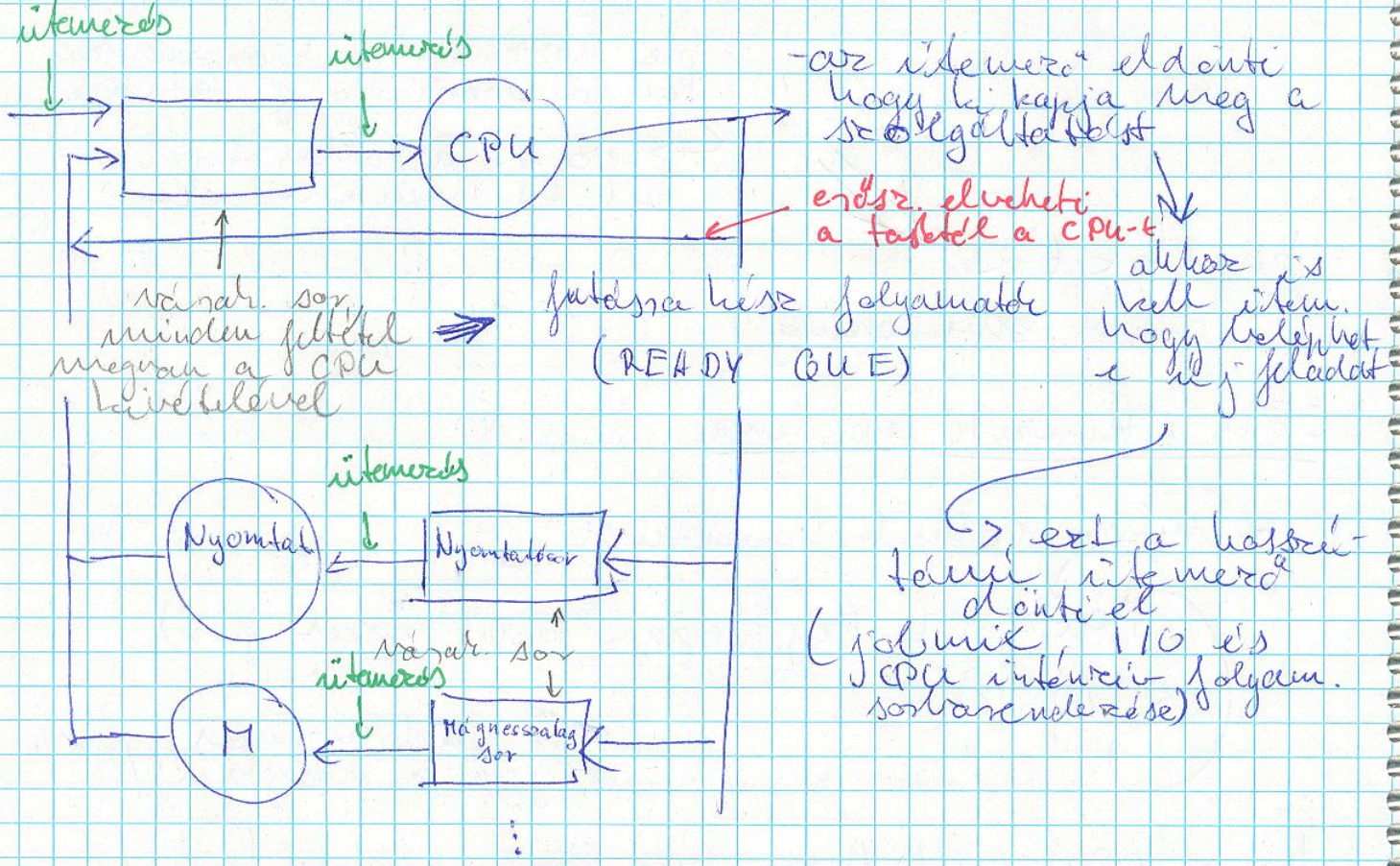


a perifériaidőkhöz képest viszonylag rövidek a CPU löketei

↳ gyors kontextusváltás szükséges, kicsi adminisztrációs idő aránt az ütemező oldal.

1. Sorállási modell

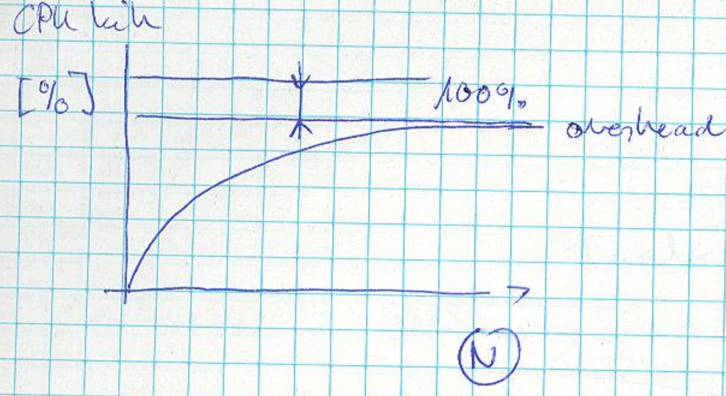
- a nagy erőf. egyszerre egyféleképpen használhatja, (sokan szeretnek várakozni minden egyik erőf. várakozási sor ↳ adminisztráció igénye)



multiprogramozás lóka: azon folyam. száma akik kontextusnak a rendszerben (erőf. versengő folyamatok)

#P : (N)

"hány job van a rendszerben"

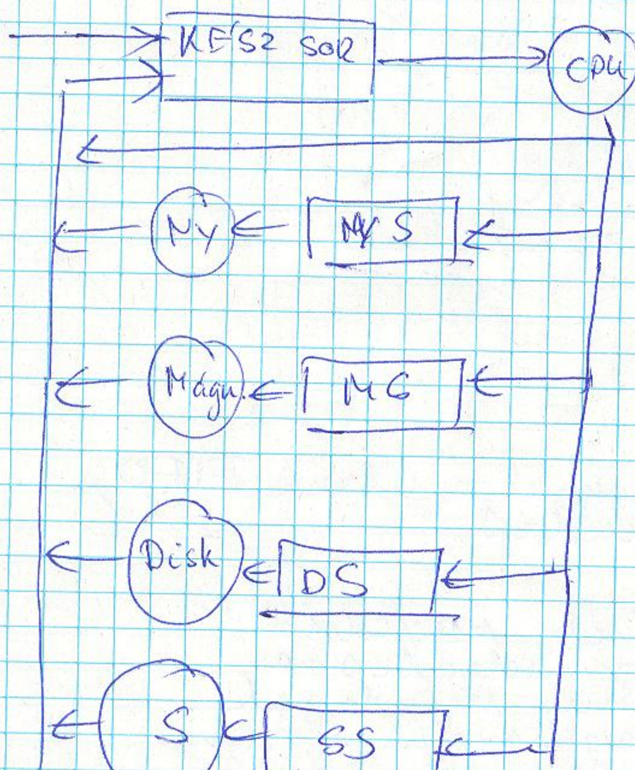


/ oprendszer futtatása hulladékidő

- minél többen vagy a görbe annál inkább közelít

- a szemaforokhoz is rendelhető várakozási sor, ugyanúgy beépíthető a rendszerbe

teljes modell



- mikor kezeli a rendszerek megl. a tárat / ez a memória kezelés szemlélete

↓

ez akkor gond ha elfogy a tár

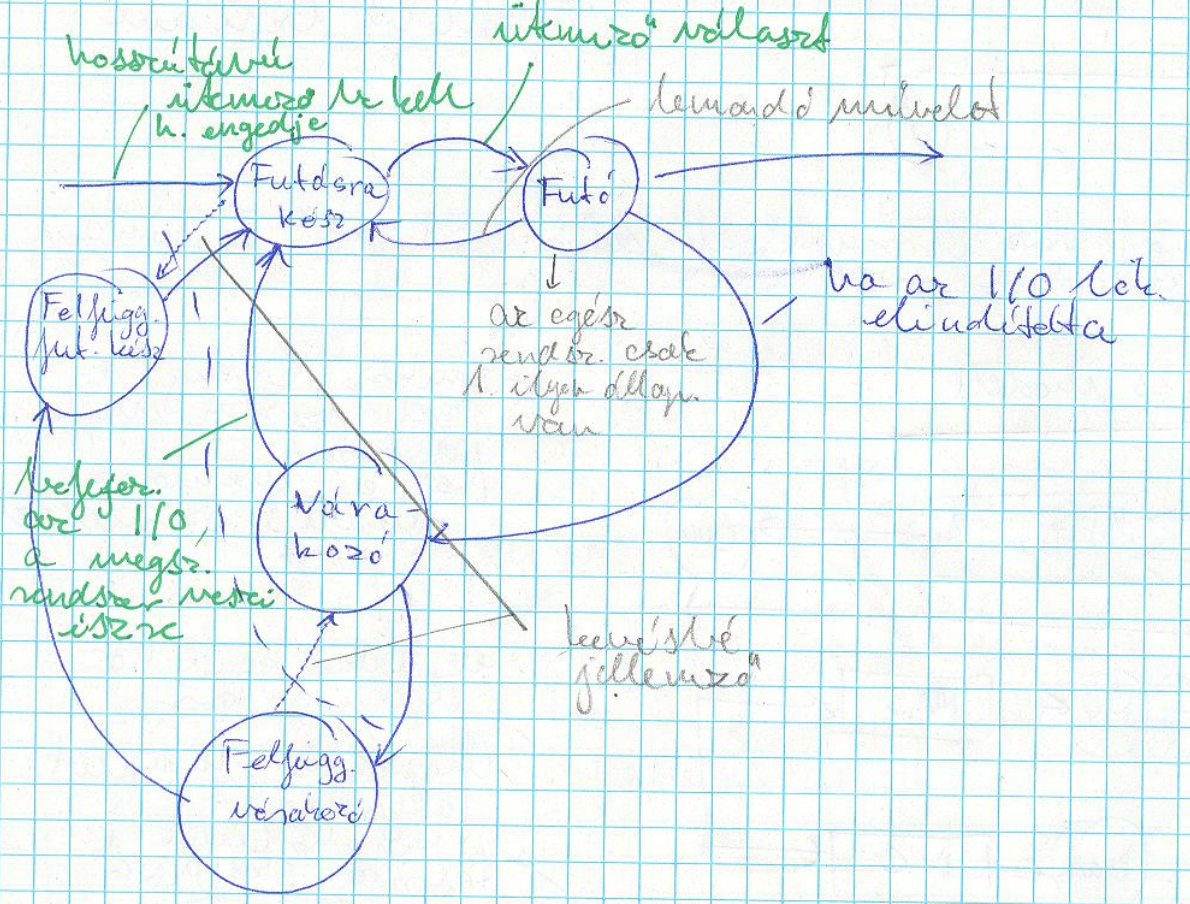
(megoldás, komplett folyamatot kipakolunk a tárba SWAPPING

└> felfüggesztem)

└> átmenetileg lecs. a multiprogramozás fokát

Memória! < felfüggesztés / középtávú ütemező foglalkozik ezzel a feladattal

2. Állapotmodell



a periféria ütemezés legegyszerűbb változata FIFO, kiterjedt specialis

hosszútávú ütemezőnél megfelelő folyamatmix, akkor ideális a rendszer kihasz. ha mindegyik csak műveletvan terhelt

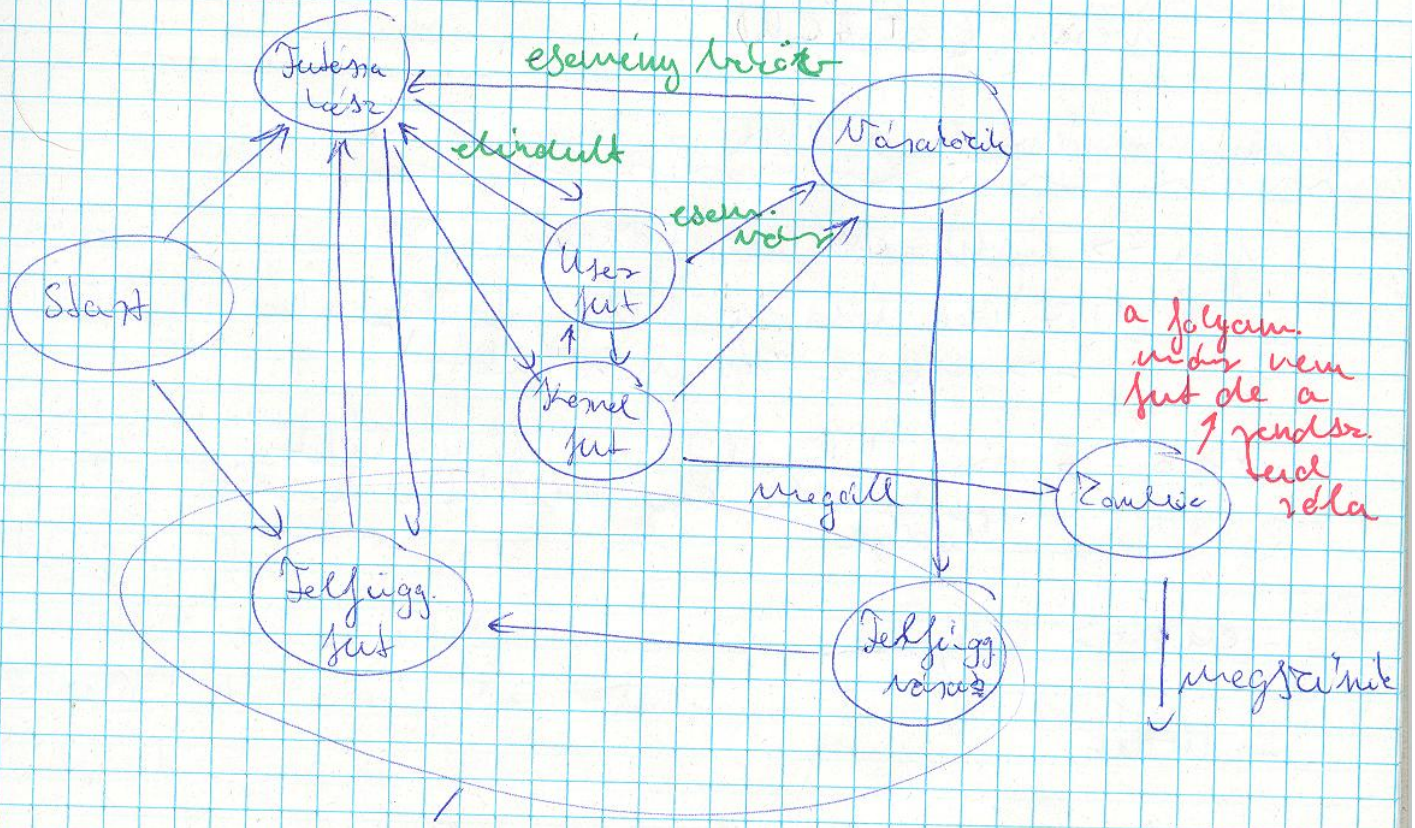
↳ nagy áteresztőképesség

! tudni kell mi az I/O igényes-e vagy CPU igényes-e !

2008. 11. 24

UNIX operációs rendszer

- folyamatok helyek állapotokban



valahol tárolni kell a táblán

2 kitüntett adatszerkezet

folyamat USER módban való futtatásához szükséges
- program kódja
- adatterülete
- kernel szintű leírója
- folyamat többi regiszter tartalma

folyamat ütemezéséhez nyilvántartott infó (mind a táblán)

szükség folyamat kell, de nem kell mind a táblán lennie

- u terület (mindig a táblán)

a, a program kódja
- osztható kód
- osztható elinam. kezigyintés
- nem osztható kód

b, adatterületek
- inicializált adat (DATA)
- nem inicializált adat (BSS) → elinam. növelik.
- verem (STACK)

Klasszos rendszerek. a UNIX a folyamatokkal.

- fork -> új processz létrehozása
  | ekkor nem kell be semmi futási kódot
  L> csinál egy másolatot)

- exec -> az aktuális futó folyamatot kicseréli egy diszken tárolt kódra

- exit -> kilépés

- wait

- vfork -> a folyamat megduplázása felesleges, ezt kiszűrhetik ki

- sleep -> adott ideig vár

ősfolyamat (init)

Processzek közötti kommunikáció

- signal (asszinkron kommunikációs mód)
- pipe (kommunikációs csőrendszerek)
- megosztott memória
- üzenet sorok
- szemaforok
- socket (elősz. a hálóz. kommunik. találhatók)

2008. 11. 25

összes folyam leír. szükséges adat tárolása

PCB process control block

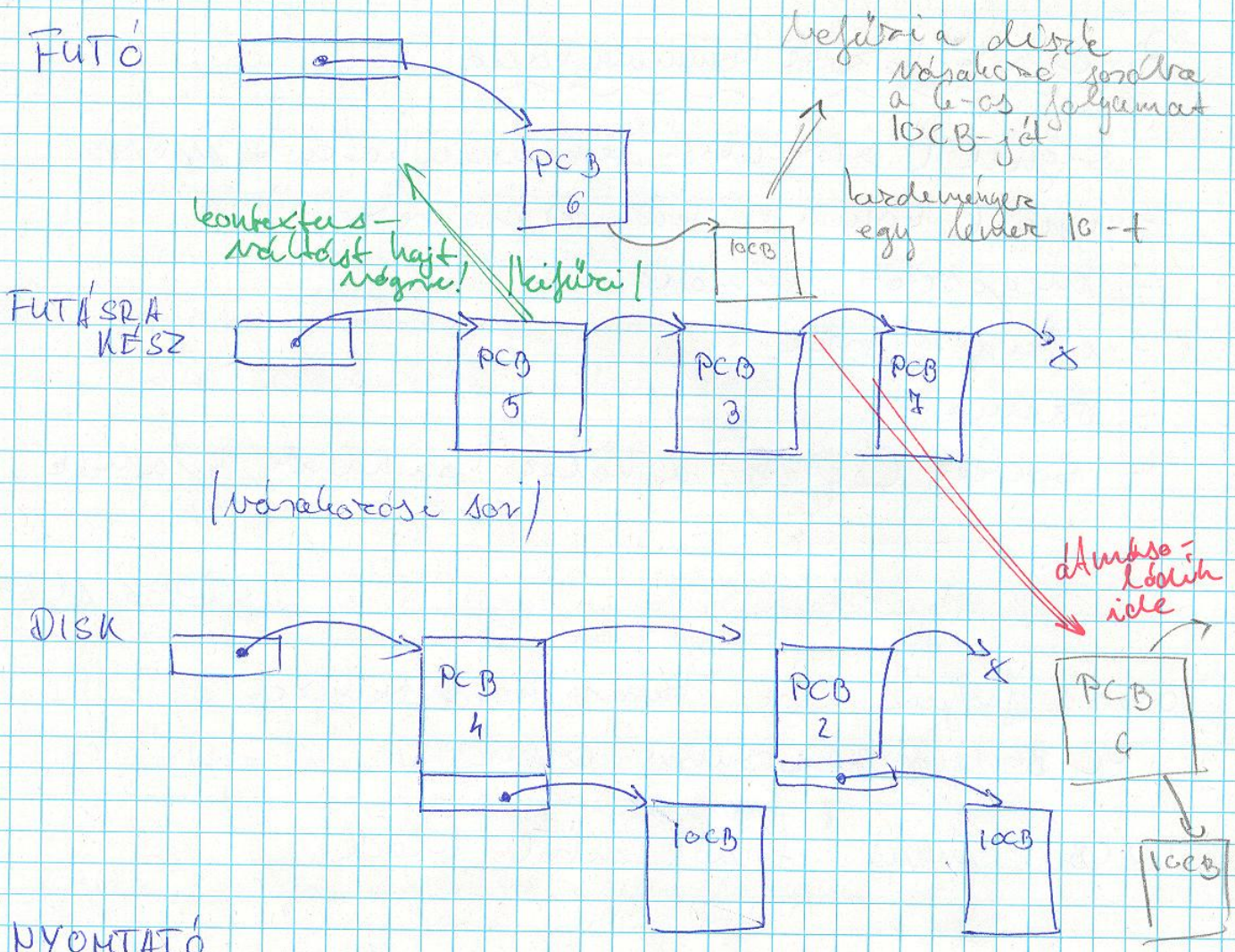
IOCB input, output control block

PCB

- azonosítója a folyam
- hely ahol a foly. állapotát tárolni kell
- kontextus tárolás

IOCB

- művelet / parancs meghatározása
- memória cím, hol (sor puffer cím (innen ír. az átvitel))
- adathossz
- lehet olyan is amit ámeresz kell
- információ hogy az átvitel hogyan ért véget



Lefoglalja a diszk várakozási sorára a 6-os folyamat IOCB-jét

Kezdeményezi egy lemez IO-t

Átmeneti tárolás ide

→ ütemező kiválasztja egy futásra kész folyamatot

perifériák várak. sora

- fifo kiszolgálási sor (néha a lemez kivétel)

Ha egy IO várak. sorban végrehajtódik, pl. PCBk feladása után visszakerülnek a futásra kész sorba! Majd folytatják a várakozási sor következő kérésének kiszolgálásával

ütemezési algoritmusok a rendszerekben

1. FIFO-elv

- a futásra kész sort az érkezési sorrendnek megfelelően szolgálják ki

FCFS (FIRST COME FIRST SERVED)

2. SJF algoritmus (SHORTEST JOB FIRST)

- kit lehet a leggyorsabban kiszolgálni elv
  (pl.: posta és a „zsákos" feladó)

GANTT diagram az ábrázoláshoz

ideális esetben így

CPU löket

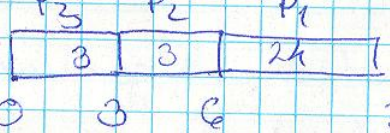
P_1	24
P_2	3
P_3	3

FCFS-el

sorban érkeznek

$$\text{Átl. átfut} = \frac{(24-0)+(27-0)+(30-0)}{3} = 27$$

SJF-el



$$\text{Átl. átf.} = \frac{(3-0)+(6-0)+(30-0)}{3} = 13$$

Becslés! → Előrejelzés → exponenciális átlagolás

- kiéheztetés problémája
  (pl.: mindig jön egy „csekkes")

t mért

$\tau_{i+1} = \lambda \cdot t_i + (1-\lambda)\tau_i \quad 0<\lambda<1$

τ becsült

- minél kisebb szám a mérés annál kisebb az együttható
- exponenciálisan felejti az értékeket, de könnyedén megteremti őket határozza

Prioritás!

- éheztetést idézhet elő

Éheztetés!

(kiszorításos) Preemptív algoritmus → a fontosabb folyamat azonnal hajt. végre

Nem preemptív

↳ ha a processzor kihasználtsága a fontos ezt használjuk

az éheztetés megoldása lehet:

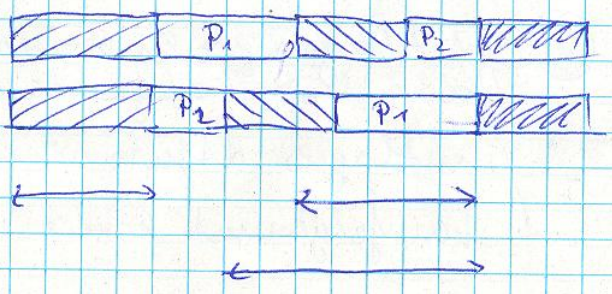
-> a régóta várakozó folyam. prioritását folyam. növeljük

Round-Robin algoritmusok:

- időszeletelése alapon (mindenki egyforma joggal!)
- időszelet mérete tartozik hozzá (80%) ← a CPU löketek 80%-ig férjen bele
- eredendően FIFO kiszolg., hasonlít a forgóprioritásosra
- a kontextusváltás ideje kritikus

SJF optimális Várj. átlag

1.



az átlagos várakozási idő optimalizálja.

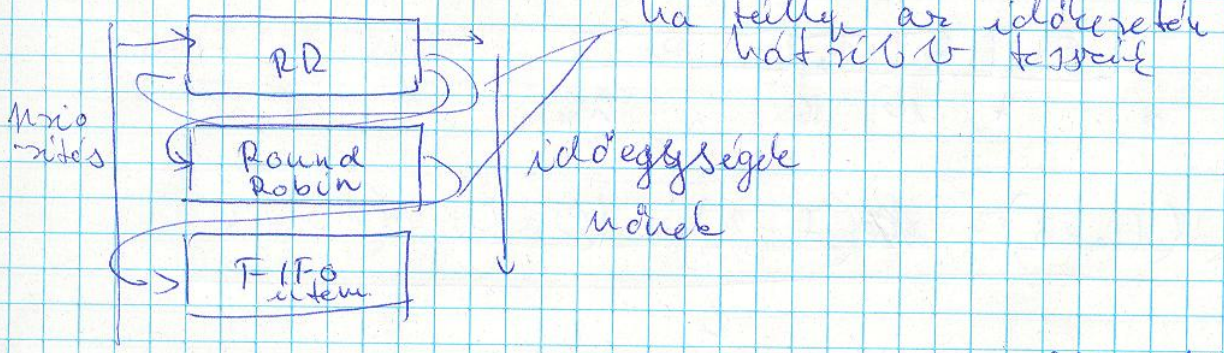
Shortest Remaining Time First algoritmus

SRTF

prioritási sor alkalm.

- a folyam. elosztanak több sorba
  => többszintű sorok

többszintű visszacsatolt sorok



2008. 12. 02

Feladat

időpillanat

nagyobb a fontossága kis, de az feladat függ"

	Érk. idő	CPU löket	Prioritásszám	Áthangszerelt várakozás
P_1	3	10	3	4
P_2	2	3	1	1
P_3	0	7	4	3
P_4	5	6	2	2

FCFS
SJF
SRTF
PP
RR (n időegységes)

⇑

ütemezési algoritmusok

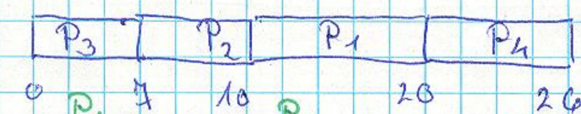
① Gantt diagram

② Átlagos válaszidő? (különböző alg. esetén)
Átlagos átfutási idő (a folyamat érkez. a befejez.)

a beérkezésétől az első CPU löketig tartó idő átlaga

FCFS

FIRST COME FIRST SERVED



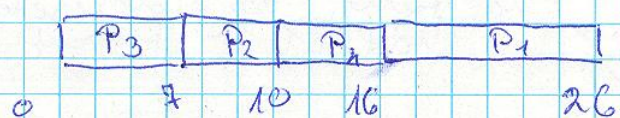
$$\bar{t}_{vál} = \frac{(10-3) + (7-2) + (0-0) + (20-5)}{4}$$

(P1, P2, P3, P4 above the terms)

$$\bar{t}_{átf} = \frac{(20-3) + (10-2) + (7-0) + (26-5)}{4}$$

SJF

SHORTEST JOB FIRST

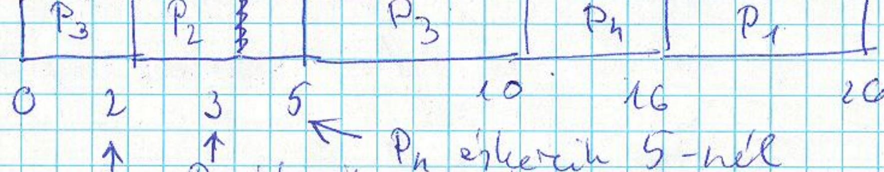


$$\bar{t}_{vál} = \frac{(16-3) + (7-2) + (0-0) + (10-5)}{4}$$

$$\bar{t}_{átf} = \frac{(26-3) + (10-2) + (7-0) + (16-5)}{4}$$

SRTF

SHORTEST REMAINING TIME FIRST



↑ 2: jön a P2 (cseréljünk-e?)
↑ 3: P1 érkezik
← 5: P4 érkezik 5-nél (3 folyamat van jelen)

a hátralévő idő alapján döntünk

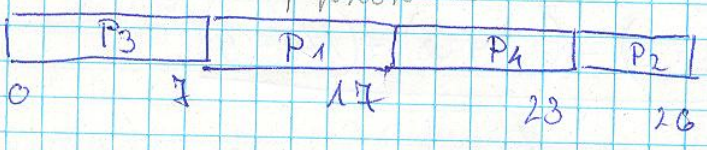
$$\bar{t}_v = \frac{(16-3)+(2-2)+(0-0)+(10-5)}{4}$$

$$\bar{t}_{átf} = \frac{(26-3)+(5-2)+(10-0)+(16-5)}{4}$$

NPP

- nem preemptív prioritásos

most 3a prioritása

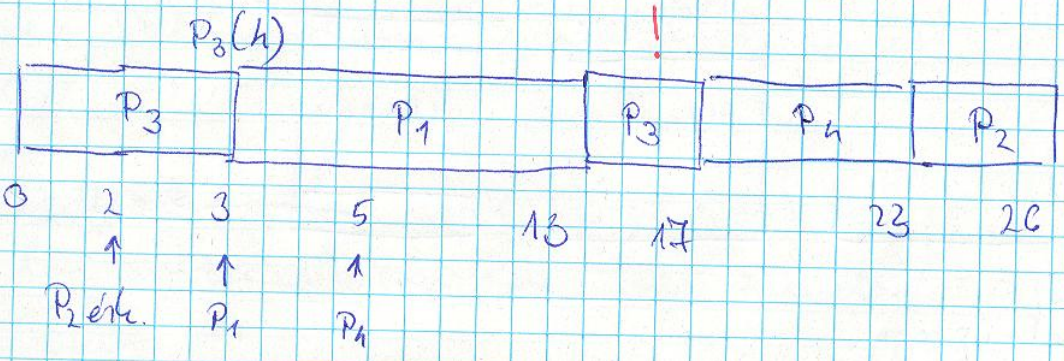


$\bar{t}_v$ = érkezési idő - amikor először szóhoz jut

$\bar{t}_{átf}$ = érkezési idő - amikor befejeződik

| tanulság |

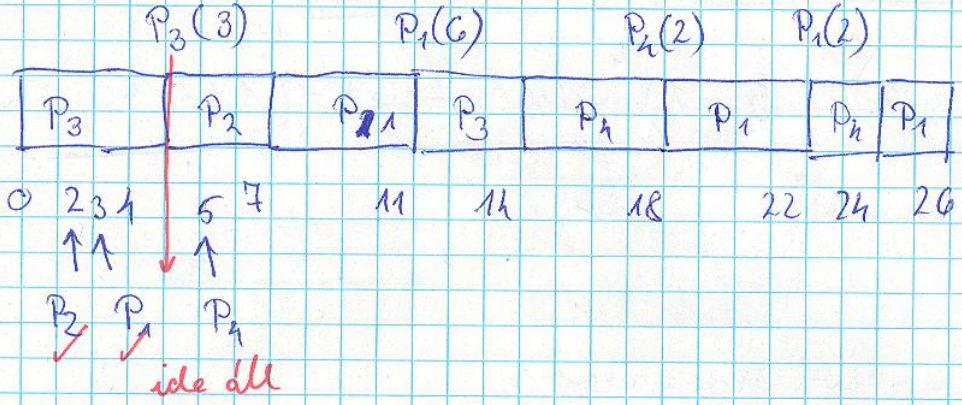
PP preemptív prioritásos



$$\bar{t}_{vár} = \frac{(3-3)+(23-2)+(0-0)+(17-5)}{4}$$

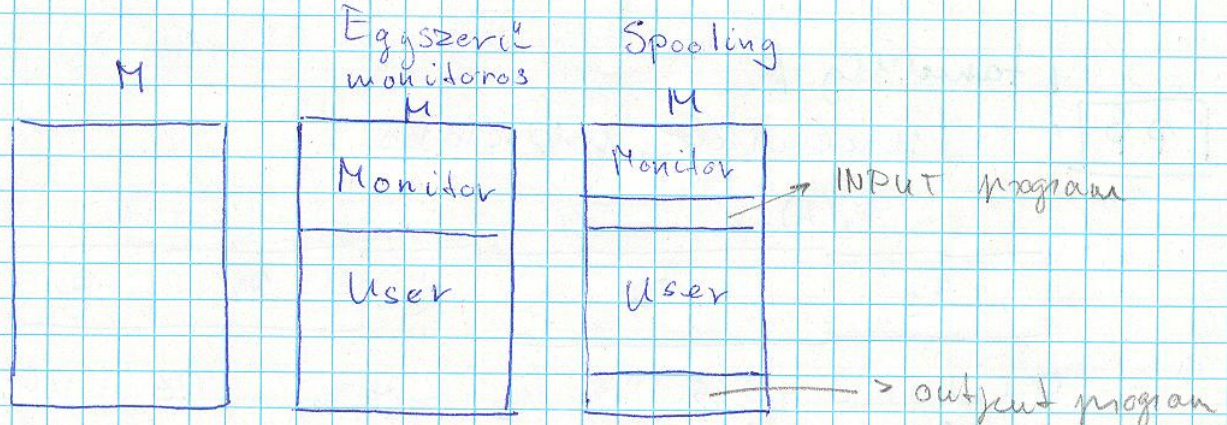
$$\bar{t}_{átf} = \frac{(13-3)+(26-2)+(17-0)+(23-5)}{4}$$

RR (4 időegységgel)



$$\overline{t_{vár}} = \frac{(7-3)+(4-2)+(0-0)+(14-5)}{4}$$

$$\overline{t_{átf}} = \frac{(26-3)+(7-2)+(14-0)+(24-5)}{4}$$



Multiprogr.

- minél többet akarunk kezdeni, annál kisebb az egyre jutó / törzidőzés, nem ugyanakkora időszeletek /
- lehetséges megoldás (fix partíciókra bontjuk)
  ↳ túl merev rendszer

- olyan technika alkalm. hogy a tényleges cím a betöltéskor alakul ki, betehetett tölteni programokat különböző helyre, de probléma marad a fix partíció (mert többet foglalhat egy program mint a fix partíció)

↳ mozgatás a memóriában (címzés)

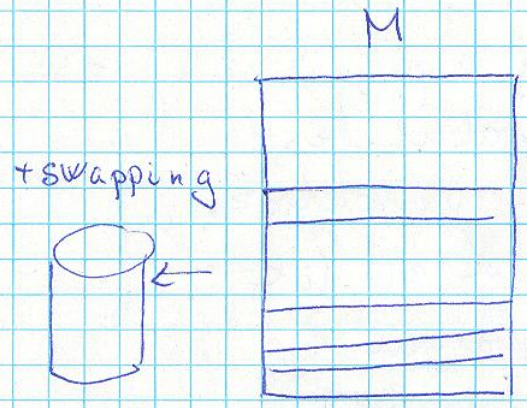
Dedolgozható kód (áthelyezhető)
Dinamikusan áthelyezhető kód

- oda töltöm ahova akarom
- futás közben mozgatható a memóriában

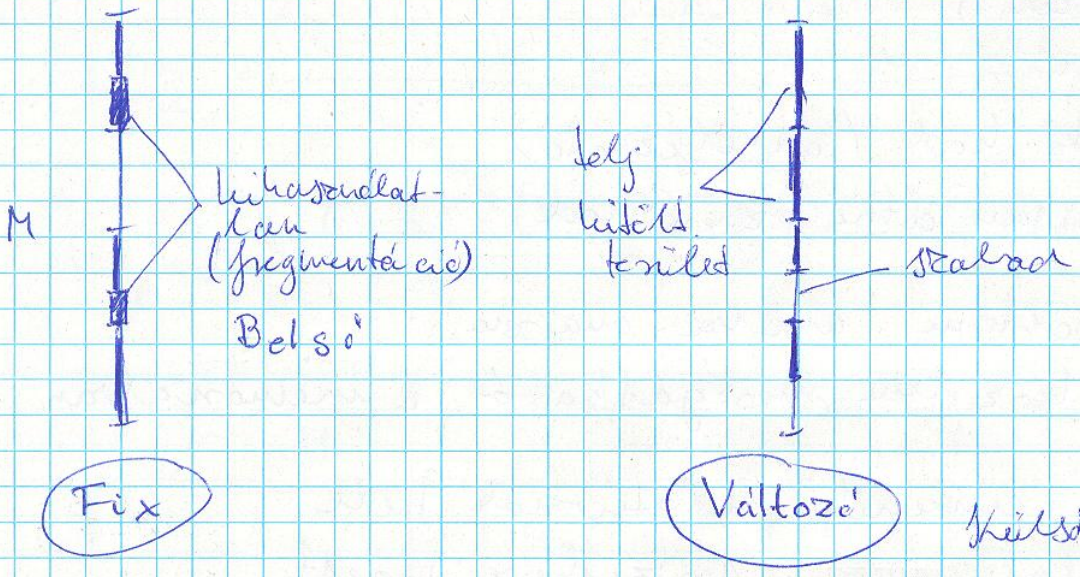
A program címeinek relatívnak kell lennie valamilyen bázishoz képest (PC)

áthelyezés után csak a bázis reg kell utánahangolni ha minden cím relatív

utasításrelatív: az utasításszámlálót mozgatom



Fix vagy Változó egységekkel kezeljük

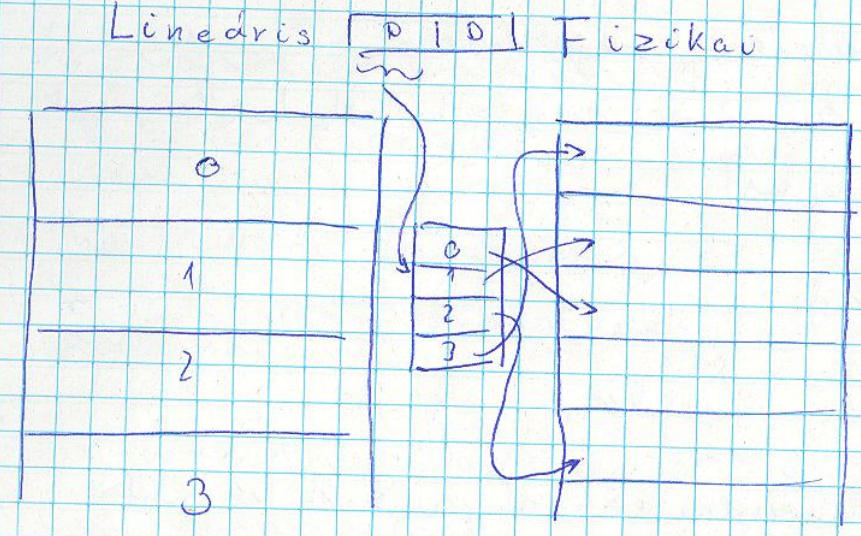


Belső fragmentáció

- az elején minek belső fragmentáció, de a keszcá-lat során kialakulnak

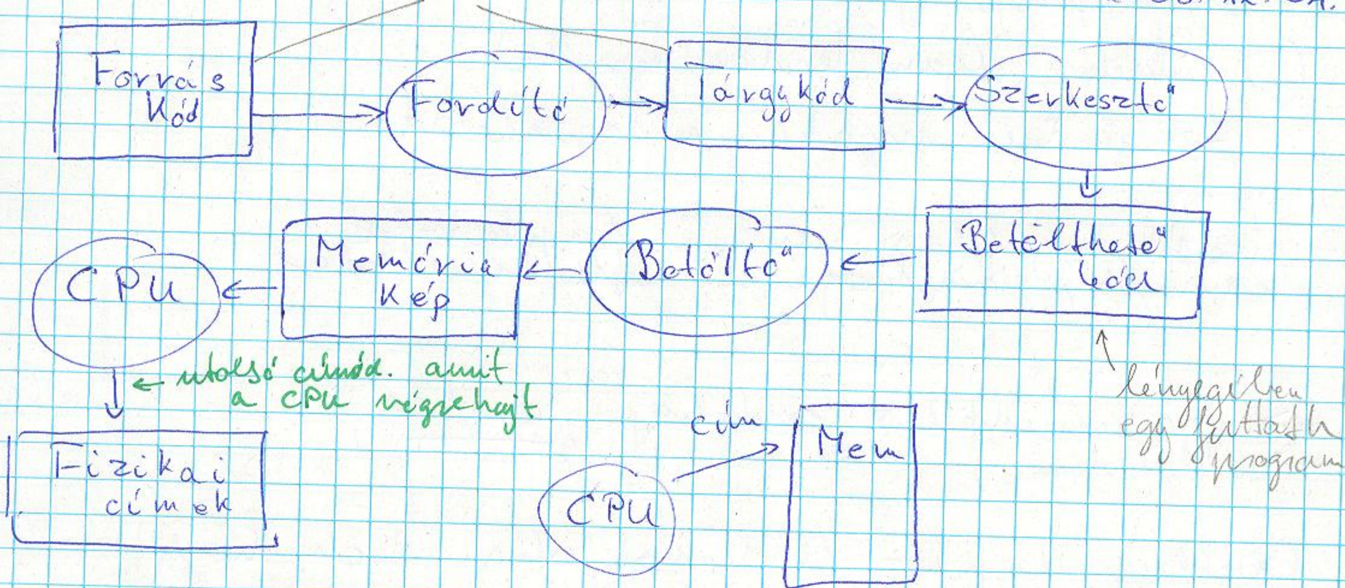
Elhelyezési algoritmus! „Hova tegyem"

Best fit (a legk. szabad hely ahova elfér)
↳ kilóadékminimum!
Worst fit (a legnagy. helyre tegyük be) → a fennmaradó hasznosítás tekintő
First fit (az első helyre tegyük be ahova elfér)



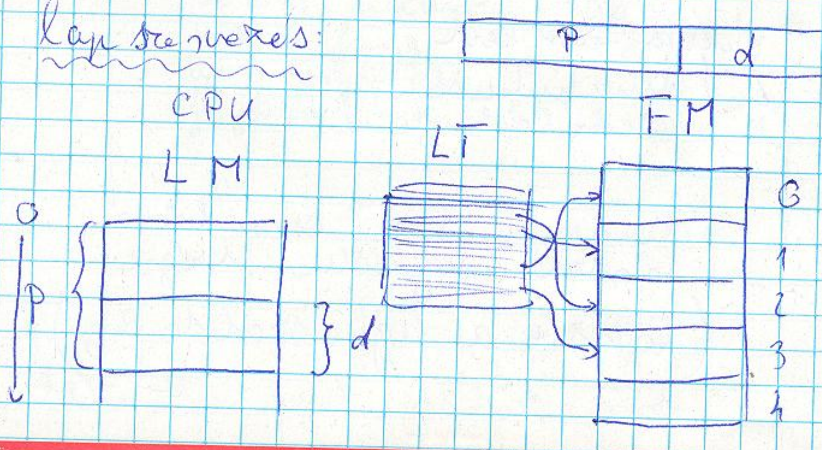
Lapozás, lapcímzés bevezet.
| a logikailag folytonos címtartom. szét lehet szórni |

2008. 12. 04.



- a szerkesztő több tárgykódot kaphat (ha mindegyik fordító ugyanolyan szerkezeti elemeket rak össze a szerkesztő akkor összetudja szerkeszteni)

Lapszervezés:



a laptábla lehetős ad arra hogy a folytonos logikai címtartományt fizikailag nem folytonossá képezzük le.

page fault vagy védelem megsértése
↳ a processzor számára megszakítást kezdeményez
(külső megszakítás)

utasítás roll-back

-> belső állapot finom elmentése

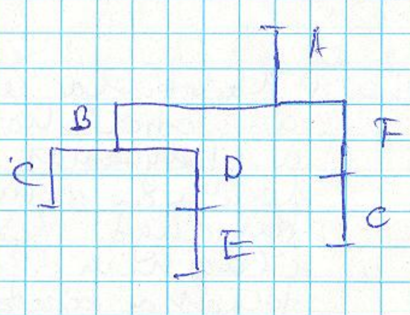
Hardware-nek kell tudni ezt a címzési módot

- ha több folyam fut a tárban, mindegyiknek külön laptáblája van
(minden folyam. külön terület)
↳ el kell menteni a laptáblát

- másik megoldás: melyik laptábla érvényes, kiviszzük... képesti eltolás

=> gyors működés érdekében cache

Overlay: a progr. lehet. van olyan tech. hogy a programozó programjának melyik egységeire van szüksége egyidejűleg
⇓
overlay fa



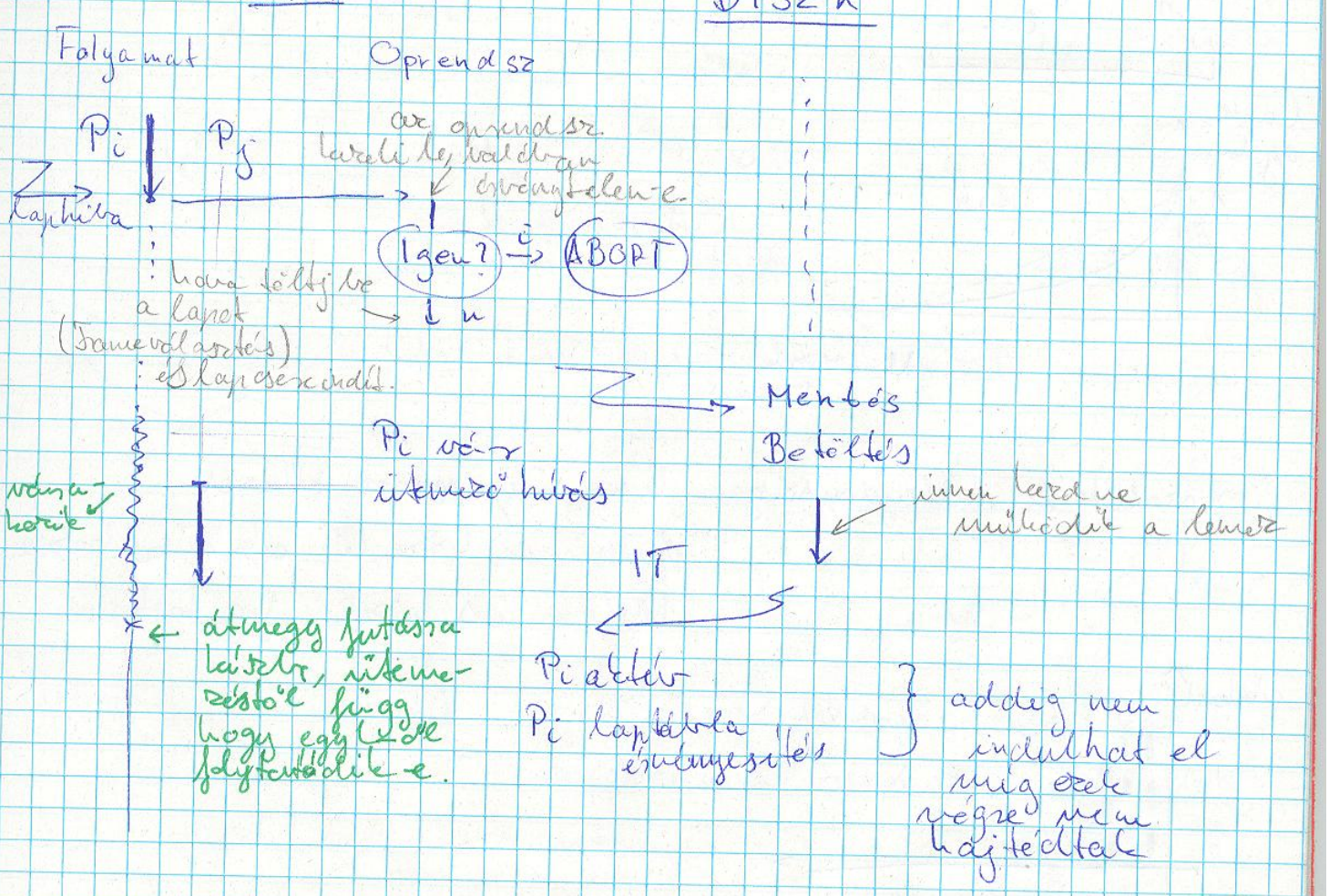
ekkora tár kell
(dinamikusan cserélődnek)

- dinamikusan cserélődő tárban hajtódik végre
(törzs + a leghosszabb ág)
ez a szükség. legn. tárterület

Virtuális tárkezelés

- csak azt hozzuk be amire szükségünk van.
- hardware támogatás
- a felh. azt a tárterületet látja, és ezen tud dolgozni

- software és hardware összjátéka
  (a hibakezelést, ha a lap nincs bent a software-nek kell kezelni)
  → behozza a lapot és utasítás után tud folytatódni a folyamat, ellenkező esetben már nem fog laphibára futni



utasítás végrehajtás ~ 100 ns = 10^{-7} s

a lemez művelet ~ (10^{-3} s) → optimális ha ez alatt a lapcsere idő alatt nemlép be újabb lapkérés

A lemezművelet kiszolgálása megdöntően

$$\frac{1}{\frac{T_{ecs}}{t_m}} > g$$

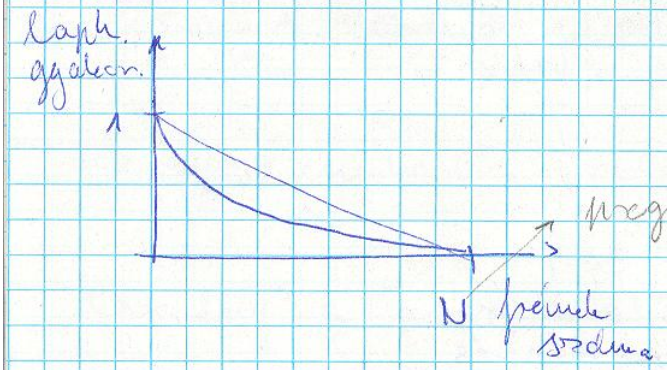
lapkérés gyakorisági mértéke

g : lapkérési gyakoriság

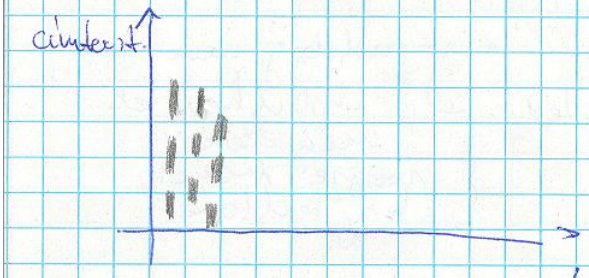
T_{ecs} : lapcsere ideje

t_m : memória hozzáférés ideje

$$g < \frac{t_m}{T_{ecs}}$$



lokalitási tulajdonság: ha egy progr. elkezd futni a közeli címhivatkozás lehet is közel jog történni



Tárhierarchia

külső
lemez
op. tár
cache
CPU

) virtualizáció
gyorsítótár (cache)

↗ mintha nagyobb lenne a tár

találati arány: $\frac{\text{jó}}{\text{összes}} = h$ jó + hibás = összes

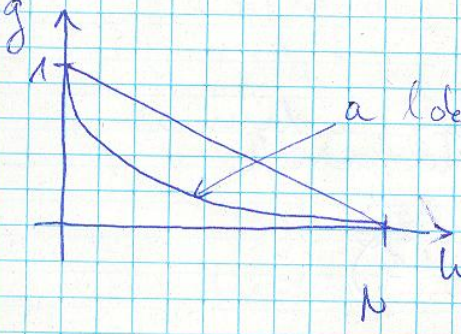
hibaarány: $\frac{\text{rossz}}{\text{összes}} = g$ $g = (1-h)$

$T_{eff} = T_{távoli} \cdot g + T_{közeli} (1-g)$ t_m => közeli
T_t => távoli
↓ lemez ↳ memória

$T_{eff} = T_{távoli}(1-h) + T_{közeli} \cdot h$

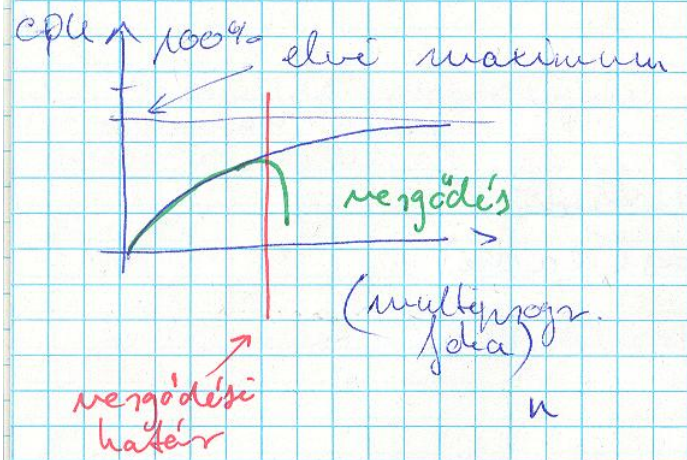
$g_{vh} = \frac{1}{\frac{T_t}{t_m}}$ ← határhatékonyság

2008. 12. 08



$g < \frac{1}{\frac{T_{et}}{t_m}} = \frac{t_m}{T_{kt}}$

vergődés (Thrashing): nem tud hasznos munkát végezni a rendszer egy idő után



minél több folyamat van a rendszerben annál több folyamat tevékenységi ideje használt munkával van a CPU-n

ezt az elvet kell megközelíteni ha optimálisan akarjuk hogy működjön a rendszer.

Betöltési filozófia:
- igény szerinti lapozás
- lesz ahead lapozás

Áldozat választási filozófia (lapcsere algoritmus)
↳ jellemzően befolyásolja a sebességet

- amelyik lapot a legrégebben használtuk (FIFO stratégia)
  - hibalehetőség - a vezérlőlapot érdemes a tárban tartani!
- a legkésőbben hivatkozandó lap legyen az áldozat (OPTIMÁLIS) (jövőbeni)
  - ezt szeretnénk előrelátni, becsülni kell (a múlt szerves folytatásaként kezeljük el)
    ↳ megnézzük a lap múltját

⇒ legrégebben hivatkozott lap legyen (LRU, last recently used)

- LRU
  - minden laphoz időbélyeget rendelünk (ez alapján választunk)
    ↳ ezt hardwares támogatással kell (egyébként lelassul a rendszer)
    → a proc. ezt nem támogatják
  - ez az algoritmus egyik proc. sincs konektül megoldva
- Second Chance (SC)
  - FIFO és LRU ötvözete
  - feltétele egy referenciabit hardware-es kezelése
    (ha hivatkozás történik a frame-re billen ez a bit)
  - FIFO szerint nézem, ha 1-es a referenciabitje akkor nem őt választom, de visszabillentem 0-ra és eggyel tovább lép mindaddig míg talál egy 0-ás referenciabitet
  - körkörös mutató + laponkénti referenciabit
  - „óra" megvalósítás mint hatékony implementáció (ha nem találna egy áldozatot sem mire végigért újrakezdi)

Írtak-e a lapra a dirty bit mondja meg
↳ a dirty bit alapján is választhatok áldozatot

választási stratégia ↓

0 0 Kombinációja:
0 1 → nem hivatk. de felülírt
1 0
1 1

RD

- Szabad pool
  - mikor áldozatot vél. összeállítunk egy áldozatlistát, ebből a listából kell választani
  - ezt az algoritmust akkor futtatják mikor a proc. szabad
  - a jelölteket egy láncolt listába fűzzük
    - ↳ ha a referenciabit idők. változott akkor nem választják

Egyenlőre kiszedik el a végére listát

- Előrehozott kiírás
  - a lemez teljhelyés csúcsait igyekszik csökkenteni
  - ha a lemez szabad kiírom a dirty lapokat

Szimulált referenciabit

- hardwares érvényességi bit | ha érvénytelen lapra hivatkozunk megszakítás
- software érvényesség
- software referencia

HÉ	SÉ	SR	
1	X	1	Betöltés
0	1	0	SC algoritmus
1	X	1	Laphiba kezelő

ez jelenti az érvényeset

a laprahivatkozáshoz (első) keletkezik egy szimulált laphiba

- hardwares támog. nélkül megtudják valósítani a referenciabitet

Példa:

4 frame

csak az új hivatk. írja le

hivatkozások:	7	0	1	2	0	3	0	4	2	3	0	3	2	5	6	1	2	0	1	7	2
frame																					
0	7	7	7	7		3			3			3			3	6	6	6	6		1
1		0	0	0		0		4			4			4	4		1	1	1		1
2			1	1		1		1			0			0	0	2	2				2
3				2		2		2			2			5	5	5	5	6			0
						x					x			x	x	x	x	x			x

FIFO

↑ laphiba

0	7			3		3							5	6	1						7
1	0			0		0							0	0	0						0
2	1			1		4							4	4	4						4
3	2			2		2							2	2	2						2
				x		x							x	x	x						x

GPTIM

0	7			3		3							3	3	1			1			1
1	0			0		0							0	0	0			0			7
2	1			1		4							5	5	5			0			0
3	2			2		4							2	2	2			2			2
				x		x							x	x	x			x			x

LRU

/előrement/ mert ott választ.

0	7	+		3	+	3	+						3	-	6	+	6	6	-		7
1	0	+		0	-	0	+						0	-	0	+	1	1	-	+	1
2	1	+		1	-	4	+						4	-	4	+	2	2	-		2
3	2	+		2	-	2	+						5	-	5	+	5	5	-		0
				x		x							x		x		x	x			x

SC

Tárgazdálkodás

- globális
- lokális (előre eldöntjük ki mennyi frame-et kap)

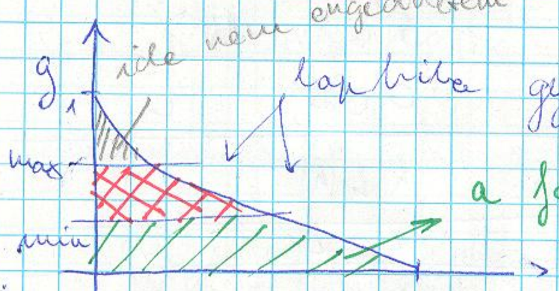
Working set (működő lapkészlet)

- ennyi db. frame-re neki szüksége van hogy műk. maradjon

laphalmaz

mért: frameszám

2008.12.09

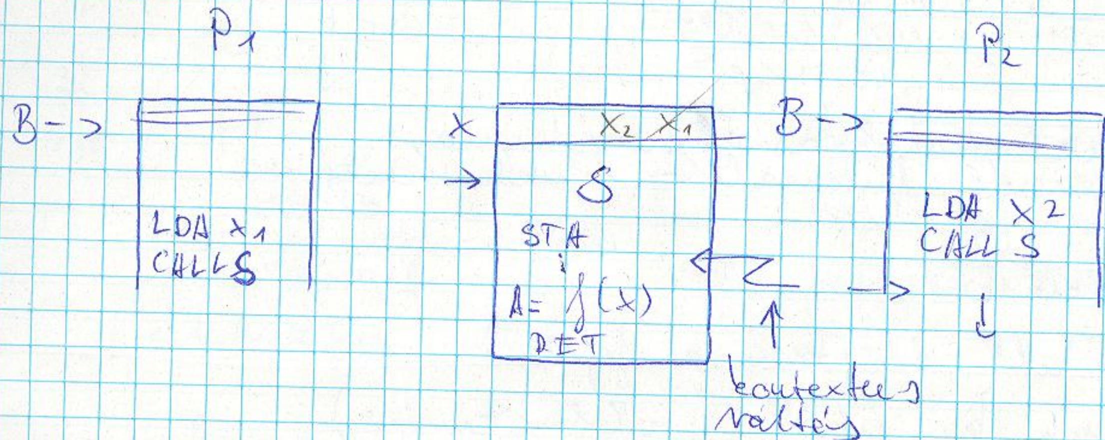


gazdálkodás hogy kinek mennyi framet adjunk

Bélády-anomália: ha egy program eggyel több framet kap akkor több laphibát generál, ez a Bélády-anomália
↳ csak a FIFO mutatja, nem mindig
„Több framen több laphiba!"

for i=1 to N do => tömb elemei sorfolytonosan
  for j=1 to M do L> egy i-hez tartozó
    T[i,j] j-k egy lapon vannak

|van oszlopfolytonos is|



· felülíródott az x1, rossz értéket ad vissza a P1-nek, ez így nem egy újrahívh. kód

az az újrahívható kód ami a folyam. számára bármilyen állapotban megh. és helyes eredm. ad vissza

-> nem lehet saját tehetsze, ez okozza a hibát
-> minden foly. külön tárterül. dolgozzon, tárterületek

-> bázisregiszteres megoldás
-> akkor a példány. fordul ami a hívó területén helyezkedik el

minden egy példányban tárolódhat

Lemez - Fájlkezelés

Fájlkezelés >> előző funkcionalitás

lemezkezelés ← (blokkosítás megvalósítása (hogy úgy látszódjon mint egy "blokk ami 0-n-ig címezhető")

szektor az írás/olvasás legkisebb egysége, ennél finomabb lépték nincs

Fejlet kijelölése:
- hova kell állítani a fejet
- melyik fej mit (többszintes)
- a forgás mentén melyiket akarom olvasni

⇓

az operációs rendszer ezt nagyon „elvarázsolt"

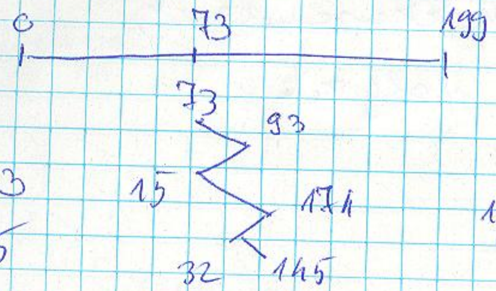
minél kisebb blokkokban tesszem át az adatot annál nagyobb a „hulladékidő" (fej pozicionálása, lemezforgás kivárása)

-> sok adatnál az egy adategység: első „hulladékidő" kisebb

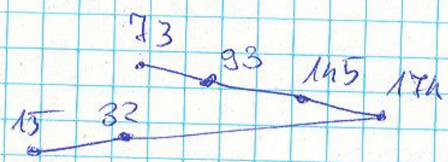
=> optimális blokkméret teljes megvalósítása! (a fél nagy blokkok sem jók)

Lemeztömörítés:

azonos fejpozícióban elérhető adatok - cilinder



→ fejpozíciók



93
15
174
32
145

↓

FIFO sorrend esetén a fej össze-vissza rohangál

(ütemezni nem kell)

↓

legközelebbi kiszolgált (ha a fej elkeveredik kiéheztetés történhet)

Lift algoritmus

seek: le-föl pásztáz, elmegy a végéig majd vissza

look

→ ha nagy a lemez akkor nagy előzések történhetnek

cseek (cirkulációs)

clook

körbemegy, ha végigért az elejére ugrik

↓

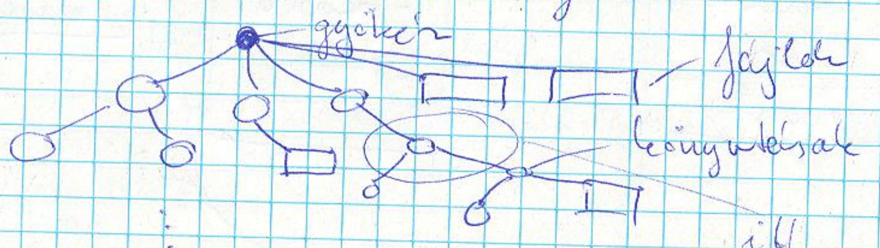
annyiban különb a seek-től hogyha az adott haladási irányba nincs több kérés visszafordul.

Fájlrendszer:

|LO|CICA|KUTYA

↑ hivatkozási mód

- hierarchikus katalógusrendszer

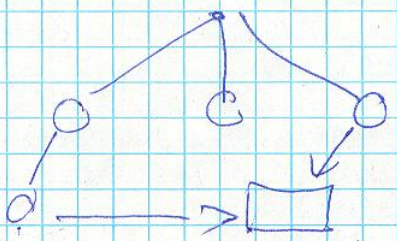


(csak az ill. dolg. a felh. ehhez tart. ágait látja)

aktuális könyvtár: megjegyzett pont, amit a fűk. kezdeti, épp ott dolgozik

- abszolút név
- relatív név (a progr. fájljain belül is lehet egy Lo(cica/ku-tya elrendezés)

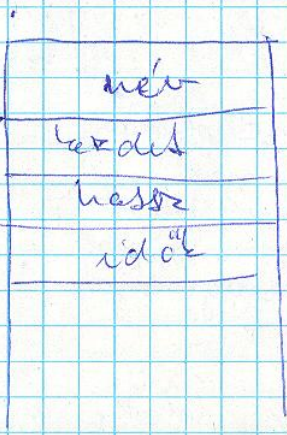
közös fájlokra minden felhasználó szeretne a saját fa szerkezetéből hivatkozni
/ hivatkozási kuszlók alakulhatnak ki!/



név. kiterjesztés

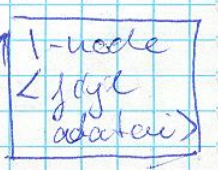
.com
.exe
.bat
} végrehajtható fájlok

hatalógus:



tovább fa szerkezetnél (egy bejegyzés egy helyen)

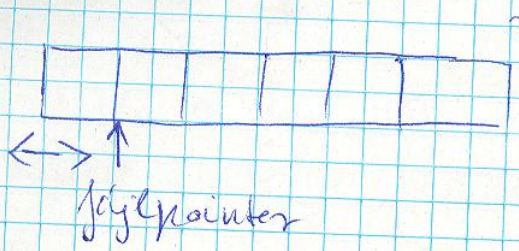
(UNIX)



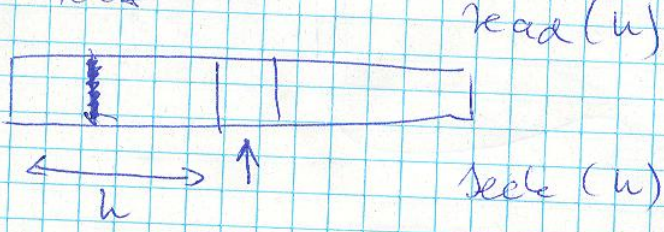
másoknak is hivatk. rá!

Fájlmodellek

szekvenciális



direkt



Allokáció: (terület foglalás)

folytonos allokáció: szomsz. blokkokba folyto-
nosan helyezzük el a
file adatait a fájlnak

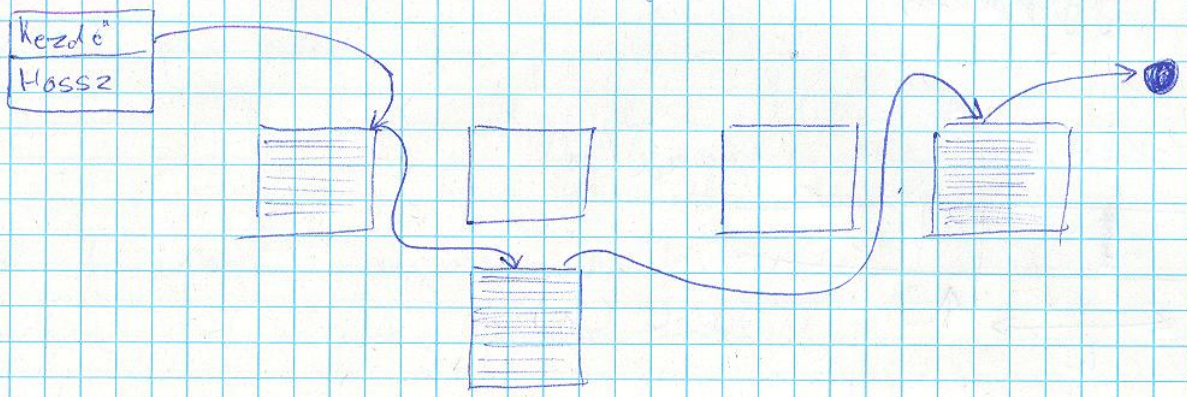
láncolt allokáció: a fájllhoz tartozó
adatok ki-be-visszahelyezése a lemezen

(a direkt elérést nem támogatja,
másolásnál hosszadalmas)

↳ file allokációs tábla
↳ indexelt allokáció

2008. 12. 11

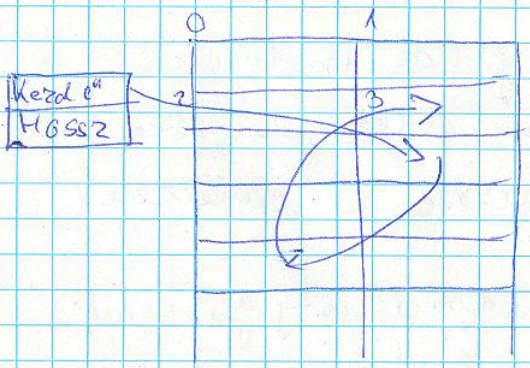
Folytonos láncolt



- láncoltban mentadunk olyan gyorsan pozícionálni

FAT: file allokációs tábla

- minden egyes blokknak van egy mezője ebben a táblázatban
- ez egy blokkcímet tartalmaz



Merevlemez: 100 GB = $100 \cdot 2^9$

Blokkméret: 10 kByte = $10 \cdot 2^3$

$10 \cdot 2^6$

24 bites címek kiír. el

$\sim$ 40 MB a FAT mérete

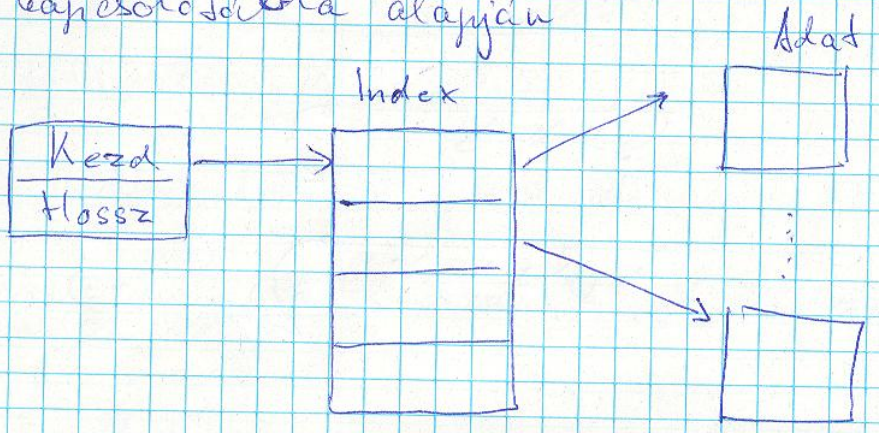
- a biztonság kedvéért 2 példányban tárolják
- láncolt allokációnál sokszor lemezműveletet az adott blokk eléréséhez.

/ FAT segít ! /

- ha a lánc „elszakad" szétesik a lánc.

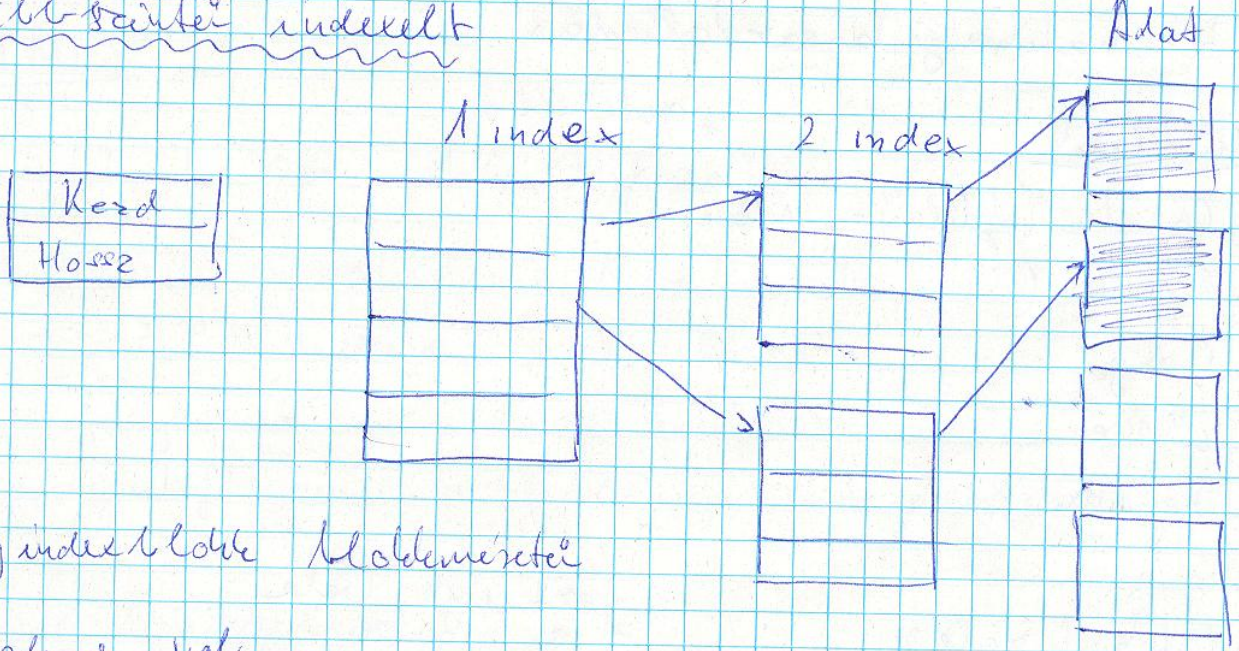
Indexelt allokáció

- a fájl adatai össze-vissza helyezkednek el
- kapcsolódásukat a tábla alapján



- minden adat eléréséhez 2 lemezművelet

Több szintű indexelt



egy indexblokk blokkmérete

Szabad hely

- minden lemez blokkhoz tartozik egy bit (bittérkép) → szabad-e az adott blokk

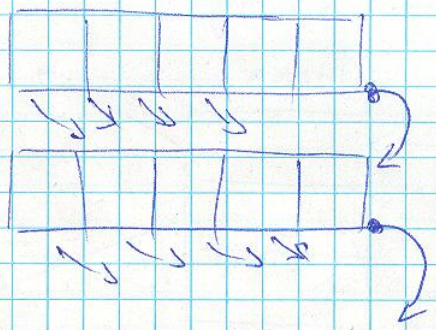
=> gyorsan keresh. szabad helyek

ezekre a lemez blokkokra nincs szükség

A lemezblokkok használat közben sérülhetnek

- ha 2 bites akkor hibajelzésre is alkalmas

Láncolt lista



A lemezek igyekeznek az adatokat közel tartani egymáshoz (lokalizációs tulajdonság)
↳ hatékony tárgazdálkodás

Műveletek:

Create (fájlnév , (méret)) ← opcionális
Delete (
Copy
Move
Rename
:

Adatok kezelése

read (file_id , adat) ← ezt írja bele a fájlba
write (file_id , adat) — név alapján történő keresés lassú
seek (file_id , pozíció)
open (file_név , file_azonosító) → ezt kapom vissza, hivatkozok a read write-ban ezzel!
close

(belső puffer)

a file cache-elt lesz! (belső puffer miatt)

Filevédelem

- egy filerendszerben több felh. van
- illetéktelen ne férh. hozzá
- többpéldányos tárolás

=> átalakultak a filerendszerek és a lemezkezelés

- védelmi mechanizmusok filerendszerenként különbözőek

„Kötetlen munka a saját adatainkon"

L> kompromisszumok

UNIX

s5fs (system file system)

- mindenki az ősfolyamat leszármazottja
  L> fa szerkezettel ábrázolható
- a folyamat saját magát „klónozza"
- a gyerekfolyamat a szülő file-jait örökli

unix lemezszerkezete

Boot	Superblokk	I-node	adatterület

Superblokk ↓

ami alapján eltelhet igazodni a lemezen

-> rendszerméret

-> szabad helyi blokkinformációk

(példány közben) a tárban műk. -> I-node cache (szabad i-node-ok)

Katalógus-szerkezet:

- ugyanarra a file-ra több hivatkozást legyen bejegyzés -> ezt megengedi

Név
i-node szám

i-node szám -> rámutat egy olyan leírásra ami a file összes lényeges adatát tartalmazza

(a többi könnyen kezelhető)

- új file létrehozásához kell egy szabad i-node is.

- i-node méretével meghat. a tárolható file-ok számát

i-node tartalma:

- minden lényeges infó az adatról
- -> tulajdonos
- -> csoport (a felh. csoportokban lehetnek)
- -> típus (szabad, normál, link, char/block, pipe, socket)

normál ↓ normál lemezen tárolt adat file

link ↓ egy már létező file-ra egy hivatkozás (a file-ban egy abszolút útvonal van)

char/block ↓ I/O karakteresen kell e lekezelni v. blokk

unixban az I/O kezelés is filekezelés

-> engedélyek, jogok (tulaj, csoport, többiek)

read, write, execute

↓

végrehajt.

r w x | r w x | r _ _

↑ tulaj mit tehet — ↑ csoport — ↑ többiek

-> idő (utolsó hozzáférés, módosítás, létrehozás)

ez a 3. időfogalom van UNIX-ban

-> linkek száma

(ha töröljük a közösen használt file-t akkor kérdés hogy törlődjön)

-> hány helyen van rámutató bejegyzés

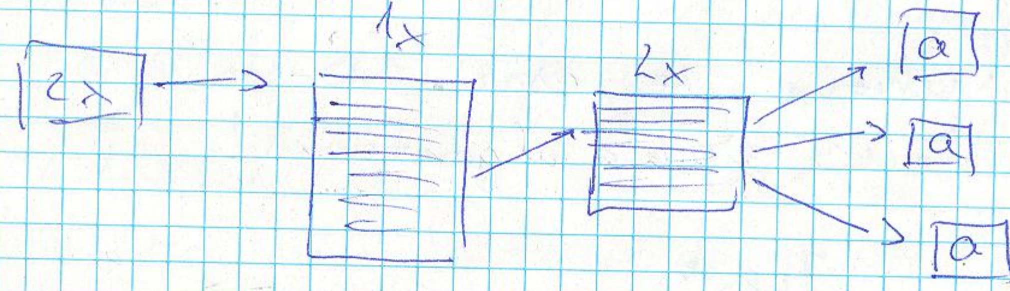
(ha nem 0 csak a hivatkozási bejegyzést töröljük)

-> adatok elhelyezkedése

10 db közvetlen hivatkozás az első 10 blokkjára

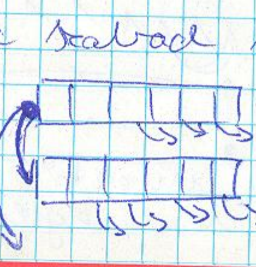
3 további pointer

1x indexelt
2x indexelt } elérési lánc pointerei
3x indexelt



a kis file-oknál természetesen lehetne kisebb blokk vagyunk.

a szabad blokkok össze vannak láncolva — a folyt. allok. nem lehet megoldani.



-innen osztogatja a szabad blokkokat amíg vannak, az utolsó blokk a következő szabad blokklistát tartalmazza

Konzultáció

16 bites szelektor
- 1 bit táblaindex (nem csökk!)
- a két legalsó bit privilégiumszintet jelent

MULTIBUS II

- ha 2 Ø-ás spec. igény, akkor is ugyanúgy lejátsz az algoritm.
(minden end-of cikl. végén beléphetnek)
- mindig 3 órajel alatt dől el!
- BRQ* akkor engedik el ha elkezdődött a kiszolgálása
- wist. megsz., üzenet protokoll
(256 kezdő és küső)

broadall 1-el csökk. mert mindenkit szól!

broadcast - általános üzés

- 32 lehetne, de csak 20-ra működik ez az algoritmus!

példa: ARB5* →

A	0	0	0	0	1	1
B	0	0	1	1	1	1
C	1	0	1	0	0	0

000011
001111

1 11111

innentől nem egyezik.

VME

- arbitrációs rendszer
  - kombinált mechanizmus
    (h szintes daisy
    h külön kérő
    h külön vállalással)
  - az egyes szintek fürtökből
    (de az prioritást jelent.

1. DB arbiter!
⇒ master lehet!
(teljesen láncolt asszinkron protokoll)

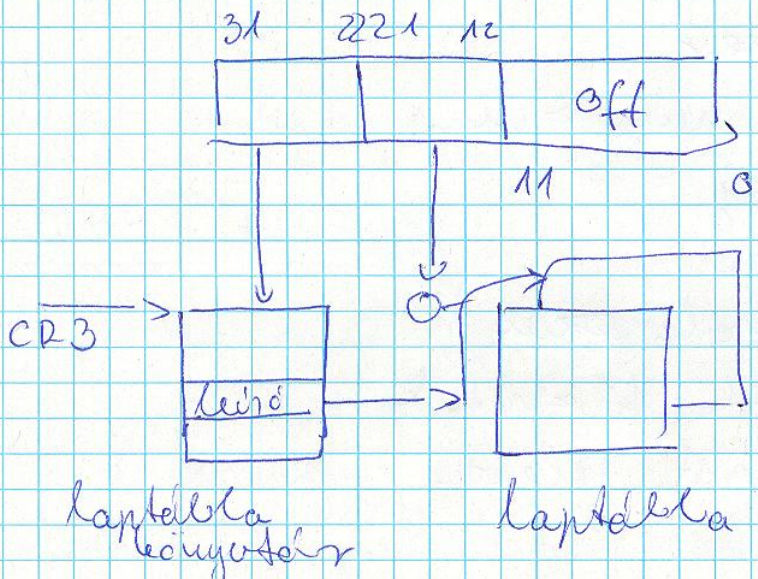
- megszak. rendszer

7 db megszk. kérő vezeték.
↳ 7 külön megszak. kezelő!

1. db. IACK* jelem van
8 bites vekt. hiányz fel az egység
(azon. tárolom) → 256 kérő egység

1 fürtő fel minden 7-hez.
3 bit vál. ki melyik szint.

2 lépcsős laptábla:



20 bites leíró = 20 °C.
+
12 bites offset

1 leíró 4 byte

12 4 leíró-

laptábla 8 byte

8 byte kell minimálisan

CR3-ban 20 értékes bit (az utolsó 12-nem kell, azt más bitből lehet)

minden taskhoz csak 1 leíró (TSS) tartozhat!
ezért kell a task gate

(csak 16 bites szelektor
ami a TSS-re mutat,
nem kell +32 bit!!!)

I/O PL

- input, output utas. is privilégium
- az adat. progr. ha magas priv. szinten fut megszakíthatja az I/O utasítást

16 bites logikai cím
512 kbyte memória

4 index
+2 verembejegyzés
!
16 regiszter

16 x 9 bites
szósz. bitsz.

1E80 H

Hexa 8000
↳ logikai cím

1000

↳ 8 as számú reg.

1000 h kbyte

0A000 hk
0AFFF

1A000 hk
1AFFF

Párh. leírása nyelvi eszk.

- fork - join
- explicit deklaráció

UNIX kommun. megold.
+ fájlrendszer (S5fS)

gyerek viselkedés: „magára tölt egy új file-t"

SOCKET: univ. csatlak. pont amin üzeneteket lehet küldeni és fogadni
- alapvetően hálózati együttműk. használják.
FIFO szerkezetek
üzenetküldés együttműk.

katalógus - i-node
- file neve

példa:

t_m (fizikai memória elérése) [ns]

T_{lcs} (lapcsereidő) [ms]

A megoldáshalmaz igaz hogy az eff.
a fixpontjai adják ellentétesek a 2x-e.

„3x-os indexelés tulajdonképpen a világ megcáfolására elegendő"