Operációs rendszerek I.

Holtpont és kezelése

Várkonyiné Kóczy Annamária
Professzor
Informatika Tanszék
(koczya@ujs.sk)
varkonyi-koczy@uni-obuda.hu

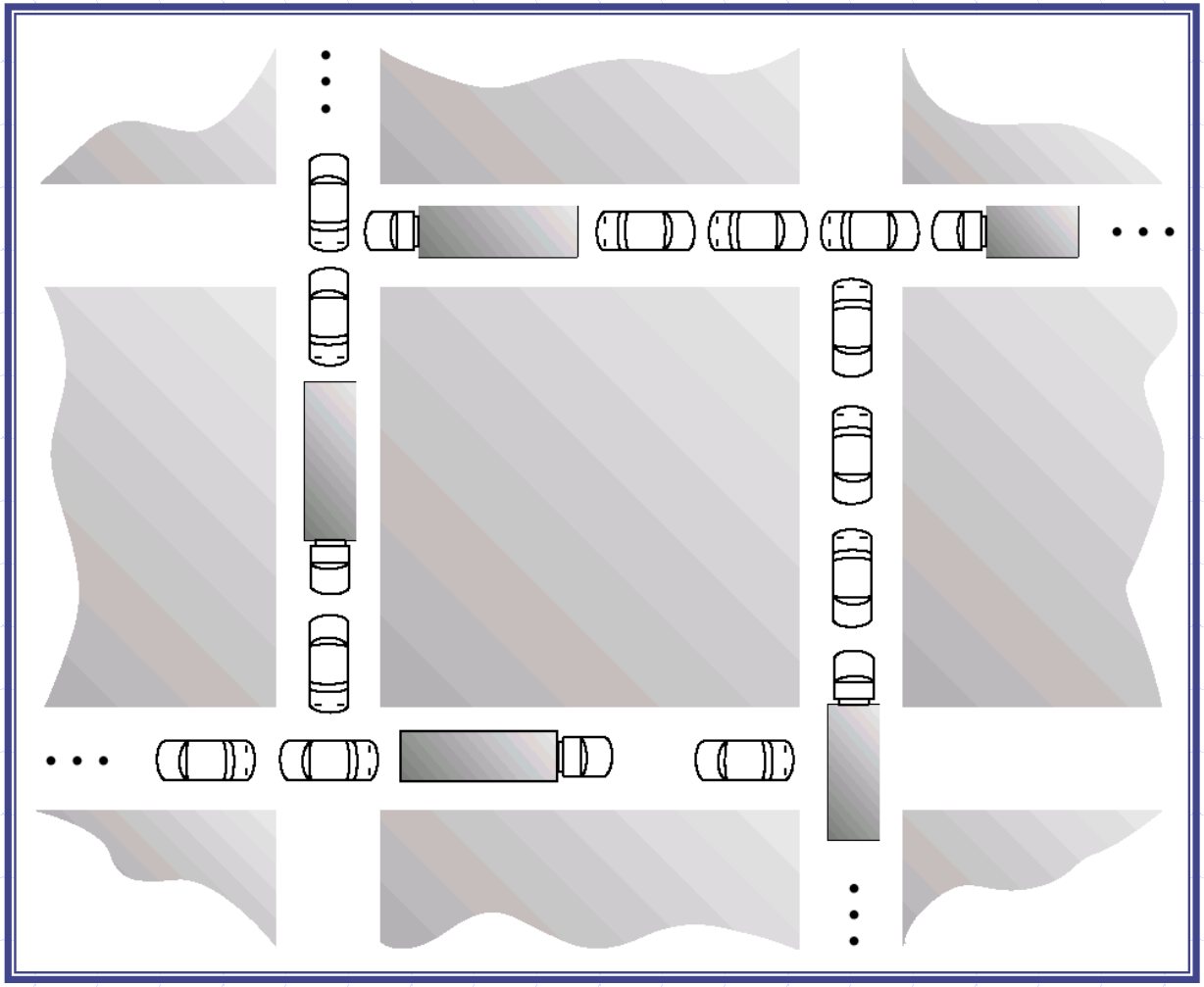
Felhasznált irodalom:

- **Kóczy-Kondorosi (szerk.): Operációs rendszerek mérnöki megközelítésben**
- **Tanenbaum: Modern Operating Systems 2nd. Ed.**
- **Silberschatz, Galvin, Gagne: Operating System Concepts**

Tartalom

- Bevezetés
  - A holtpont kialakulásának szükséges feltételei
  - Az erőforrás-használati gráf
- A holtpont kezelése
  - holtpont megelőzése
  - holtpont elkerülése
  - holtpont felismerése és megszüntetése
- Kombinált stratégiák
- Kommunikációs holtpontok

Bevezetés helyett...



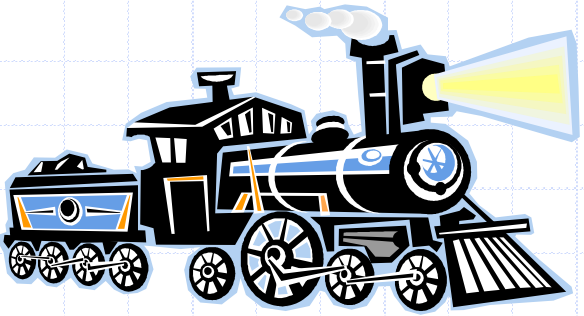
4.1 Holtpont definíciója

Folyamatok egy halmaza akkor van holtponton, ha a halmaz minden folyamata olyan eseményre* vár, amelyet csak egy másik, ugyancsak halmazbeli (várakozó) folyamat tudna előidézni.

Az esemény többnyire egy erőforrás felszabadulását jelenti.

A klasszikus példa…

„When two trains approach each other at a crossing, they shall both come to a full stop and neither shall start up until the other has gone.”



Kansas állam törvénye a XX. sz. elejéről

Rendszermodell 1.

- A rendszerben véges számú erőforrás áll rendelkezésre véges számú folyamat számára
- Az erőforrások osztályokba sorolhatók, az egy osztályon belüli erőforrások azonosak (egy igénylő folyamat bármelyiket igénybe veheti)
- Az erőforrás lehet
  - elvehető (preemptív) [pl. CPU, memória]
  - nem elvehető (nem preemptív) [pl. nyomtató]

Rendszermodell 2.

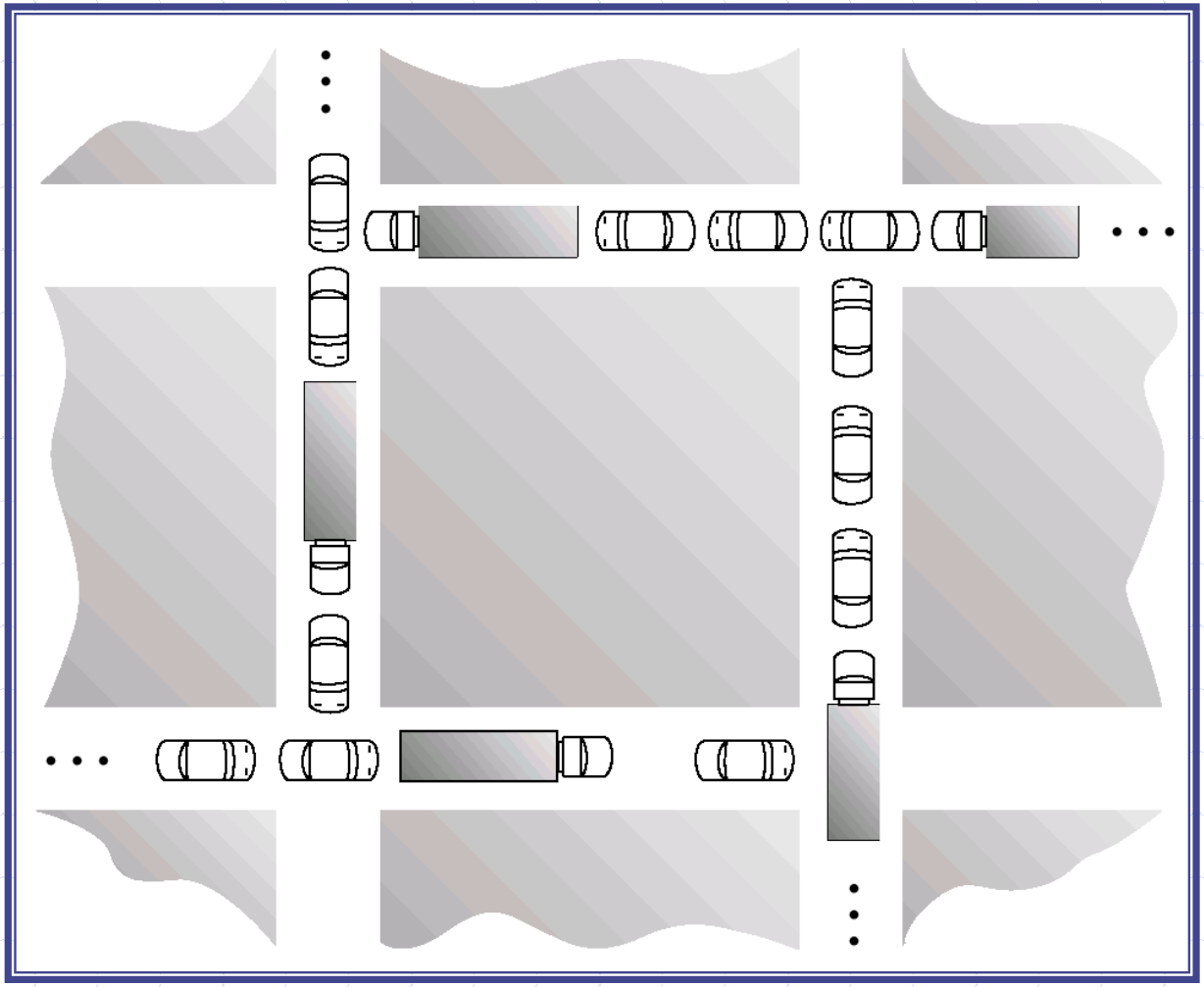
- Erőforrás használata
  1. Igénylés. Ha az igény nem teljesíthető (erőforrás foglalt) akkor a folyamat várakozik.
  2. Felhasználás. A folyamat az erőforrást kizárólagosan használja.
  3. Felszabadítás. Folyamat elengedi az erőforrást. Ha más folyamatok várakoznak rá, akkor valamelyik várakozó folyamat továbbléphet.
- Az 1. és 3. lépések általában rendszerhívások
  - request – release (device)
  - open – close (file)
  - allocate – free (memory)

4.2 Holtpont kialakulásának szükséges feltételei

Holtpont akkor jöhet létre, ha az alábbi négy feltétel egyszerre teljesül

- Kölcsönös kizárás
  - Egy erőforrást egyszerre legfeljebb egy folyamat használhat.
- Foglalva várakozás
  - Van olyan folyamat, amely lefoglalva tart erőforrásokat, miközben más erőforrásokat kér (és azokra várakozik).
- Nem elvehető erőforrások vannak a rendszerben.
  - Az erőforrást a folyamat csak önszántából szabadíthatja fel.
- Körkörös várakozás
  - Létezik egy olyan $\{P_0, P_1, ..., P_n\}$ sorozat, amelyben P_0 egy P_1 által lefoglalva tartott erőforrásra vár, P_i egy P_{i+1} által foglaltra, végül P_n pedig egy P_0 által foglalt erőforrásra vár.

Teljesülnek a feltételek?

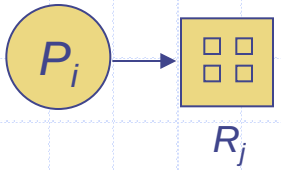


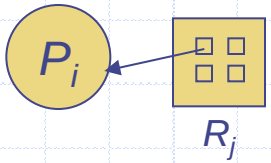
4.3 Az erőforrás-használati gráf

- A rendszer pillanatnyi állapotát a rendszer erőforrás-használati gráfja (resource allocation graph) írja le.
  - $P = \{P_1, P_2, \ldots, P_n\}$, a rendszer folyamatainak halmaza
  - $R = \{R_1, R_2, \ldots, R_n\}$, az erőforrás-osztályok halmaza
  - $P_i \rightarrow R_i$ erőforrás *igénylés*
  - $R_i \rightarrow P_i$ erőforrás *használat*
- Ha az igény kielégíthető, akkor az erőforrás igénylés átvált erőforrás használattá.

Az erőforrás-használati gráf

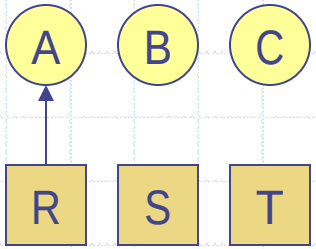
- Folyamat
- Erőforrás osztály 4 erőforrással
- P_i egy R_i osztálybeli erőforrást igényel
- P_i egy R_i osztálybeli erőforrást foglal



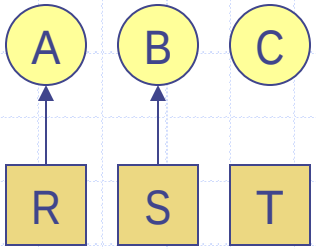


Példa holtpont kialakulására

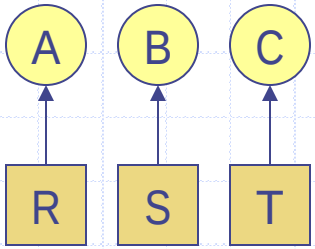
A:
Requests R
Requests S
Release R
Release S
B:
Requests S
Requests T
Release S
Release T
C:
Requests T
Requests R
Release T
Release R



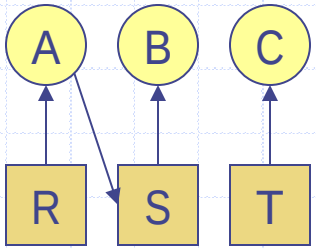
A requests R



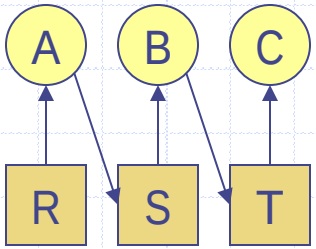
B requests S



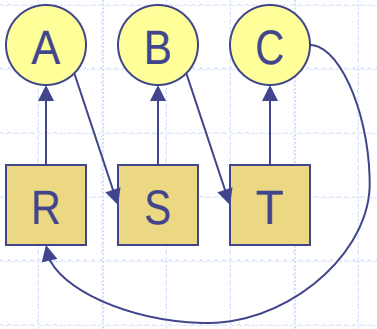
C requests T



A requests S



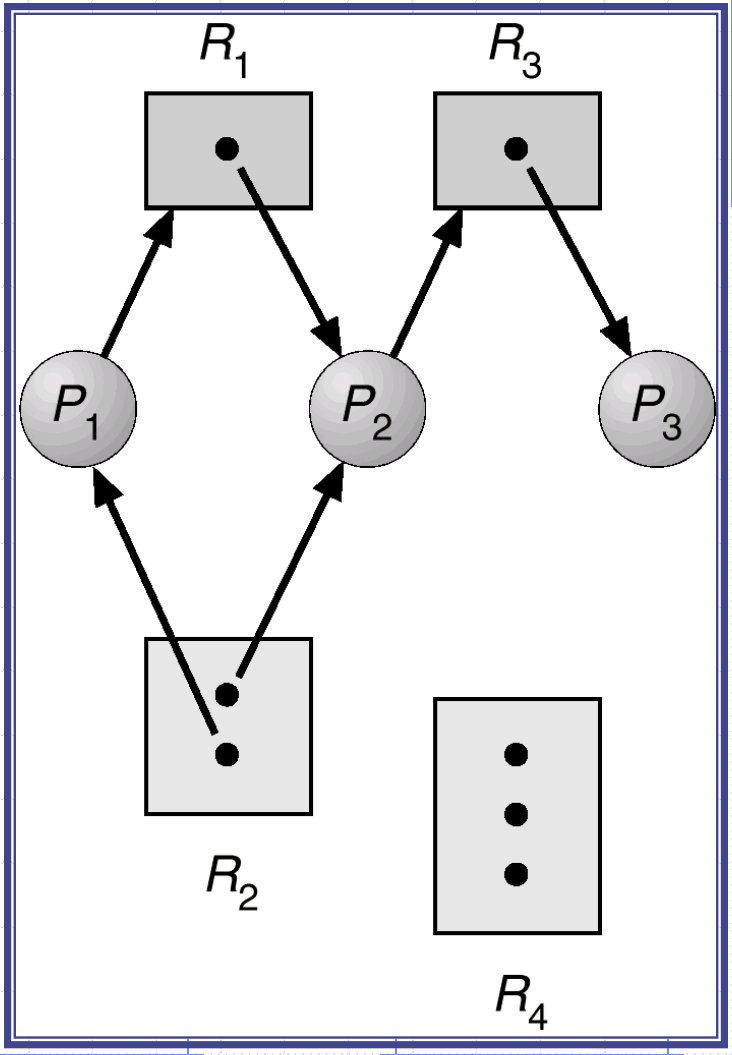
B requests T



C requests R

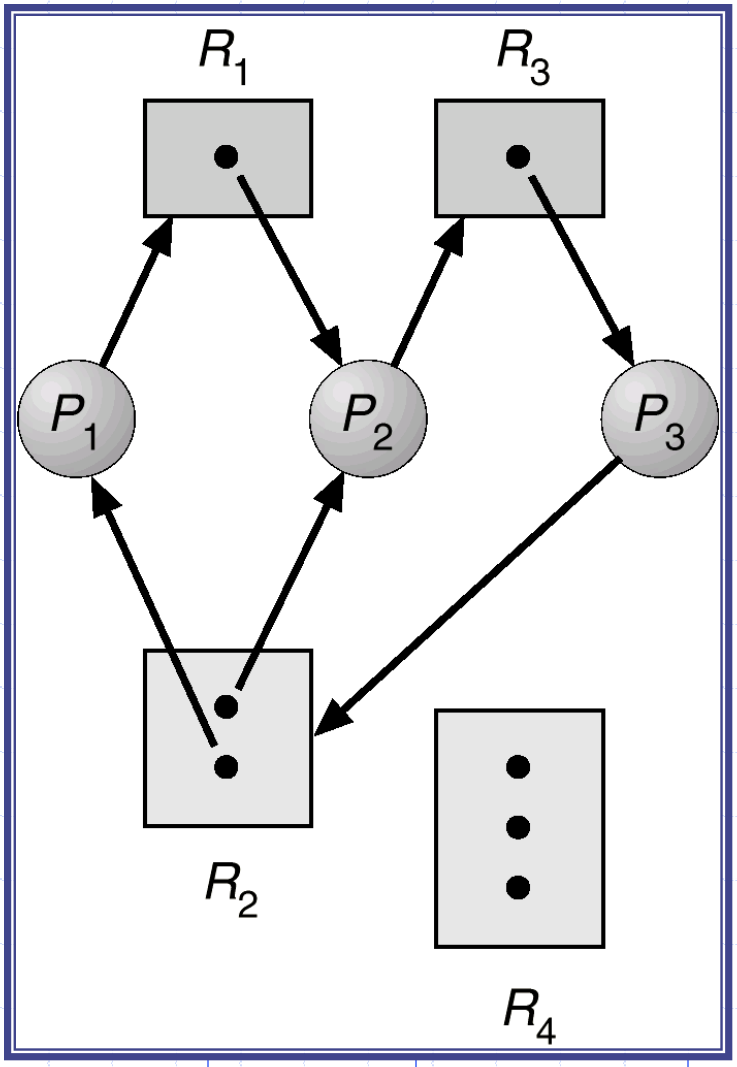
Egy erőforrás-használati gráf

- R_1-et P_2 használja
- P_1 vár R_1-re
- R_2 egy-egy példányát használja P_1 és P_2
- R_4-et nem használják
- ...



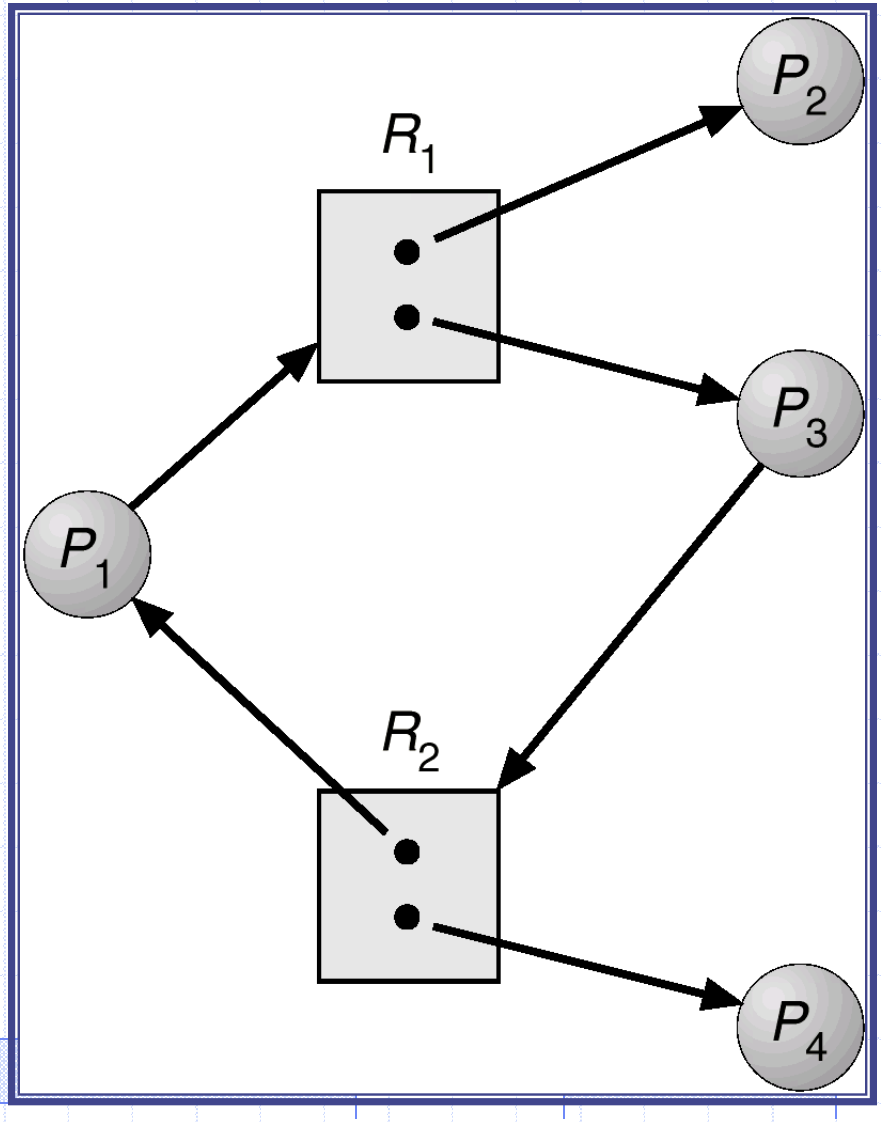
Egy erőforrás-használati gráf holtponttal

- Mindhárom folyamat vár
- Az igényelt erőforrások felszabadítását csak ezen folyamatok tudnák elvégezni

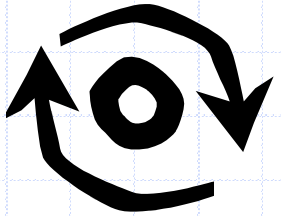


Egy erőforrás-használati gráf körrel, de holtpont nélkül

- Az erőforrásokat P_2 és P_4 is felszabadíthatja!
- *A ciklus szükséges és elégséges feltétel, ha minden erőforrás-osztályba csak egy erőforrás tartozik.*
- *Szükséges, de nem elégséges, ha egy erőforrás-osztály több egyedet is tartalmaz.*



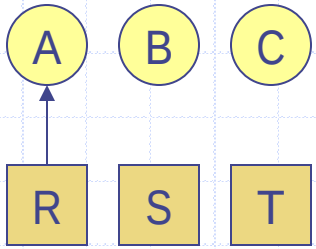
A kör és a holtpont...



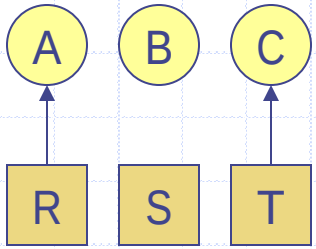
- Ha nincs kör, nincs holtpont. (*A kör szükséges feltétel.*)
- Ha van kör és minden erőforrás-osztályba csak egy erőforrás tartozik, akkor holtpont van. (*Egyszeres erőforrásoknál a kör elégséges feltétel is.*)
- Ha van kör és egy erőforrás-osztály több egyedet is tartalmaz, akkor holtpont kialakulhat, de nem szükségszerűen. (*A kör léte többszörös erőforrásoknál nem elégséges feltétel.*)

Példa: holtpont elkerülése

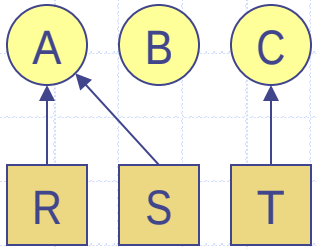
A:
Requests R
Requests S
Release R
Release S
B:
Requests S
Requests T
Release S
Release T
C:
Requests T
Requests R
Release T
Release R



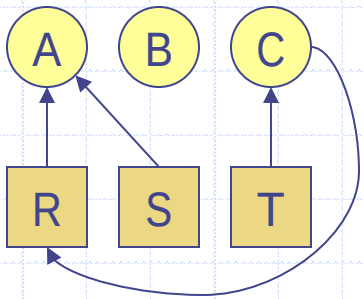
A requests R



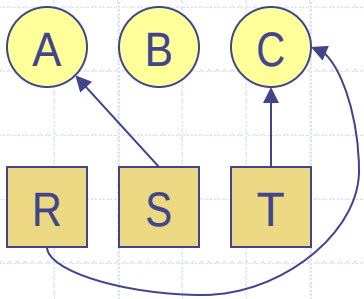
C requests T



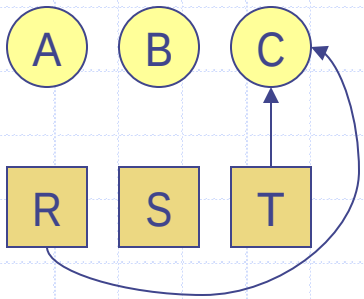
A requests S



C requests R



A releases R



A releases S

4.4 A holtpont kezelése

Stratégiák:

1. Strucc algoritmus: nem veszünk tudomást a problémáról és nem teszünk semmit.
   - Bizonyos típusú rendszereknél (nagy kockázat) nem engedhető meg.
   - Megengedhető kisebb kockázatú rendszereknél, ahol tolerálható a „kiszállunk-beszállunk elv”.
   - Mérlegelni kell a probléma súlyát és a megoldás árát.
2. Védekezés holtpont kialakulása ellen
   - Az erőforrások használatánál bevezetünk szabályokat, ami biztosítja a holtpont elkerülését.
   - holtpont megelőzés (deadlock prevention)
   - holtpont elkerülés (deadlock avoidance)
3. Detektálás/feloldás: a holtpont kialakulása után avatkozunk bele
   - holtpont felismerés (deadlock recognition)
   - holtpont megszüntetés (deadlock recovery)

4.5 Holtpont megelőzés (deadlock prevention)

- A holtpont kialakulásnak szükséges feltételek közül valamelyiket kizárjuk.
  - Kölcsönös kizárás 
  - Foglalva várakozás 
  - Erőforrások nem elvehetők 
  - Körkörös várakozás 

4.5.1 Foglalva várakozás kizárása

Két stratégia:

1. A folyamat elindulásakor egyszerre igényli az összes szükséges erőforrást. Csak akkor mehet tovább, ha mindegyiket megkapta.
2. Folyamat csak akkor igényelhet, ha más erőforrást nem foglal.

- Problémák:
  - rossz erőforrás kihasználtság, szükségesnél tovább birtokolják azokat
  - fennáll a kiéheztetés veszélye: ha egy folyamat több "népszerű" erőforrást használ, nagy az esélye, hogy egyszerre az összes erőforrást soha nem kapja meg.
- Példa: Számítás szalagról nyomtatóra. Erőforrások: szalagos tároló – diszk – nyomtató

4.5.2 Erőforrások elvétele

Két hasonló stratégia

1. Ha egy folyamat valamely erőforrásigénye nem elégíthető ki, akkor az összes többit is elvesszük tőle. Ezekre a továbbiakban várakozik. Akkor futhat tovább, ha az összes erőforrásigényét egyszerre ki lehet elégíteni.
2. Ha egy *P* folyamatnak olyan erőforrásigénye van, amelyeket más *várakozó* $\{Q_i\}$ folyamatok foglalnak, akkor az erőforrásokat Q_i folyamatoktól elveszik és *P* futhat tovább (de csak ha *P* összes igénye egyszerre kielégíthető!), különben pedig *P* is várakozik.

- Problémák:
  - Az erőforrások egy része csak úgy vehető el, ha közben a futási eredmények is elvesznek.
  - Fennáll a kiéheztetés veszélye.

4.5.3 Körkörös várakozás elkerülése

- Rendeljünk a rendszer összes erőforrásához egy növekvő számsorozat egy-egy elemét.
- Két algoritmus:

1. A folyamatok csak növekvő sorrendben igényelhetik az erőforrásokat.
2. A folyamat csak akkor igényelhet egy erőforrást, ha nem használ az igényeltnél magasabb sorszámút.

- Problémák:
  - Nehéz az erőforrásokat olyan módon beszámozni, hogy az tükrözze az erőforrás szokásos sorrendjét.
  - Interaktív rendszereknél nem jó (nem lehet megjósolni a folyamatok erőforrás használatát).
- Logikailag függő folyamatokra alkalmazható.

4.6 Holtpont elkerülése

- A kért erőforrásokat óvatosan allokáljuk. A rendszerminden kérésnél mérlegeli, hogy nem vezet-e holtpontveszélyre a kérés, fenntartható-e a *biztonságos állapot*.

- Feltételezések:
  - Ismerjük a folyamatok erőforrás-típusonkénti max igényeit.
  - Feltételezzük, hogy ha egy folyamat minden erőforrásigényét kielégítettük, az véges időn belül lefut.

4.6.1 Biztonságos állapot

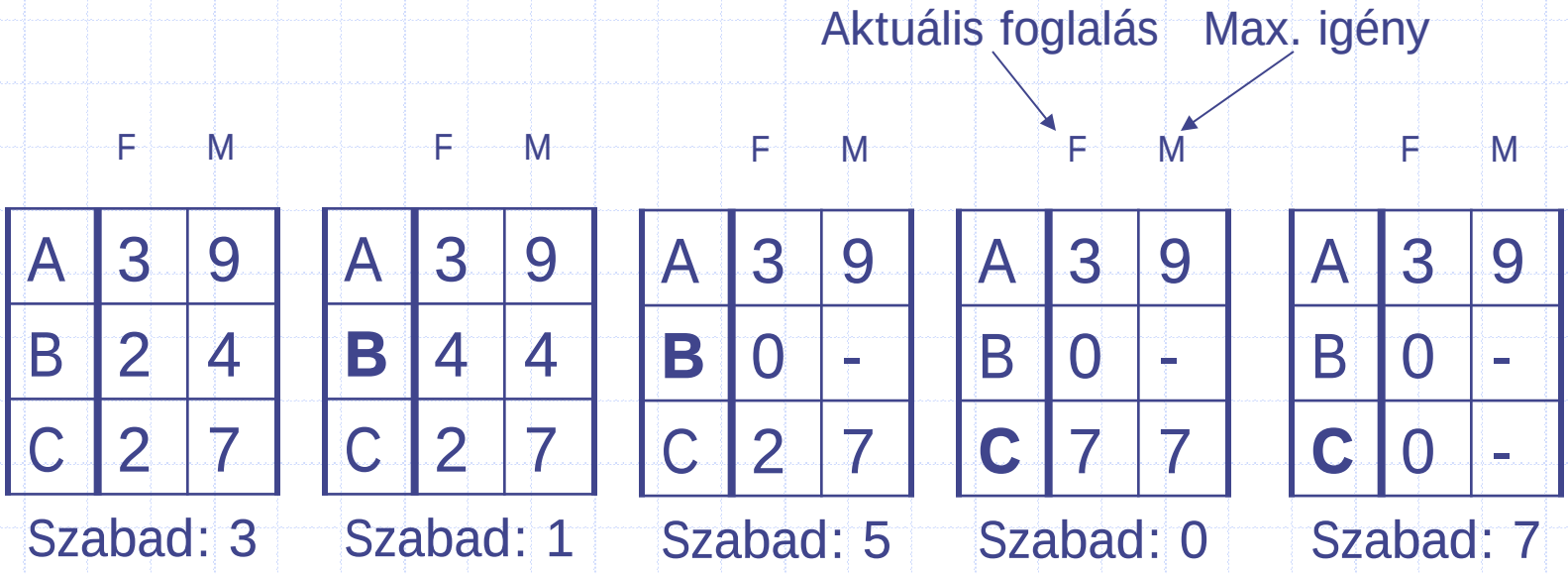
- Biztonságos állapot: létezik az összes folyamatot tartalmazó biztonságos sorozat.
- Biztonságos sorozat: a folyamatok olyan $\{P_0, P_1, \ldots, P_n\}$ sorozata, ahol bármelyik P_k folyamat erőforrásigénye kielégíthető a rendelkezésre álló, valamit a többi P_i ($i < k$) folyamat által használt (és majdan felszabadított) erőforrással.

A biztonságos állapot és a holtpont

- A rendszer biztonságos állapotban van → nincs holtpont
- A rendszer nem biztonságos állapotban van → holtpont lehetséges
- Holtpont elkerülése: biztosítani kell, hogy a rendszer soha nem kerül nem biztonságos állapotba

Példa: biztonságos állapot

- A kiinduló állapot biztonságos
- (csak 1 erőforrás-osztályra)



Példa: nem biztonságos állapot

A második állapot már nem biztonságos!

	F	M
A	3	9
B	2	4
C	2	7

Szabad: 3

	F	M
A	4	9
B	2	4
C	2	7

Szabad: 2

	F	M
A	4	9
B	4	4
C	2	7

Szabad: 0

	F	M
A	4	9
B	-	-
C	2	7

Szabad: 4

4.6.2 Bankár algoritmus (Dijkstra,1965)

Az algoritmus előfeltételei:

1. Az erőforrás-osztályok több egyedből állhatnak
2. Minden folyamat előzetesen megadja maximális igényét
3. Egy igénylő folyamat várakozni kényszerülhet
4. Ha egy folyamat megkapja az igényelt erőforrásait, véges időn belül visszaadja.

Bankár algoritmus - adatszerkezetek

- n = folyamatok száma
- m = erőforrás-osztályok száma
- SZABAD: m elemű vektor. Ha SZABAD[j] = k, akkor az R_j típusú erőforrásból k példány elérhető.
- MAX: n x m méretű mátrix. A MAX[i] m elemű sorvektor jelzi, hogy az egyes erőforrásosztályokból P_i folyamat maximum hány példányt használhat (2. előfeltétel alapján).
- FOGLAL: n x m méretű mátrix. A FOGLAL[i] m elemű sorvektor jelzi, hogy az egyes erőforrásosztályokból P_i folyamat jelenleg hány példányt használ.
- MÉG: n x m méretű mátrix. A MÉG[i] m elemű sorvektor jelzi, hogy az egyes erőforrásosztályokból P_i folyamatnak feladata befejezéséhez még hány példányra lehet szüksége.
- KÉR: n x m méretű mátrix. A KÉR[i] m elemű sorvektor jelzi P_i folyamat kérését az egyes erőforrásosztályokra.
- Nyilvánvalóan: MÉG[i,j] = MAX[i,j] – FOGLAL[i,j].

Bankár algoritmus – kérés feldolgozása

Alapötlet:

- Ha a kérés egyébként teljesíthető, akkor tegyünk úgy, mintha már teljesítettük volna.
- Vizsgáljuk meg, hogy ez az állapot biztonságos-e.
- Ha igen, valóban teljesíthetjük a kérést.

Bankár algoritmus – kérés feldolgozása

Pi folyamat kéri a KÉR[i] erőforrásokat:

A kérés ellenőrzése:

if KÉR[i] > MÉG[i] then STOP; (HIBA: túllépte a maximális igényt)

if KÉR[i] > SZABAD then VÉGE; (Most nincs elég szabad erőforrás)

1. A nyilvántartás átállítása az új állapotra:

   SZABAD := SZABAD - KÉR[i];

   FOGLAL[i] := FOGLAL[i] + KÉR[i];

   MÉG[i] := MÉG[i] - KÉR[i];
2. Vizsgálat: a létrejött állapot biztonságos-e? (lásd később)
3. if BIZTONSÁGOS then

   KÉRÉS TELJESÍTÉSE;

   else állapot visszaállítása a (2) pont előttire:

   SZABAD := SZABAD + KÉR[i];

   FOGLAL[i] := FOGLAL[i]-KÉR[i];

   MÉG[i] := MÉG[i] - KÉR[i];

   KÉRÉS ELUTASÍTÁSA: A FOLYAMATNAK VÁRNIA KELL

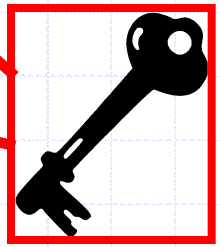
Bankár algoritmus – biztonságos állapot vizsgálata

Működés (biztonságos sorozat keresése):

- Keressünk olyan folyamatot, ami le tud futni a most éppen rendelkezésre álló szabad erőforráskészlettel.
- (Ha nincs ilyen, de van várakozó folyamat, akkor holtpont van)
- A gondolatban lefuttatott folyamat által birtokolt erőforrásokat visszaadjuk, így most már több erőforrással próbálkozhatunk újra.
- Újabb változók:
  - SZABAD_MOST: mint SZABAD. Munkaváltozó.
  - LEFUT: n elemű vektor. Ha LEFUT[j] = igaz, akkor P_j folyamat mindenképpen le tud futni.

Bankár algoritmus – biztonságos állapot vizsgálata

B1. Kezdőérték beállítása:
SZABAD_MOST := SZABAD
LEFUT[i] := hamis minden i-re (i=1,2,...,N)
B2. Továbblépésre esélyes folyamatok keresése:
Keress i-t amelyre (LEFUT[i] = HAMIS AND MÉG[i] <= SZABAD_MOST);
if van ilyen i, then
SZABAD_MOST := SZABAD_MOST + FOGLAL[i];
LEFUT[i] := igaz;
ismételd a B2. lépést
else folytasd a B3. lépéssel
B3. Kiértékelés
if LEFUT[i] = igaz minden i-re (i=1,2,...,N), then
BIZTONSÁGOS
else
NEM BIZTONSÁGOS
(Pi folyamatok, amelyekre LEFUT[i] = hamis holtpontra juthatnak)



Bankár algoritmus – jellemzés

Az algoritmus problémái:

- Időigényes
- Az alapfeltételek (ismert folyamatszám, maximális igények, folyamat biztos befejeződése) nem biztosíthatók.
- Túlzott óvatosság, feleslegesen várakoztat folyamatokat (az erőforrások kihasználtsága rosszabb, mint holtpont elkerülés nélkül).

Bankár algoritmus – példa

- 5 folyamat: P_0, P_1, P_2, P_3, P_4
- 3 erőforrás-osztály: A (10), B (5), C (7)
- A rendszer kiinduló állapota:

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

- Ez biztonságos állapot, mert van egy biztonságos sorozat: pl. $<P_1, P_3, P_4, P_2, P_0>$

Bankár algoritmus – példa

- Teljesíthető-e P_1 (1,0,2) kérése?
- Ellenőrzés:
  - (1,0,2) <= (1,2,2) (KÉR[1] <= MÉG[1]: OK)
  - (1,0,2) <= (3,3,2) (KÉR[1] <= SZABAD: OK)
- Kiinduló állapot:

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Bankár algoritmus – példa

- Kérés teljesítése (1,0,2)

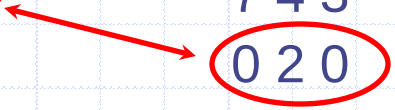
Az algoritmusban ez a SZABAD_MOST átmeneti változóban tárolódik

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	2 3 0	7 4 3
P_1	3 0 2	3 2 2		0 2 0
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Bankár algoritmus – példa

- Biztonságos sorozat keresése 1.
  - P_1 le tud futni

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	2 3 0	7 4 3
P_1	3 0 2	3 2 2		0 2 0
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1



Bankár algoritmus – példa

- Biztonságos sorozat keresése 1.
  - P_1 lefut és ...

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	2 3 0	7 4 3
P_1	3 0 2	3 2 2		0 2 0
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Bankár algoritmus – példa

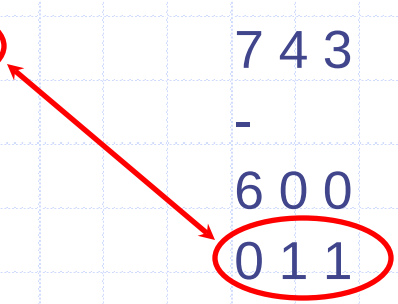
- Biztonságos sorozat keresése 1.
  - P_1 lefut és erőforrásait visszaadja

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	5 3 2	7 4 3
P_1	0 0 0	3 2 2		-
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Bankár algoritmus – példa

- Biztonságos sorozat keresése 2.
  - P_3

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	5 3 2	7 4 3
P_1	0 0 0	3 2 2		-
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1



Bankár algoritmus – példa

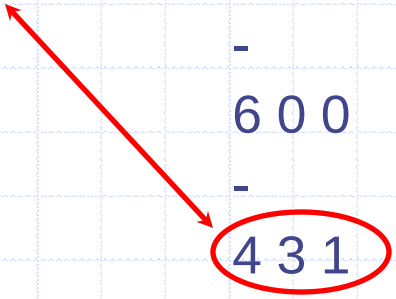
- Biztonságos sorozat keresése 2.
  - P_3 lefut

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	7 4 3	7 4 3
P_1	0 0 0	3 2 2		-
P_2	3 0 2	9 0 2		6 0 0
P_3	0 0 0	2 2 2		-
P_4	0 0 2	4 3 3		4 3 1

Bankár algoritmus – példa

- Biztonságos sorozat keresése 3.
  - P_4

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	7 4 3	7 4 3
P_1	0 0 0	3 2 2		-
P_2	3 0 2	9 0 2		6 0 0
P_3	0 0 0	2 2 2		-
P_4	0 0 2	4 3 3		4 3 1



Bankár algoritmus – példa

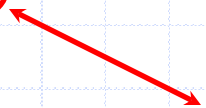
- Biztonságos sorozat keresése 3.
  - P_4 lefut

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	7 4 5	7 4 3
P_1	0 0 0	3 2 2		-
P_2	3 0 2	9 0 2		6 0 0
P_3	0 0 0	2 2 2		-
P_4	0 0 0	4 3 3		-

Bankár algoritmus – példa

- Biztonságos sorozat keresése 4.
  - P_2

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	7 4 5	7 4 3
P_1	0 0 0	3 2 2		-
P_2	3 0 2	9 0 2		6 0 0
P_3	0 0 0	2 2 2		-
P_4	0 0 0	4 3 3		-



Bankár algoritmus – példa

- Biztonságos sorozat keresése 4.
  - P_2 lefut

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	10 4 7	7 4 3
P_1	0 0 0	3 2 2		-
P_2	0 0 0	9 0 2		-
P_3	0 0 0	2 2 2		-
P_4	0 0 0	4 3 3		-

Bankár algoritmus – példa

- Biztonságos sorozat keresése 5.
  - P_0

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	10 4 7	7 4 3
P_1	0 0 0	3 2 2		-
P_2	0 0 0	9 0 2		-
P_3	0 0 0	2 2 2		-
P_4	0 0 0	4 3 3		-

Bankár algoritmus – példa

- Biztonságos sorozat keresése 5.
  - P_0 lefut

	Foglal	*Max*	*Szabad*	*Még*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 0 0	7 5 3	10 5 7	-
P_1	0 0 0	3 2 2		-
P_2	0 0 0	9 0 2		-
P_3	0 0 0	2 2 2		-
P_4	0 0 0	4 3 3		-

Biztonságos sorozat: $< P_1, P_3, P_4, P_2, P_0 >$

Bankár algoritmus – gyakorló példák

1. Teljesíthető-e P_4 (3,3,0) kérése?
2. Teljesíthető-e P_0 (0,2,0) kérése?

Kiinduló állapot:

	Foglal	*Max*	*Szabad*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

4.7 A holtpont felismerése

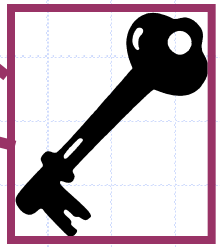
- Egyszeres erőforrások esete
  - Kör keresése az erőforrás-foglalási gráfban
- Többszörös erőforrások esete
  - Itt a kör nem elégséges feltétel, tehát bonyolultabb vizsgálat kell.
  - Hasonlóan a biztonságos állapot vizsgálatánál közölt algoritmushoz (vigyázat: nem ugyanaz az algoritmus!)

Coffman algoritmus - adatszerkezetek

- N = folyamatok száma
- M = erőforrás-osztályok száma
- SZABAD: m elemű vektor. Ha SZABAD[j] = k, akkor az R_j típusú erőforrásból k példány elérhető.
- SZABAD_MOST: mint SZABAD. Munkaváltozó.
- FOGLAL: n x m méretű mátrix. A FOGLAL[i] m elemű sorvektor jelzi, hogy az egyes erőforrásosztályokból P_i folyamat jelenleg hány példányt használ.
- KÉR: n x m méretű mátrix. A KÉR[i] m elemű sorvektor jelzi P_i folyamat kérését az egyes erőforrásosztályokra.
- LEFUT: n elemű vektor. Ha LEFUT[j] = igaz, akkor a P_j folyamat *a mostani erőforrásigények szerint* le tud futni.

Holtpont-detektáló algoritmus (Coffman algoritmus)

C1. Kezdőérték beállítása:
SZABAD_MOST := SZABAD
LEFUT[i] := hamis minden i-re (i=1,2,...,N)
C2. Továbblépésre esélyes folyamatok keresése:
Keress i-t amelyre (LEFUT[i] = HAMIS AND KÉR[i] <= SZABAD_MOST);
if van ilyen i, then
SZABAD_MOST := SZABAD_MOST + FOGLAL[i];
LEFUT[i] := igaz;
ismételd a C2. lépést
else folytasd a C3. lépéssel
C3. Kiértékelés
if LEFUT[i] = igaz minden i-re (i=1,2,...,N), then
NINCS HOLTPONT
else
HOLTPONT:
Azon Pi folyamatok, amelyekre LEFUT[i] = hamis



Coffman algoritmus – példa

- Kiinduló állapot:

	Foglal	*Kér*	*Szabad*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Van-e holtpont?
  - Nincs: $<P_0, P_2, P_3, P_1, P_4>$
- Mi a helyzet, ha P_2 kér még egy C-t?
  - Holtpont: (P_1, P_2, P_3, P_4)

A holtpont-detektálás időzítése

Mikor kell a detektáló algoritmust futtatni?

- Erőforrás-igény kielégítésekor
  - Sok időt igényel
  - Rögtön kiderül a holtpont
  - Tudjuk, melyik folyamat zárta be a holtpontot
- Meghatározott időnként
  - Nehéz meghatározni a megfelelő gyakoriságot
  - Túl sűrű futtatás: sok időt igényel
  - Túl ritka futtatás: holtpont sokáig fennáll

4.8 A holtpont felszámolása

- Holtpont felismerése után azt - operátori beavatkozással vagy automatikusan - fel kell számolni.

- Módszerek:
  1. Folyamatok terminálása
  2. Erőforrások elvétele

A holtpont felszámolása: folyamatok terminálása

- Minden holtpontban résztvevő folyamatot megszüntetünk (radikális)
  - Biztos, de költséges (megszűnt folyamatok eredményei elvesznek)
- Egyesével szüntetjük meg folyamatokat, amíg a holtpont meg nem szűnik.
  - Minden terminálás után újabb detektálás
  - Szempontok a folyamat kiválasztásához:
    - hány holtpont körben szerepel
    - mekkora a prioritása
    - mennyi ideje futott, mennyit futna még (ha ismerjük)
    - mennyi erőforrást tart lefoglalva
    - mennyi további erőforrásra lenne szüksége
    - interaktív vagy batch program

A holtpont felszámolása: erőforrások elvétele

- A holtpontra jutott folyamatoktól egyesével elvesszük az erőforrásokat.
- Megoldandó problémák:
  - Kitől vegyük el és melyik erőforrást?
  - Kiszemelt folyamatot vissza kell léptetni egy olyan állapotba, ahonnan a futását folytatni tudja (leggyakrabban újra kell kezdenie a futását). Egyes OS-ek a folyamatokhoz ellenőrzési pontokat rendelnek (checkpoint); a futást a legutóbbi ellenőrzési ponttól kell folytatni.
  - El kell kerülni a folyamatok kiéheztetését (ne mindig ugyanattól a folyamattól vegyünk el erőforrásokat).

4.9 Kombinált holtpont-kezelő stratégiák

Erőforrásokat csoportokra osztjuk, és az egyes csoportokra különböző stratégiát alkalmazunk. Pl.:

ME
- Belső erőforrások (rendszertáblák, leírók, stb.)
  - Rendszer-folyamatokat érinti. Folyamatok létrejöttekor azonos sorrendben igénylik ezen erőforrásokat, a rendezett erőforrás-foglalás könnyen megvalósítható.

D/F
- Operatív tár
  - Menthető állapotú erőforrás, így alkalmazható az erőszakos elvétel módszere.

EK
- Egy munkához tartozó erőforrások (állományok, eszközök)
  - A munka leírásából ismerhetjük az igényeket, használhatunk holtpont elkerülő algoritmust.

ME
- Munkaterület a lemezen (swap)
  - általában ismert igények vannak, egyszerre kell kérni a szükséges méretet, nincs rákérés (előzetes lefoglalás).

ME

4.10 Kommunikációs holtpontok

- Holtponthelyzet nemcsak erőforrás-használat miatt alakulhat ki, hanem folyamatok tetszőleges olyan együttműködése során, amely a folyamatok körkörös várakozásra vezet.
- Pl.
  - Kliens-szerver architektúrájú rendszer, ahol az ügyfelek és a szolgáltatók is folyamatok. Az ügyfél-szolgáltató lánc záródik.
- A gráfos reprezentáció itt is használható: várakozási gráf (wait-for graph)
  - csomópontjai a folyamatok
  - irányított élei pedig a várakozást jelzik (várakozóból a várakoztatóhoz vezet).