Operačné systémy / Uviaznutie

Dušan Bernát
bernat@fmph.uniba.sk

Prehľad

- Definícia uviaznutia.
- Podmienky vzniku.
- Riešenia, detekcia, prevencia, vyhýbanie sa.
- Príklady.
- Reálne príklady.

Čo je uviaznutie?

Uviaznutie

- Uviaznutie vznikne ak proces čaká na udalosť ktorá nemôže nastať.
  - Bude teda čakať nekonečne dlho a nikdy sa neukončí.
- Uviaznutie nie je chyba ktorá by sa prejavila chybovým hlásením, alebo nesprávnym výsledkom.
- Uviaznutý proces čaká (či už aktívne, alebo je zablokovaný).
  - Proces nemá možnosť rozoznať či sa čakanie ukončí, alebo nie.
- Definícia [Tanenbaum]:
  - Množina procesov je v stave uviaznutia, ak každý čaká na udalosť, ktorú môže spôsobiť len iný proces z tejto množiny.
- Procesy väčšinou čakajú na prostriedky, ktoré majú pridelené iné čakajúce procesy.
- Predpokladá sa, že procesy nečakajú na nič iné.
- Nie každé “zaseknutie” procesu je uviaznutím.

Nutné podmienky vzniku uviaznutia (Coffman, 1971)

1) Vzájomné vylučovanie (*Mutual exclusion*)
   - Prostriedky nie je možné zdieľať. Prostriedok je vždy pridelený práve jednému procesu, alebo je voľný.
2) Nepreemptívne prideľovanie prostriedkov (*Non preemption condition*)
   - Pridelený prostriedok nie je možné procesu odobrať. Proces musí prostriedok uvoľniť sám.
3) Čiastočné prideľovanie prostriedkov (*Hold and wait condition*)
   - Proces môže o prostriedky žiadať postupne. Proces už má pridelenú časť prostriedkov ktoré potrebuje, zatiaľ čo čaká na ďalšie.
4) Čakanie na prostriedky (*Circular wait condition*)
   - Ak proces nemá pridelené prostriedky o ktoré žiadal, čaká.
   - Existuje postupnosť procesov (P_1, P_2, ..., P_n) taká, že $P_i \rightarrow P_{i+1}$ pre $i<n$ a $P_n \rightarrow P_1$, kde relácia $P_i \rightarrow P_j$ znamená, že proces P_i čaká na prostriedok pridelený procesu P_j.

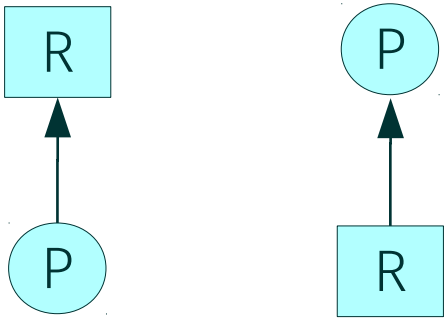
Podmienky vzniku uviaznutia

- Prvé tri podmienky sú nutnými podmienkami. Ak odstránime platnosť ľubovoľnej z nich, uviaznutie nemôže nastať.
- Ak je splnená aj štvrtá podmienka, uviaznutie nastane (nastalo).
  - Ak nastalo kruhové čakanie a platia prvé tri podmienky, tak je toto čakanie neodstrániteľné.
- Spolu teda uvedené podmienky predstavujú nutné aj postačujúce podmienky vzniku uviaznutia.
- Model správania procesu:
  - Proces spravidla uvoľní alokované prostriedky keď sa ukončí.
  - Na to, aby sa proces ukončil, musí najskôr alokovať všetky potrebné prostriedky. Prostriedky môže proces alokovať postupne.
  - Na pridelenie prostriedkov čaká (bez nich sa nemôže vykonávať ďalej).

Modelovanie uviaznutia

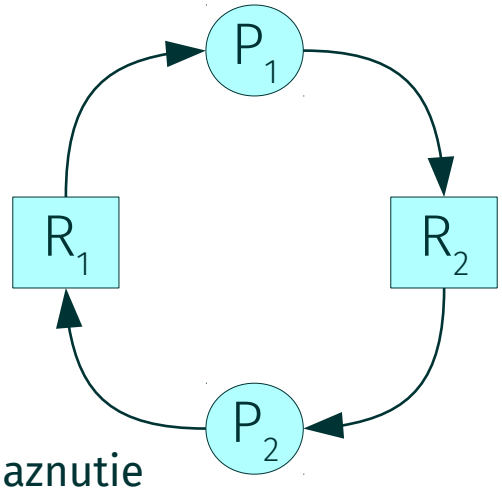
Graf alokácie prostriedkov

- Graf alokácie prostriedkov (*Resource allocation graph*) je bipartitný orientovaný graf.
  - Vrcholy predstavujú množinu procesov *P* a množinu prostriedkov *R*.
  - Hrana *P* → *R* znamená, že proces *P* potrebuje prostriedok *R* (a bude naň čakať), resp. *P* čaká na udalosť *R*.
  - Hrana *R* → *P* znamená, že proces *P* má pridelený prostriedok *R* (a uvoľní ho, až keď sa ukončí), resp. *P* generuje *R*.



P čaká na R

P má R

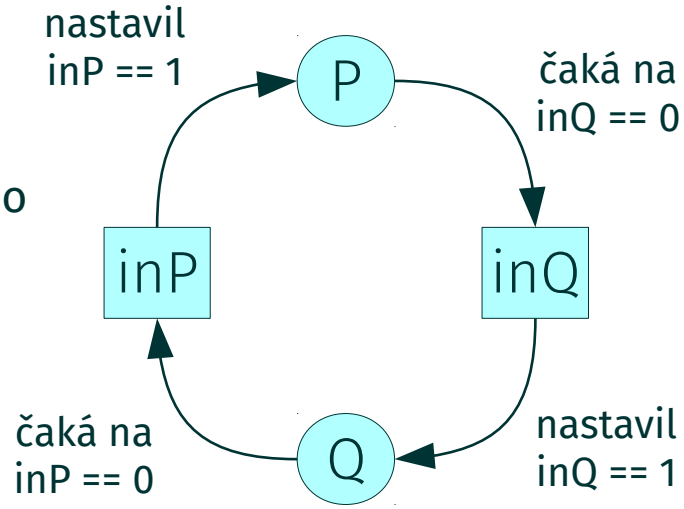


uviaznutie

Príklad: Pokus o vzájomné vylučovanie

```
1: ProcessP() {                    ProcessQ() {
2:    while(1) {                      while(1) {
3:       inP = TRUE;                     inQ = TRUE;
4:       while(inQ);                     while(inP);
5:       UseResource();                  UseResource();
6:       inP = FALSE;                    inQ = FALSE;
7:    }                               }
8: }                               }
```

- Spoločnými prostriedkami sú tu zdieľané premenné.
- Udalosť, na ktorú sa tu čaká, je nastavenie na nulu.
- Všetky plánovania obsahujúce podpostupnosť (P3, Q3) alebo (Q3, P3) vedú k cyklickému čakaniu.
- K uviaznutiu nemusí dôjsť vždy. Ostatné plánovania budú bez uviaznutia.

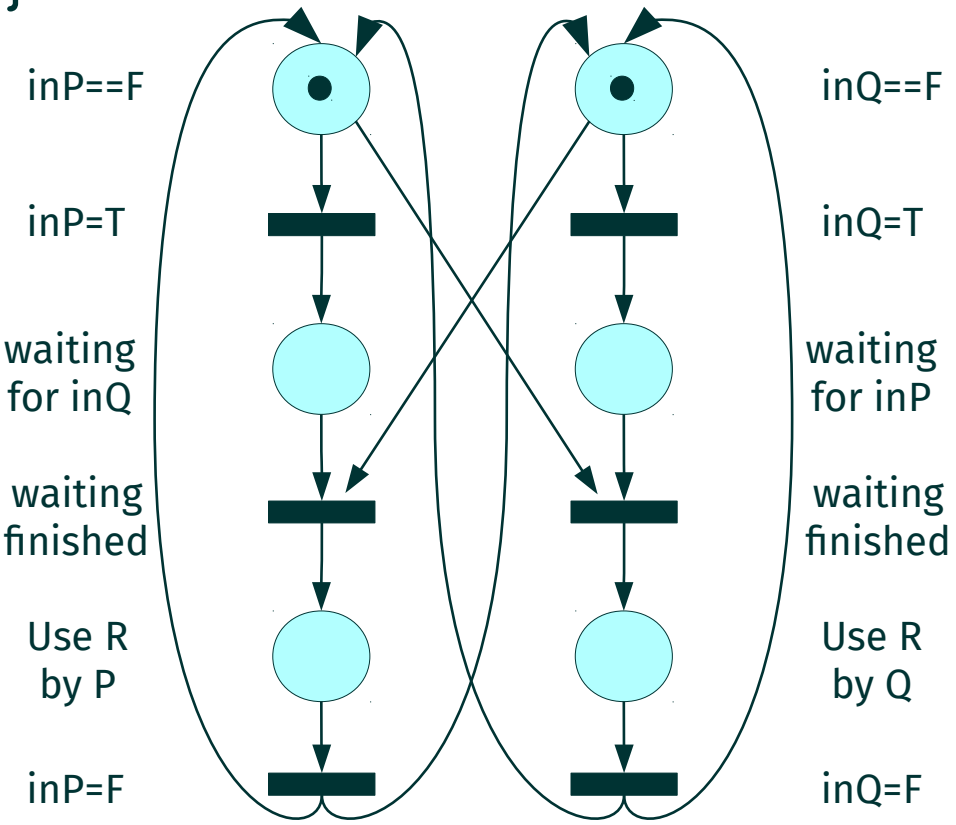


Pokus o vzájomné vylučovanie (Petriho siete)

```
1: ProcessP() {
2:    while(1) {
3:       inP = TRUE;
4:       while(inQ);
5:       UseResource();
6:       inP = FALSE;
7:    }
8: }
```

```
ProcessQ() {
   while(1) {
      inQ = TRUE;
      while(inP);
      UseResource();
      inQ = FALSE;
   }
}
```

- Oba prechody **inP=T** a **inQ=T** sú pripravené súčasne.
- Pokiaľ sa oba vykonajú skôr než jeden z nich vojde do KO, žiadny prechod už nebude pripravený.
- Ak sú tokeny v oboch čakacích miestach, nastalo uviaznutie.

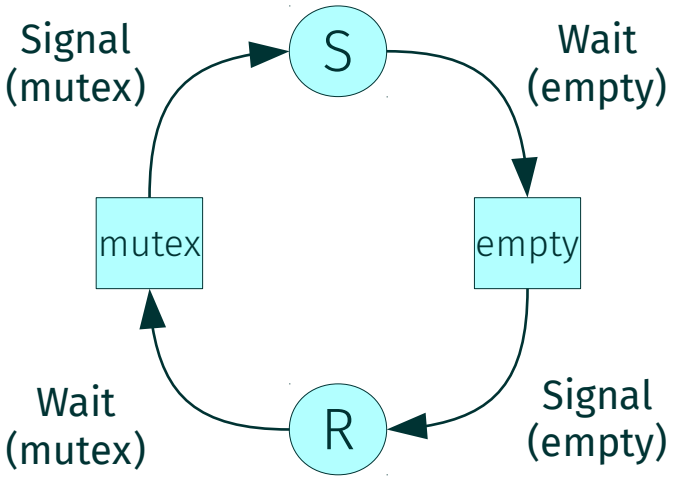


Príklad: Rad pevnej dĺžky

```
Send(message msg) {
    Wait(mutex);
    Wait(empty);
    enqueue(q, msg);
    Signal(mutex);
    Signal(full);
};
```

```
Receive(message *msg) {
    Wait(full);
    Wait(mutex);
    *msg = dequeue(q);
    Signal(mutex);
    Signal(empty);
}
```

- Implementácia radu s pevnou dĺžkou pomocou semaforov.
- Vymenené poradie čakania.
- Existuje taká postupnosť vykonania, ktorá vedie k cyklickému čakaniu.
  - Ak je rad plný a zavolá sa `Send()`.
- Pri pôvodnom (správnom) poradí to nie je možné.



Riešenia problému uviaznutia

Prístupy k riešeniu uviaznutia

- Operačný systém môže riadiť prístup len k prostriedkom ktoré poskytuje (CPU, pamäť, diskový priestor, súbory, semafory, ...).
  - Mnohé z nich sú reprezentované tabuľkami v pamäti.
- Nemá však vplyv na prostriedky implementované priamo procesmi (napr. spoločnými premennými v ich pamäti).
  - Jadro nevie, že nejaká konkrétna adresa v spoločnej pamäti môže pre procesy predstavovať zdieľaný prostriedok.
- Možné prístupy k problému uviaznutia väčšinou delíme do týchto skupín:
  1) Ignorovanie problému
  2) Detekcia uviaznutia a zotavenie
  3) Predchádzanie uviaznutiu
  4) Vyhýbanie sa uviaznutiu

Ignorovanie uviaznutia

- Nie je to skutočné riešenie, je to však možný prístup.
- Uviaznutie môže byť menej problematické, než (nepovšimnutá) nekonzistencia dát.
- Náklady na návrh a realizáciu riešenia môžu byť značne vyššie než odstránenie následkov uviaznutia.
- Z praktického hľadiska môže byť výhodnejšie tento problém ignorovať.
  - Uviaznutie procesov v dôsledku vyčerpania prostriedkov OS (napr. tabuľka procesov) je nepravdepodobné a zriedkavé.
- Mnohé používané OS uviaznutie ignorujú.
  - Linux má detekciu uviaznutia napr. pre zámky na súboroch (POSIX *locks*), FUTEX_LOCK_PI (*priority-inheritance lock*), aj pre *spinlock*, *rwlock*, *mutex* v jadre (modul *lockdep*).

Detekcia uviaznutia a zotavenie

- Pri tomto prístupe môže dôjsť k uviaznutiu. Keď uviaznutie nastane, OS reštartuje niektorý z množiny uviaznutých procesov.
- Nie je potrebná žiadna zmena v implementácii procesov.
- OS musí mať implementovaný detekčný mechanizmus:
  1) V najjednoduchšom prípade to môže byť operátor, ktorý stav zistí a zasiahne.
     - Nie sú potrebné žiadne zmeny ani v OS.
  2) Jednoduchý mechanizmus môže napríklad zrušiť proces zablokovaný určenú (dlhú) dobu.
  3) Do operácií alokácie a uvoľnenia prostriedkov sa pridá úprava alokačného grafu. Ak sa v ňom nájde slučka, nastalo uviaznutie.

Detekcia uviaznutia a zotavenie

- Zotavenie:
  - Zrušenie všetkých uviaznutých procesov,
  - zrušenie jedného z nich,
  - návrat do neuviaznutého stavu (*checkpoint & restart*).
- Napríklad Linux má implementovanú detekciu uviaznutia aj v jadre (bez zotavenia):
  - Modul *lockdep* implementuje detekciu uviaznutia aj pre *spinlock*, *rwlock*, *mutex* v jadre.
  - Použitie napr. pre ovládače zariadení, alebo iné časti jadra (nie pre procesy). Skôr pre vývojárov OS než aplikačných programátorov.

Predchádzanie uviaznutiu

Predchádzanie uviaznutiu

- Implementáciu upravíme tak, aby niektorá z nutných podmienok nebola splnená.
  - V systéme natrvalo zrušíme platnosť (aspoň) jednej zo štyroch podmienok.
  - Vznik uviaznutia je tým štrukturálne vylúčený.
- Uviaznutie nemôže vôbec nastať bez ohľadu na to, ako sú procesy plánované.
- Riešenie by malo mať niektorú z týchto vlastností:
  - Nepoužívať vzájomné vylučovanie.
  - Používať preempciu, teda možnosť odobrať už pridelený prostriedok.
  - Alokovať všetky potrebné prostriedky vždy naraz.
  - Zabraňovať vzniku kruhového čakania na prostriedky.
- Prvé tri predstavujú nepriame a štvrtá priame predchádzanie uviaznutiu.
- Tieto podmienky sú značne obmedzujúce, môžu viesť k neefektívnym alebo aj komplikovaným riešeniam.

(1) Odstránenie vzájomného vylučovania

- Prístup k spoločnému prostriedku môže byť sprostredkovaný koordinačným procesom, ktorý bude k nemu pristupovať ako jediný.
- Ostatné procesy budú komunikovať s koordinačným procesom, napríklad zasielaním správ.
- Procesy nebudú súťažiť o prístup medzi sebou.
- Príklad: *print spooler*
  - Namiesto toho, aby každý proces ktorý potrebuje tlačiareň si ju alokoval na čas tlačenia sám, pošle úlohu vyhradenému procesu.
- Tento prístup nie je možné aplikovať na každý typ prostriedku.
  - Napríklad tabuľky v jadre OS (tabuľka procesov, i-uzlov, ...).
- Môže to viesť k vzniku novej kritickej oblasti.

(2) Preemptívne prideľovanie prostriedkov

- Preempcia – odoberanie prostriedku ktorý už bol pridelený (bez toho, aby ho proces ktorý ho má uvoľnil).
- Už pridelené prostriedky by sa niektorým procesom odobrali, aby ich bolo možné prideliť inému procesu, ktorý na ne čaká.
- Ten by sa následne ukončil a prostriedky uvoľnil.
  - Alebo by mu boli opäť odobrané.
- Odobratie prostriedku ktorý jeden proces využíva a jeho pridelenie inému procesu môže mať fatálne dôsledky
  - Vedie k nesprávnej funkcii a zlým výsledkom.
- Tento prístup je možné využiť pre zdroje ktorých stav je možné uchovať.
  - OS ho využíva napríklad pre CPU a fyzickú pamäť.

(3) Pridelenie všetkých prostriedkov

- Uviaznutie môže nastať ak má každý proces pridelenú časť prostriedkov, ale ešte musia čakať na ďalšie, aby sa mohli dokončiť.
- Pokiaľ by proces dostal všetky prostriedky potrebné na svoje vykonanie naraz, táto situácia by nenastala.
  - Ak bude musieť čakať tak bez toho, aby sám ostatným procesom nejaké prostriedky odoberal.
- Problém je v tom, že mnohé procesy vopred nevedia, koľko prostriedkov budú potrebovať.
- Alternatívne by proces mohol vždy pred alokovaním ďalších prostriedkov najskôr dočasne uvoľniť všetky ktoré už má a potom požiadať znovu o zväčšený objem prostriedkov naraz.

(4) Eliminovanie cyklického čakania

- Zrušenie tejto podmienky je možné dosiahnuť rôznymi spôsobmi:
- Len jeden proces. Príliš obmedzujúce.
  - Aj ten by mohol uviaznuť v čakaní na prostriedok ktorý už má – rekurzívne získanie zámku, napríklad v obsluhe prerušenia.
- Len jeden prostriedok. Ak by chcel proces alokovať iný prostriedok, ten prvý by najskôr musel uvoľniť. To však nemusí byť vždy možné.
- Prístup k prostriedkom v určenom poradí.
  - Všetky prostriedky sú zoradené (očíslované). Každý proces smie žiadať o prostriedky len v poradí ich čísiel (nie o nižší).
  - Týmto spôsobom nemôže vzniknúť cyklus v grafe alokácií prostriedkov.
  - Alternatívne by proces smel žiadať o prostriedok s vyšším číslom, než práve má. Teda ak prostriedok s vyšším číslom uvoľní, môže opäť žiadať o nižší.
  - Nájsť vhodné usporiadanie všetkých prostriedkov ktoré by umožňovalo vykonanie všetkým procesom nemusí byť možné.

Vyhýbanie sa uviaznutiu

Vyhýbanie sa uviaznutiu

- Systém síce umožňuje vznik uviaznutia, OS (a procesy) sa ale správajú tak, aby k uviaznutiu nedošlo.
  - Vyhýbajú sa mu počas vykonávania.
- Žiadna z nutných podmienok vzniku uviaznutia nie je trvale odstránená, ale procesy sa vyhýbajú tomu, aby platili všetky súčasne.
  - V každom čase teda aspoň jedna neplatí.
- Tento prístup je menej obmedzujúci než prevencia uviaznutia.
- Procesy musia byť implementované s ohľadom na vyhýbanie sa uviaznutiu.
- Procesy musia vopred vedieť poskytnúť informácie o potrebných prostriedkoch.
  - Napr. koľko jednotlivých prostriedkov celkovo potrebujú na svoje vykonanie.

Odmietnutie spustenia procesu

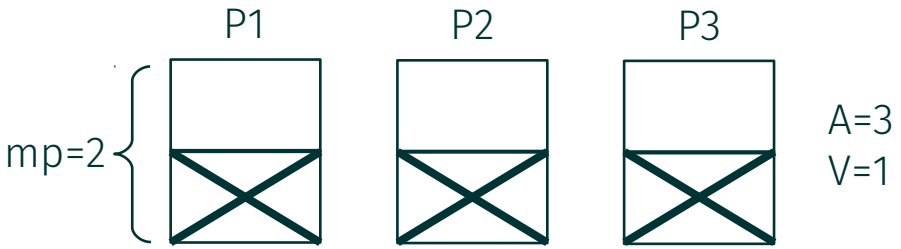
- Máme *n* procesov a *m* prostriedkov (*i=1..n*, *j=1..m*).
- Stav je daný
  - $A=(a_{ij})$, aktuálne prideleným množstvom prostriedku *j* procesu *i*,
  - počtom voľných prostriedkov $V=(v_1, v_2, ..., v_m)$,
  - $MP=(mp_{ij})$, maximálnou potrebou prostriedku *j* pre proces *i*.
- Celková kapacita jednotlivých prostriedkov: $c_j = v_j + \sum_{i=1}^{n} a_{ij}$
- Platí: $a_{ij} \leq mp_{ij} \leq c_j, \forall\, i=1,\dots n, j=1,\dots,m$
- Nový proces P_{n+1}, ktorý potrebuje mp_{n+1j} jednotlivých prostriedkov sa spústí len ak platí: $c_j \geq mp_{n+1\,j} + \sum_{i=1}^{n} mp_{ij}, \forall\, j=1,\dots,m$
  - Teda ak celkové požiadavky nového a bežiacich procesov nepresiahnu dostupnú kapacitu jednotlivých prostriedkov. Inak hrozí uviaznutie.

Príklad 1

- V systéme sú štyri prostriedky jedného typu zdieľané tromi procesmi. Každý proces potrebuje najviac dva prostriedky.
- Môže nastať uviaznutie?
- $n=3$, $c=4$, $mp_i=2$ pre $i=1, 2, 3$.

Príklad 1

- V systéme sú štyri prostriedky jedného typu zdieľané tromi procesmi. Každý proces potrebuje najviac dva prostriedky.
- Môže nastať uviaznutie?
- $n=3$, $c=4$, $mp_i=2$ pre $i=1, 2, 3$.
- [Len jeden typ prostriedku, o jeden index menej.]
- Najhorší prípad:
  - Každý proces má o jeden prostriedok menej než potrebuje ($a_i = mp_i - 1$).
  - Všetky procesy teda čakajú, pričom obsadzujú najviac prostriedkov.
  - Zostane dosť voľných prostriedkov, aby sa aspoň jeden proces dokončil?
- $A = n \,.\, (mp_i - 1) \;\rightarrow\; A = 3 \times (2 - 1) = 3 < c \;\rightarrow\; V = c - A = 4 - 3 = 1 > 0$
- Aj v najhoršom prípade zostanú voľné prostriedky.
- <u>Uviaznutie nemôže nastať.</u> ■



Príklad 2

- V systéme je M prostriedkov jedného typu a N procesov. Procesy žiadajú a uvoľňujú prostriedky jeden po druhom. Každý potrebuje 1 .. M prostriedkov, pričom platí, že suma pridelených prostriedkov je vždy menšia než M+N.
- Môže nastať uviaznutie?

Príklad 2

- V systéme je M prostriedkov jedného typu a N procesov. Procesy žiadajú a uvoľňujú prostriedky jeden po druhom. Každý potrebuje 1 .. M prostriedkov, pričom platí, že suma pridelených prostriedkov je vždy menšia než M+N.
- Môže nastať uviaznutie?
- Suma pridelených prostriedkov $A = \sum_{i=1}^{N} a_i < M + N$
- Celkový objem prostriedkov $c = M$, voľné prostriedky $V = c - A$.
- Nevieme, koľko najviac jeden proces potrebuje, ale bude to najviac M, $1 \leq pm_i \leq M$.
- Najhorší prípad:
  - Každý proces má o jeden prostriedok menej než potrebuje ($a_i = mp_i - 1$).
- $V = M - (mp_1 - 1 + mp_2 - 1 + \ldots + mp_n - 1)$
- $V = M - (A - N) = M + N - A$, a zároveň platí $A < M + N \rightarrow V > 0$
- Uviaznutie nemôže nastať. ■

Bankárov algoritmus

Bankárov algoritmus (Dijkstra, 1965)

- Odmietnutie pridelenia prostriedku.
- Operačný systém (bankár) sa rozhoduje, či klientovi (procesu) dá pôžičku v danej mene (pridelí prostriedok daného typu).
- Každý proces musí na začiatku oznámiť celkové množstvo jednotlivých prostriedkov potrebných na svoje dokončenie.
  - To je prakticky len ťažko splniteľné.
  - Po ukončení proces musí pridelené prostriedky vrátiť. (Ani toto nemusí byť vždy splnené, napr. súbory.)
  - Bankárov algoritmus teda nie je prakticky veľmi použiteľný.
- Bankár prostriedky poskytne len ak existuje postupnosť, ktorá vedie k uspokojeniu všetkých procesov.
  - Stav je bezpečný ak vedie do ďalšieho bezpečného stavu.
  - Prostriedky budú pridelené, len ak ich pridelenie vedie do bezpečného stavu.

Bankárov algoritmus (Dijkstra, 1965)

- Princíp: snažíme sa postupne uspokojiť jednotlivé procesy a zistiť, či môže nastať uviaznutie.

1) Každý proces P_i na začiatku deklaruje maximálny objem potrebných prostriedkov mp_{ij}. Voľné prostriedky $T := V$.
2) Nájsť proces P_i, ktorého aktuálna potreba je menšia než aktuálne voľné prostriedky, čiže P_i: $mp_{ij} - a_{ij} \leq t_j$ pre všetky prostriedky $j=1, \ldots, m$.
   - Ak taký neexistuje, stav nie je bezpečný a **môže dôjsť k uviaznutiu**.
3) Proces P_i môže dostať všetky potrebné prostiredky, **ukončí sa** a prostriedky vráti; $t_j := t_j + a_{ij}$ pre všetky prostriedky j.
4) Opakovať kroky 2) a 3).
   - Ak sú už všetky procesy ukončené, **stav je bezpečný**.
   - Ak nie, môže nastať uviaznutie, **stav nie je bezpečný**.

Príklad 1 (Bankárov algoritmus)

- Prostriedok jedného typu. Je stav bezpečný?
- Voľné $V = 2$
- Prostriedky potrebné na ukončenie $r_i = mp_i - a_i$

	mp_i	a_i	r_i
P0	7	0	7
P1	3	2	1
P2	9	3	6
P3	2	2	0
P4	4	0	4

Príklad 1, riešenie (Bankárov algoritmus)

- Prostriedok jedného typu. Je stav bezpečný?
- Voľné $V = 2$
- Prostriedky potrebné na ukončenie $r_i = mp_i - a_i$

	mp_i	a_i	r_i	
P0	7	0	7	**5)** $a_0 = 7, V = 2 \rightarrow a_0 = 9, V = 9$
P1	3	2	1	**2)** $a_1 = 3, V = 3 \rightarrow a_1 = 0, V = 6$
P2	9	3	6	**4)** $a_2 = 9, V = 0 \rightarrow a_2 = 0, V = 9$
P3	2	2	0	**1)** $a_3 = 2, V = 0 \rightarrow a_3 = 0, V = 4$
P4	4	0	4	**3)** $a_4 = 4, V = 2 \rightarrow a_4 = 0, V = 6$

- Napríklad postupnosť *(P3, P1, P4, P2, P0)* vedie k ukončeniu všetkých procesov.
- Stav je bezpečný. ■

Príklad 2 (Bankárov algoritmus)

- Prostriedok jedného typu. *P4* žiada 3. Je možné prostriedky prideliť?
- Voľné $V = 3$
- Prostriedky potrebné na ukončenie $r_i = mp_i - a_i$

	mp_i	a_i	r_i
P0	7	0	7
P1	3	2	1
P2	9	3	6
P3	2	2	0
P4	4	0	4

Príklad 2, riešenie (Bankárov algoritmus)

- Prostriedok jedného typu. *P4* žiada 3. Je možné prostriedky prideliť?
- Voľné $V = 3$
- Prostriedky potrebné na ukončenie $r_i = mp_i - a_i$

	mp_i	a_i	r_i	
P0	7	0	7	
P1	3	2	1	**3)** $a_1 = 3, V = 2 \rightarrow a_1 = 0, V = 5$
P2	9	3	6	
P3	2	2	0	**2)** $a_3 = 2, V = 0 \rightarrow a_3 = 0, V = 3$
P4	4	0	4	**1)** $a_4 = 2, V = 1 \rightarrow a_4 = 2, V = 1$

- Po pridelení požadovaných prostriedkov procesu *P4* by sa procesy *P0* a *P2* nemohli vykonať.
- Požiadavka nevedie k bezpečnému stavu.
- Požiadavka bude zamietnutá, respektíve proces *P4* bude musieť na alokáciu čakať. ■

Detekcia uviaznutia v jadre OS

Detekcia uviaznutia v OS Linux

- Systémové volanie **`futex()`** môže vrátiť EDEADLK
  - pri rekurzívnom volaní, teda ak volajúci už má zamknutú hodnotu na danej adrese (pri operáciách FUTEX_LOCK_PI, FUTEX_TRYLOCK_PI, FUTEX_CMP_REQUEUE_PI),
  - ak jadro zistí uviaznutie pri operácii FUTEX_CMP_REQUEUE_PI.
- Funkcia **`pthread_mutex_lock()`** môže vrátiť EDEADLK
  - Pri rekurzívnom volaní pre zámok ktorý nebol inicializovaný ako rekurzívny, čiže keď volajúci už daný zámok má zamknutý.
  - Rekurzívne zámky je možné zamknúť opakovane a rovnaký počet krát odomknúť.
- Operačný systém si pre každý proces udržuje zoznam zámkov na ktoré čaká (zoznam je položkou PCB/task_struct).
- Napríklad:

```
struct rt_mutex_waiter      *pi_blocked_on;
...
struct mutex_waiter         *blocked_on;
```

Detekcia uviaznutia pre zámky na súboroch

- Zámky na súboroch (*POSIX locks*) je možné nastaviť systémovým volaním `fcntl()`.
- Operácia F_SETLK umožňuje procesu uzamknúť na čítanie alebo zápis (*l_type*) istý rozsah súboru (*l_whence*, *l_start*, *l_len*).
  - Toto volanie je neblokujúce a v prípade, že zámok nie je možné získať (napríklad ak daný rozsah je už zamknutý iným procesom), vráti chybový kód.
- Operácia F_SETLKW bude čakať, až kým nebude možné zámok získať.
  - V prípade, že jadro zistí kruhové čakanie, systémové volanie `fcntl()` pre jeden zo zablokovaných procesov vráti EDEADLK.
  - Napríklad:
    - Proces A zamkne na zápis bajt 100 a proces B bajt 200. Ak sa potom A pokúsi zamknúť cez F_SETLKW bajt 200 a B bajt 100, môže nastať uviaznutie. Jeden z nich dostane EDEADLK.
  - Detekcia nie je dokonalá. Môže sa vyskytnúť aj nedetegované uviaznutie, aj falošne detegované uviaznutie.
  - Cyklus zablokovaných procesov je obmedzený na dĺžku 10.

To je zatiaľ všetko