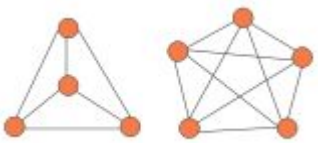
Státnice - 36. PRL

Sdílená paměť

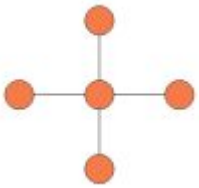
- Všechny procesory mají přístup do celého paměťového prostoru.
- Řešení současného přístupu k jedné buňce:
  - EREW - Exclusive Read, Exclusive Write (velmi omezující)
  - CREW - Concurrent Read, Exclusive Write (časté, jednoduché)
  - ERCW - Exclusive Read, Concurrent Write (nedává smysl)
  - CRCW - Concurrent Read, Concurrent Write (složité)
- Úrovně
  - Multitasking
    - 1CPU přepíná kontext (virtuální procesor), paměť je sdílená, předávání zpráv simulováno SW
  - Systém se sdílenou pamětí
    - CPU mají svou cache, zbytek na sběrnici (boj), předávání zpráv může být v HW nebo simulace SW
  - Virtuální sdílená paměť
    - CPU má svou paměť, ale je virtuálně spojena v simulovanou sdílenou, opět HW/SW simulované zasílání zpráv
  - Systém s předáváním zpráv
    - CPU vázány volně (např. počítačová síť), sdílená paměť simulovaná SW
- Předávání zpráv
  - Každý procesor má vlastní adresový prostor
  - Také každý procesor má vlastní fyzickou paměť, přístup jinam komunikací

Základní typy topologií paralelních architektur

- Statické propojovací sítě
  - všechny uzly jsou procesory
  - všechny hrany jsou komunikační kanály
  - pro architektury bez sdílené paměti
  - vlastnosti:
    - Průměr (diametr): nejdelší délka nejkratších cest mezi všemi dvojicemi uzlů
    - Konektivita: minimální počet hran, které je nutné odstranit pro rozdělení na dvě části
    - Šířka bisekce: minimální počet hran, které spojují dvě přibližně stejně velké části sítě (určuje, zda v síti nevzniká úzké místo - tzv. bottleneck)
  - typická statická propojení
    - Úplné propojení
      - Průměr: 1
      - Konektivita: p-1
      - Šířka bisekce: $p^2 / 4$

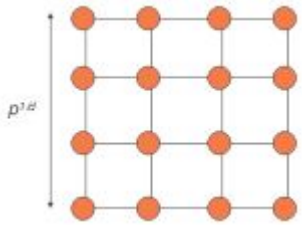


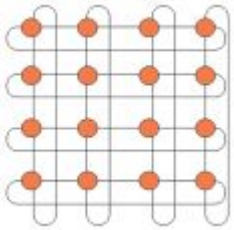
    - Hvězda
      - pro úlohy, které lze rozdělit na samostatné části
      - Průměr: 2
      - Konektivita: 1
      - Šířka bisekce: (p-1)/2



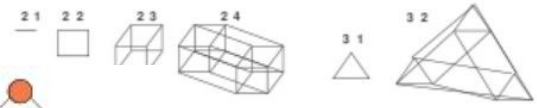
- Lineární pole
  - data proudí přes všechny procesory
  - Průměr: 1
  - Konektivita: 1
  - Šířka bisekce: 1

- D-rozměrná mřížka
  - d-rozměrná mřížka šířky p
  - Průměr: $dp^{1/d}$
  - Konektivita: d
  - Šířka bisekce: $2p^{(1-1/d)}$

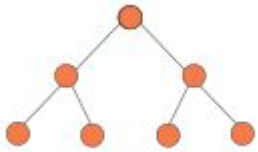




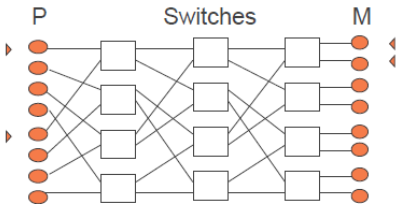
- K-ární d-rozměrná kostka
  - Průměr: d(k/2)
  - Konektivita: 2d
  - Šířka bisekce: $2k^{d-1}$



- D-ární strom
  - Průměr: $2\log_d((p+1)/2)$
  - Konektivita: 1
  - Šířka bisekce: 1



- Dynamické propojovací sítě
  - uzly jsou procesory, paměťové moduly nebo přepínače
  - dynamické přepojování propojení - změna topologie za běhu
  - často implementují sdílenou paměť
  - Křížový přepínač (crossbar)
    - v jednom okamžiku propojení p prvků
    - neblokující
    - Průměr: 1
    - Konektivita: 1
    - Šířka bisekce: p
  - Sběrnice
    - propojení pouze 2 prvků v jednom okamžiku
    - blokující
  - Víceúrovňové sítě
    - spojují p procesorů s p paměťovými moduly pomocí p.log(p) přepínačů.
    - pokud různé procesory přistupují k různým pamětem může dojít k souboji o přepínače

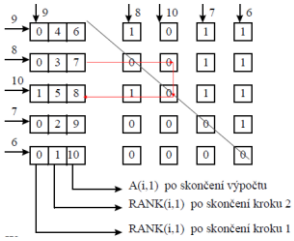


Distribuované a paralelní algoritmy

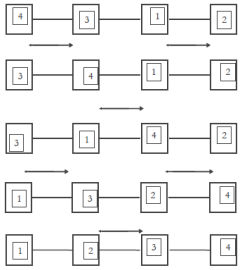
- počet procesorů p je odvozen on délky vstupu n. p(n) = {1, c, log(n), n, n.log(n), n^2, . . . , n^r, r^n}
- čas výpočtu t je také odvozen od n a je udáván v jednotkách (krocích)
- cena algoritmu c(n) = p(n).t(n)
  - algoritmus s optimální cenou je stejně drahý jako sekvenční algoritmus (jde o cenu, ne rychlost) $c_{opt}(n) = t_{seq}(n)$
- zrychlení paralelizací je dáno vztahem $t_{seq}(n)/t(n)$, efektivnost pak $t_{seq}(n)/c(n)$, nastavení je závislé na případu použití
- složitostí většinou rozumíme počet procesorů
  - při výpočtu závislosti na délce vstupu je nejzajímavější nejhorší případ, takže pokud jedna část algoritmu vyžaduje p(n) procesorů a druhá $p(n^2)$ procesorů, výsledná složitost je $p(n^2)$

Algoritmy řazení

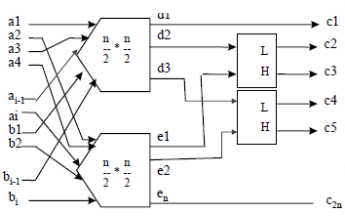
- máme posloupnost prvků X={x1,...,xn} s n prvky a lineární uspořádání >
- cílem je vytvořit z prvků x novou posloupnost Y={y1,...,yn} kde platí $y_i < y_{i+1}$, i=1,...,N-1
- v X nejsou žádné dva prvky rovny
- optimální sekvenční algorimus (platí pro řadící algoritmy založené na porovnávání prvku)
  - p(n) = 1
  - t(n) = O(n.log n)
  - c(n) = O(n.log n)
- Enumeration sort



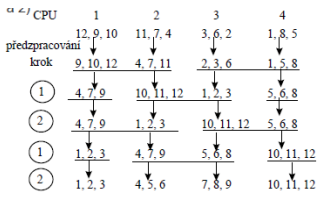
  - Princip: výsledná pozice prvku je dána počtem prvků, které jsou menší
  - Topologie: mřížka n krát n, řádky a sloupce jsou binární stromy v poli
  - Procesory: registry A,B,RANK; do A,B zápis prvku, RANK inkrementace; možnost poslat registr synům
  - Algoritmus: pomocí jedné řady se prvky porovnají (a přitom se mění RANK); správná pozice je RANK; nakonec se prvky přesunou stromem
  - Složitost: t(n) = O(log(n)) – nejrychlejší paralelní řešení; $c(n) = O(n^2.\log(n))$ – není optimální
- Odd-even transposition sort



  - Princip: paralelní bubble-sort, porovnávají se jen sousedé a mohou se přehodit
  - Topologie: lineární pole n procesorů
  - Procesory: obsahují jediný registr s hodnotou prvku
  - Algoritmus: na počátku se pole naplní posloupností; v lichém kroku pracují liché procesory, v sudém sudé; porovná se s následníkem a případně prohodí hodnoty; algoritmus končí po n krocích (lze urychlit testem na prohození)
  - Složitost: t(n) = O(n) – nejrychlejší řešení pro lineární topologii; $c(n) = O(n^2)$ – není ideální
- Odd-even merge sort



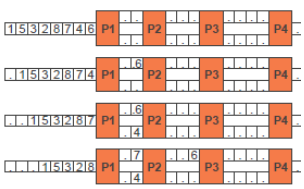
  - Princip: síť složená ze speciálních procesorů
  - Topologie: procesory propojeny tak, aby složením jednotlivých porovnání byla seřazená posloupnost
  - Procesory: 2 vstupy a 2 výstupy, porovná vstupy a dá na výstupy high a low
  - Algoritmus: spočívá v zapojení sítě, kaskáda 1 × 1, 2 × 2, 4 × 4, . . .
  - Složitost: $t(n) = O(\log^2(n))$; $c(n) = O(n.\log^4(n))$ – není optimální
- Merge-splitting sort



  - Princip: varianta odd-even sortu, každý procesor řadí krátkou posloupnost
  - Topologie: lineární pole procesorů – p(n) < n
  - Procesory: obsahuje m prvků, které umí seřadit optimálním sekvenčním algoritmem

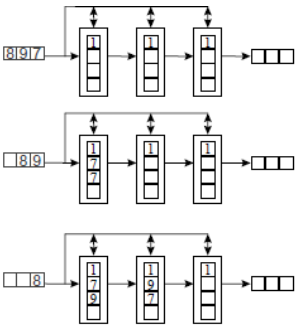
- Algoritmus: místo porovnání sousedů se provede spojení posloupností (O(n)) a pak rozdělení na půl
- Složitost: c(n) = O(n.log(n)) + O(n.p), optimální pro p =< log(n)

- Pipeline merge sort

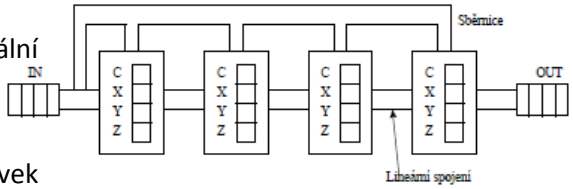


- Princip: rozděleno na několik kroků, první spojuje posloupnosti délky 1, pak 2, atd.
- Topologie: linární pole procesorů – p(n) = log(n) + 1
- Procesory: umí spojovat dvě seřazené posloupnosti O(n)
- Algoritmus: ze vstupní posloupnosti se vezme první prvek a dá jej do jedné posloupnosti, druhý do druhé; další vybere vždy největší prvek a první dva dává do první posloupnosti, druhé dva do druhé; třetí krok také bere největší, ale střídá posloupnosti po čtyřech, atd.
- Složitost: t(n) = O(n); c(n) = O(n).O(log(n) + 1) = O(n.log(n)) – optimální

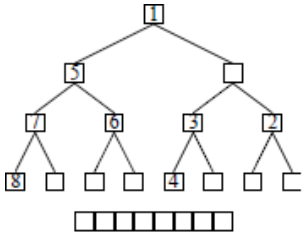
- Enumeration sort 2



- Princip: porovnání se všemi prvky a počet menších určuje pořadí
- Topologie: lineární pole n procesorů a sběrnice, která může přenést jednu hodnotu
- Procesory: registr C (počet menších - kolikrát byl Y =< X), X (prvek na dané pozici), Y (prvek posloupnosti, který se porovnává) a Z (výsledná pozice)
- Algoritmus: C se nastaví na 1; vstupní prvek xi se sběrnicí vloží na Xi a lineárním spojením do Y1; všechny Y se posunou doprava; každý procesor porovná X a Y a pokud X>Y => C++; po vyčepání vstupu posílají procesory zleva hodnoty X sběznicí na Z procesoru určeného hodnotou C (Y se stále posouvá)
- Složitost: t(n) = n; c(n) = $O(n^2)$ – není optimální

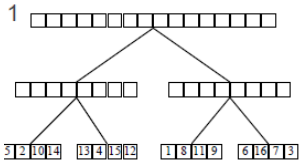


- Minimum extraction sort



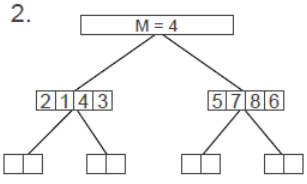
- Princip: stromem odebírá vždy nejmenší prvek
- Topologie: strom s n listy, log(n) + 1 úrovněmi a 2n − 1 procesory
- Procesory: listový procesor obsahuje prvek posloupnosti, nelistové prvky umí porovnat syny
- Algoritmus: naplní se listy; v každém kroku otec vybere menší hodnotu; jakmile je v kořenu hodnota, je to první prvek seřazené posloupnosti
- Složitost: t(n) = O(n); c(n) = $O(n^2)$ – není optimální

- Bucket sort



- Princip: stromem spojené procesory, které řadí menší posloupnosti a pak spojení
- Topologie: strom s m listy, kde n = 2^m
- Procesory: listové procesory řadí krátkou posloupnost, ostatní spojují syny – O(n)
- Složitost: t(n) = O(n); c(n) = O(n.log(n)) – optimální

- Median finding and spliting



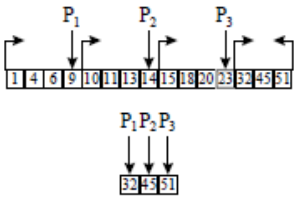
- Princip: dělí posloupnost mediánem až na dvojice, které porovná
- Topologie: strom s m listy, kde n = 2^m
- Procesory: listové procesory porovnají dvojici, ostatní vyberou medián a rozdělí posloupnost – O(n)
- Algoritmus: je jasný, pro výběr mediánu je potřeba optimální, např. select
- Složitost: t(n) = O(n); c(n) = O(n.log(n)) – optimální

Select (medián)

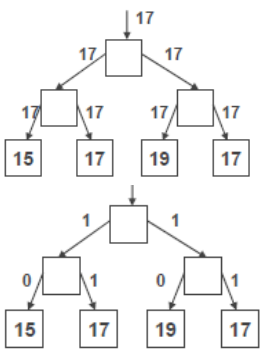
- Sequential select
  - Princip: hledá k-tý nejmenší prvek v posloupnosti S; je-li k=|S|/2, jde o medián
- Parallel select
  - Princip: k-tý nejmenší prvek v posloupnosti S; EREW PRAM s N procesory P1..Pn; používá sdílené pole M o N prvcích
  - Složitost: t(n) = O(n/N) pro n > 4, N < n/log n; p(n) = N; c(n) = t(n).p(n) = O(n) - optimální
- Parallel splitting
  - Krok 4 algoritmu Parallel select
  - Princip: Je dána posloupnost S a číslo m; Mají se vytvořit tři posloupnosti:
    - L = {si ϵ S: si < m}
    - E = {si ϵ S: si = m}
    - G = {si ϵ S: si > m}
  - Složitost sekvenčního algoritmu je O(n)
  - Paralelní řešení - máme N procesorů, které si sekvenci S rozdělí na podposloupnosti Si o délce n/N
  - Složitost: t(n) = O(log N + n/N) = O(n/N) pro dostatečně malé N; Cena c(n) = O(n) - optimální

Algoritmy vyhledávání

- Vyhledávání zjišťuje přítomnost zadaného prvku v posloupnosti a případně i jeho pozici
- Optimální složitost podle sekvenčního algoritmu je O(n) pro neseřazenou posloupnost – sekvenční vyhledávání; O(log(n)) pro seřazenou posloupnost – binární vyhledávání
- N-ary search



  - Seřazeno: ano
  - Princip: paralelní analogie k binárnímu hledání, zjišťuje se polovina, ve které prvek je
  - Topologie: lineární pole m procesorů, kde m < n; CREW (hledaný prvek)
  - Procesory: porovnávají prvek na svém místě s hledaným, registr, který říká, na které straně pokračovat
  - Algoritmus: v každé iteraci se nastaví registry, v úseku, kde na obou stranách je hodnota odlišná se hledá v další iteraci
  - Složitost: t(n) = $O(\log_{m+1}(n + 1))$; c(n) = $O(m.\log_{m+1}(n + 1))$ – není optimální
- Unsorted search
  - Seřazeno: ne
  - Princip: paralelně volaný sekvenční algoritmus
  - Topologie: lineární pole m procesorů, kde m < n
  - Procesory: sekvenčně hledají prvek X s přidělenou posloupností
  - Algoritmus: procesory načtou prvek do registru a provedou hledání, mohou nastavit flag nalezení
  - Složitost:
    - EREW – c(n) = O(m.log(m) + n)
    - CREW – c(n) = O(n) (pokud zapisuje pouze jediný procesor)
- Tree search



  - Seřazeno: ne
  - Princip: listy porovnají a výsledek se propaguje stromem
  - Topologie: strom s 2n − 1 procesory
  - Procesory: listy umějí porovnat, otec logický OR výsledků synů
  - Algoritmus: kořen načte hledaný prvek a stromem propaguje; listy obsahují prvky posloupnosti a porovnají; otec udělá OR
  - Složitost: t(n) = O(log(n)); c(n) = O(n.log(n)) – není optimální

Interakce mezi procesy

- kooperace: spolupráce na řešení úkolu, je potřeba synchronizace (čtenáři - písaři)
- soupeření: o omezené zdroje, je potřeba výlučný přístup (producent - konzument)
- požadavky
  - jen jeden proces může být v KS a jen omezený čas
  - pokud není nikdo v KS a někdo do ní chce, musí se tam dostat bez prodlení
- problémy: jelikož procesy běží paralelně, není jasné pořadí vykonávání instrukcí obou programů
- Řešení vzájemného vyloučení
  - Hardwarové
    - atomické instrukce implementované hardwarově (Test-and-set, Swap)
    - Test-and-set solution

```
Test-and-set solution
shared var lock = 0;
repeat
    while testAndSet(&lock) do skip;
    critical section
    lock = 0;
    remainder section
until false;
```

    - Swap

```
Atomic Swap Solution
shared var lock = 0;
repeat
    key := 1;
    repeat
        swap (&lock, &key);
    until key=0;
    critical section
    lock := 0;
    remainder section
until false;
```

  - V operačním systému
    - Semafory
      - slouží k určení výlučného přístupu k datům
      - struktura obsahující počet možných procesů v kritické zóně, a frontu čekajících procesů (větší než 0)
      - použití semaforu pro synchronizaci: fronta je nastavená na 0
      - kritická sekce je mezi instrukcemi P a V
      - dvě atomické operace:

P(S):	V(S):

S.count--;
if (S.count<0) {
block this process
place this process in S.queue
}
	S.count++;
	if (S.count<=0) {
	remove a process P from S.queue
	place this process P on ready list
	}

    - Monitory
      - eqvivalent semaforu (lze ho i implementovat pomocí semaforů), ale jednodušší na ovládání

- zajišťuje exkluzivitu přístupu k datům
    - lokální proměnné jsou přístupné pouze procedurám monitoru
    - v jednu chvíli může monitor využívat pouze jeden proces
    - monitor uzamkne sdílená data, jakmile k nim proces přistoupí
- synchronizaci procesů zajišťuje programátor pomocí stavových proměnných
- mohou být nastaveny pouze dvěma operacemi
    - wait(a): blokuje provedení volajících procesů
    - signal(a): návrat k provedení nějakého blokovaného procesu

- Softwarové (bez podpory OS nebo programovacího jazyku)
    - předpokládáme
        - čtení/zápis jedné hodnoty je atomické (nemusí platit při používání stránkování)
        - současné čtení/zápis budou provedeny v náhodném pořadí za sebou
    - Dekkrův algorimus (Dekker's algorithm)

```
shared var flag: array [0..1] of boolean; //flagy žádosti o vstup do kritické sekce
 turn: 0..1; // označení aktuálního kola

 repeat
  flag[i] := true;          // proces žádá o vstup do kritické sekce (nastaví flag žádosti)
   while flag[j] do         // proces ověřuje dokud o vstup žádá i druhý proces
    if turn=j then          // Pokud je právě kolo druhého procesu ...
     flag[i] := false;      // ... proces se vzdá své žádosti o krit. sekci...
     while turn=j do skip; // ... a čeká dokud neskončí kolo druhého procesu
     flag[i] := true;       // Poté znovu nastaví svoji žádost o krit. sekci
    endif
   endwhile                 // Proces vstupuje do krit. sekce pokud druhý proces flag nenastavil
  <critical section>
  turn := j;                // Proces nastaví kolo na druhý proces
  flag[i] := false;
  <remainder section>
 until false;
```

- Petersonův algoritmus (Peterson's algorithm)

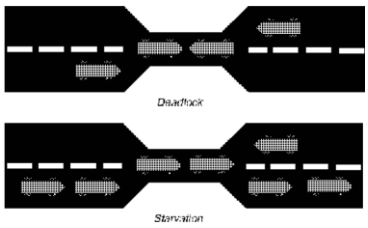
```
shared var flag: array [0..1] of boolean; //flagy žádosti o vstup do kritické sekce
 turn: 0..1; // označení aktuálního kola

 repeat
  flag[i] := true;                  // Proces žádá o krit. sekci
  turn := j;                        // Proces nastaví kolo na druhý proces
  while (flag[j] and turn=j) do skip;   // Čeká dokud je kolo druhého procesu a žádá o krit. sekci
  <critical section>
  flag[i] := false;                 // Po dokončení krit. sekce proces stáhne žádost o krit sekci
  <remainder section>
 until false;
```

- Operátor <await B->S>
    - je původně pouze teoretický operátor, implementovaný ve vyšších programovacích jazycích pro paralelní programování
    - problémová implementace
    - <> - atomičnost; B - Bool expr; S - sekvence příkazů
    - S je atomicky vykonaná jen když je B = true
    - příklad: <await s>0 → s = s - 1>
- Critical Region
    - obalení semaforů ve vyšších jazycích

- vymezení klíčovým slovem region
- deklaruje sdílenou proměnnou shared
- pokud se zanořuje, může dojít ke konfliktu
- Jednoduchá implementace (téměř makro)
- Conditional Critical Region
  - rozšíření CR o podmínku, která zamezí kolizi
  - složitá implementace

Typické problémy paralelismu



- deadlock
- vyhladovění
- Večeřící filozofové (Dining philosophers)
  - 5 filozofů
  - 5 vidliček
  - filozofové přemýšlejí, pak dostanou hlad, nají se a opět přemýšlejí
  - Každý filozof potřebuje 2 vidličky aby se najedl
  - Problém: vyhladovění, deadlock
- Čtenáři a písaři (Readers/writers)
  - sdílená proměnná
  - několik čtenářů, kteří chtějí číst hodnotu
  - jeden nebo několik písařů měnících hodnotu
  - číst může libovolný počet zaráz
  - pokud se zapisuje nikdo jiný nesmí zapisovat ani číst
- Producenti a konzumenti (producer/consumer problem)
  - producenti produkují data
  - konzumenti produkovaná data přijímají a spotřebovávají
  - vyžaduje buffer pro ukládání dat
  - nutné zaručit, že se nebude číst prázdný buffer
  - pokud má buffer omezenou velikost je nutná zabránit přetečení
  - vzájemné vyloučení v přístupu k bufferu

Komunikace

- přenáší data
- sdílená paměť
  - všechny CPU mají přístup do společného paměť. prostoru
  - boj o sběrnici, konflikt při zápisu
  - řešení současného přístupu k jedné buňce paměti:
    - EREW; CREW; ERCW; CRCW
- zasílání zpráv (kanály, volání vzdálených procedur, broadcast)
  - každý CPU má svůj adresový prostor
  - procesory mají vlastní paměť
  - Zasílání zpráv
    - synchronní
      - blokující, čeká na potvrzení, může mít buffer
      - lze simulovat asynchronním kanálem (použití ACK)
    - asynchronní
      - neblokující, neomezený buffer
    - operace:
      - send(channel, msg)
      - receive(channel, msg)

PRAM

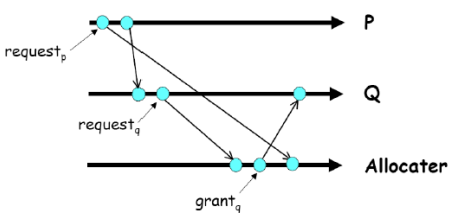
- synchronní model paralelního výpočtu ve kterém procesory komunikují sdílenou pamětí
- výpočet probíhá po krocích: čtení sdílené paměti; lokální operace; zápis do sdílené paměti
- přístupy do paměti: EREW, ERCW, CREW, CRCW
- řešení zápisových konfliktů
  - COMMON: všechny zapisované hodnoty musí být shodné
  - ARBITRARY: zapisované hodnoty mohou být různé, zapíše se libovolná
  - PRORITY: procesory mají prioritu
- broadcast
  - hodnota uložená v paměti má být rozšířena mezi N procesory
  - pro CREW a CRCW řešení v konstantním čase
  - pro EREW je třeba simulovat současné čtení (P1 přečte D a zpřístupní jej P2, P1 a P2 jej zpřístupní P3 a P4...)

Distribuovaný broadcast

- komunikace: známe horní mez zpoždění zprávy; lokální hodiny pro každý proces
- topologie: obecná topologie grafu
- specifikace
  - m = zpráva z množiny možných zpráv
  - každá zpráva obsahuje
    - totožnost odesilatele: sender(m)
    - pořadové číslo (kolikátou zprávu odesilatel poslal): seq(m)
- základní vlastnosti
  - Platnost (validity): pokud správný proces odešle zprávu, všem procesům se zpráva předá
  - Dohoda (Agreement): pokud správný proces předává zprávu, všem procesům se zpráva předá
  - Integrita (celistvost): každý proces přijme zprávu jen jedenkrát
  - FIFO řazení (FIFO order): pokud se odešle n před m, tak žádný proces nesmí poskytnout m před n
  - Náhodné řazení (Casual order): pokud n náhodně předchází m, tak žádný proces nesmí poskytnout m před n
  - Celkové řazení (Total order):
- klasifikace broadcast
  - Spolehlivé = Platnost + Dohoda + Integrita
  - FIFO = Spolehlivé + FIFO order

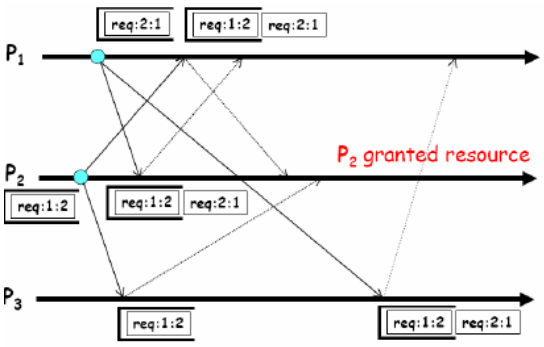
Synchronizace v distribuovaných systémech

- absence globálních a synchronizačních hodin
- vzájemné vyloučení
  - existuje pevný počet procesorů, které sdílí jeden zdroj a ten může v danou chvíli používat pouze jeden procesor
  - centralizované řešení



  - distribuované řešení
    - založené na soupeření (Lamportův algoritmus)
    - založené na předávání pověření (Raymond)

- Lamportův algoritmus
  - předpokládá oboustranné FIFO kanály mezi procesy
  - každý proces udržuje vlastní frontu požadavků
  - používá časových značek k uspořádání požadavků
  - vyžaduje 3(n-1)zpráv
  - fáze: požadavek (request), potvrzení (reply), uvolnění (release)
  - algoritmus:
    - proces P požaduje poslání požadavku do všech procesů zasláním req
    - požadavek je zařazen do fronty uspořádané dle časových značek, poslání zprávy ack
    - ukončení zpracování požadavku, odstranění požadavku z pronty a poslání všem rel
    - přijetí rel od P odstranění procesu P z fronty
    - povolení zpracování požadavku, požadavek je na prvním místě ve frontě a je povolen ostatními procesy



- Raymond
  - procesy uspořádány do stromu
  - pracuje na bázi předávání pověření
  - udržuje frontu požadavků na pověření od procesů nižších úrovní
  - požaduje-li vstup do KS a fronta je prázdná, posílá požadavek nadřazenému a řadí se do fronty, požaduje-li podřízený proces vstup do KS, předá mu pověření nebo jej zařadí do fronty, obdrží-li pověření předá ho prvnímu ve frontě