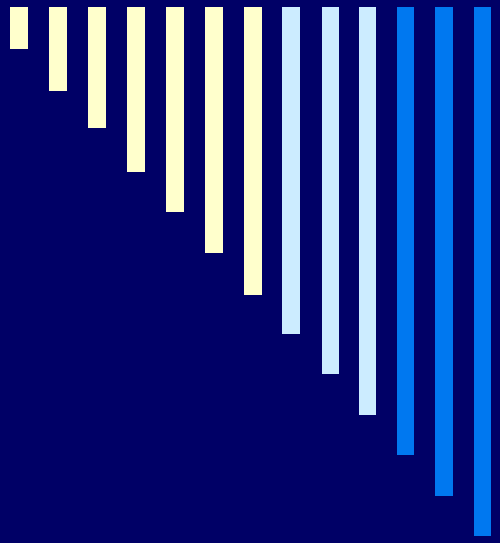
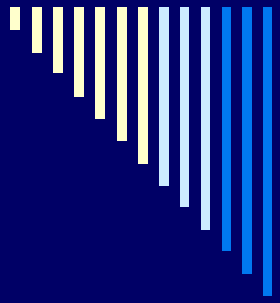


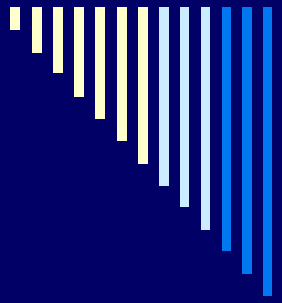
05. Meziprocesová komunikace – zprávy, RPC

ZOS 2006



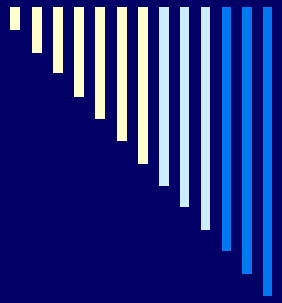
Meziprocesová komunikace

- Předávání zpráv
- Primitiva send, receive
- Mailbox, port
- RPC
- Ekvivalence semaforů, zpráv, …
- Bariéra, problém večeřících filozofů



Problém sdílené paměti

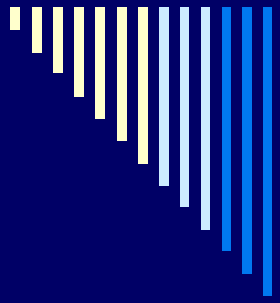
- Uvedené mechanismy
  - umístění objektu ve sdílené paměti
- Někdy není **vhodné**
  - Bezpečnost – globální data přístupná kterémukoliv procesu bez ohledu na semafor
- Někdy není **možné**
  - Procesy běží na různých strojích, komunikují spolu po síti
- Řešení – předávaní zpráv



Předávání zpráv – send, receive

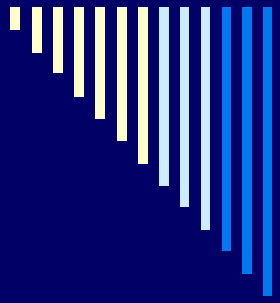
- Zavedeme 2 primitiva
- send (adresát, zpráva) - odeslání zprávy
- receive(odesilatel, zpráva) - příjem zprávy

- Send
  - Zpráva (lib. datový objekt) bude zaslána adresátovi
- Receive
  - Příjem zprávy od určeného odesilatele
  - Přijatá zpráva se uloží do proměnné „zpráva“



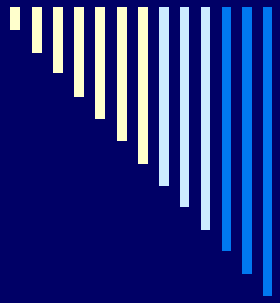
Vlastnosti

- synchronizace (blokující – neblokující)
- unicast, multicast, broadcast
- přímá komunikace x nepřímá komunikace
- délka fronty zpráv
- pevná vs. proměnná délka zprávy



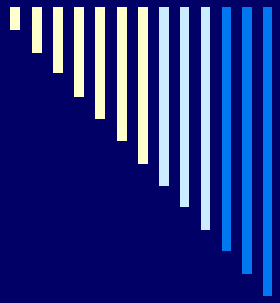
Synchronizace

- blokující (synchronní)
- neblokující (asynchronní)
  - čeká send na převzetí zprávy příjemcem?
  - co když při receive není žádná zpráva?
- většinou send neblokující, receive blokující



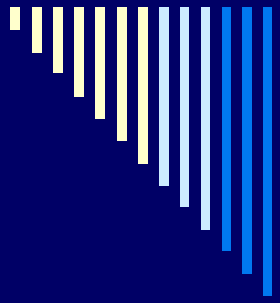
Send - receive

- blokující send
  - čeká na převzetí správy příjemcem
- neblokující send
  - vrací se ihned po odeslání zprávy
  - většina systémů
- blokující receive
  - není-li ve frontě žádná zpráva, zablokuje se
  - většina systémů
- neblokující receive
  - není-li zpráva, vrací chybu



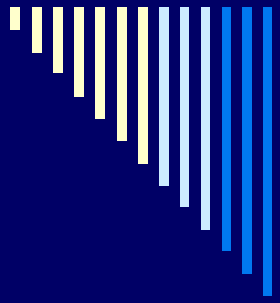
Receive s omezeným čekáním

- receive (odesilatel, zprava, t)
  - čeká na příchod zprávy dobu t
  - pokud zpráva nepřijde, vrací se volání s chybou



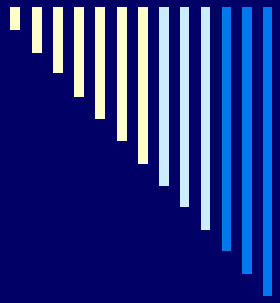
Adresování

- send 1 příjemce nebo skupina?
- receive 1 odesílatel nebo různí?



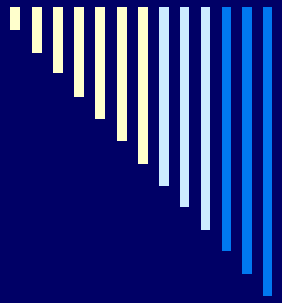
Skupinové a všesměrové adresování

- skupinové adresování (multicast)
  - zprávu pošleme skupině procesů
  - zprávu obdrží každý proces ve skupině
- všesměrové vysílání (broadcast)
  - zprávu posíláme “všem” procesům
  - tj. více nespecifikovaným příjemcům
- pozn.: další varianta - anycast (IPv6)



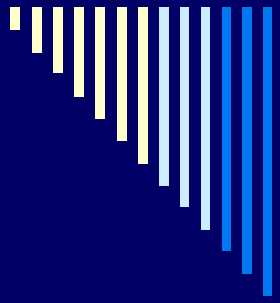
Poznámky – anycast (IPv6)

- spíše se týká síťové komunikace …
- nearest server selection
  na základě směrovací cesty
- abstrakce služby
  unikátní identifikátory služeb (DNS, HTTP proxy)
- spolehlivost



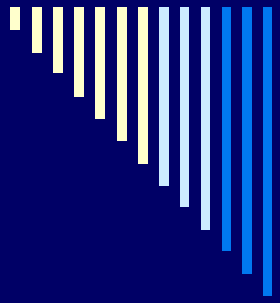
Poznámky

- Většina systémů umožnuje odeslání zprávy skupině procesů a příjem zprávy od kteréhokoliv procesu



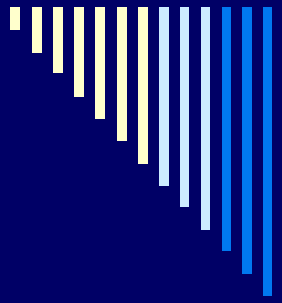
Další otázky

- vlastnosti fronty zpráv
  - kolik jich může obsahovat, je omezená?
- pokus odeslat zprávu a fronta zpráv plná?
  - většinou odesilatel pozastaven
- v jakém pořadí jsou zprávy doručeny?
  - většinou v pořadí FIFO
- jaké je zpoždění mezi odesláním zprávy a možností zprávu přijmout?
- jaké mohou v systému nastat chyby, např. mohou se zprávy ztrácet?



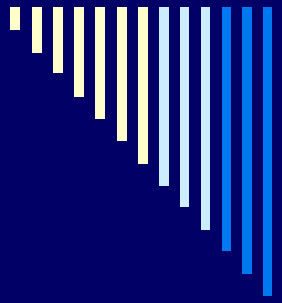
Délka fronty zpráv (buffering)

- nulová délka
  žádná zpráva nemůže čekat
  odesílatel se zablokuje – “randezvous”
- omezená kapacita
  blokování při dosažení kapacity
- neomezená kapacita
  odesilatel se nikdy nezablokuje



Poznámka

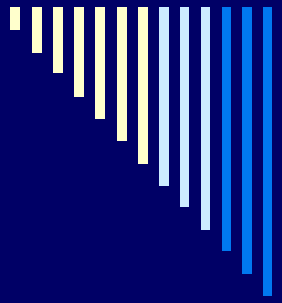
- Volbu konkrétního chování primitiv send a receive provádějí návrháři systému
- Některé systémy nabízejí několik alternativních primitiv send a recive s různým chováním



Terminologická poznámka

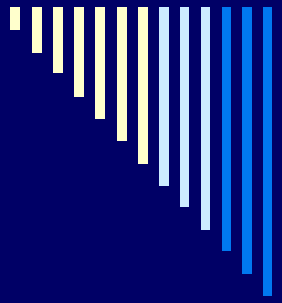
- neblokující send

- v některých systémech send, který se vrací ihned – ještě před odesláním zprávy
- odeslání se provádí paralelně s další činností procesu
- používá se zřídka



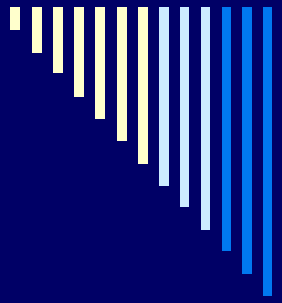
Předpoklady pro další text

- send je neblokující, receive blokující
- receive – umožnuje příjem od libovolného adresáta – receive(ANY,zpráva)
- fronta zpráv – dostatečně velká na všechny potřebné zprávy
- zprávy doručeny v pořadí FIFO a neztrácejí se

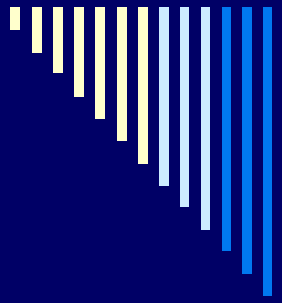


Producent – konzument pomocí zpráv

- symetrický problém
- producent generuje plné položky
  - pro využití konzumentem
- konzument generuje prázdné položky
  - pro využití producentem



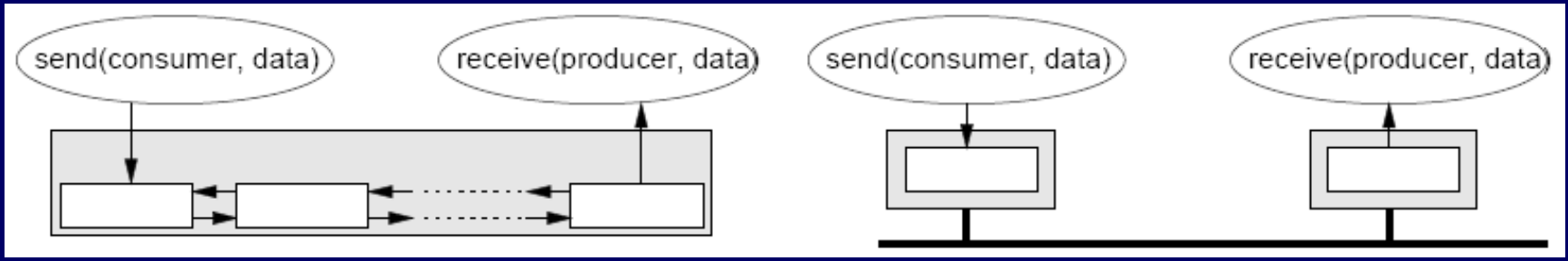
```
cobegin
    while true do                    { producent }
    begin
        produkuj záznam;
        receive(konzument, m);
                    // čeká na prázdnou položku
        m := záznam;
                    // vytvoří zprávu
        send(konzument, m);
                    // pošle položku konzumentovi
    end {while}
||
```

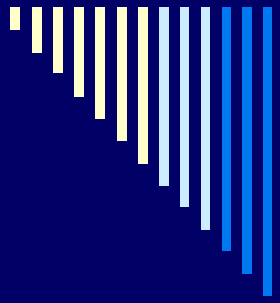


```
for i:=1 to N do                    { inicializace }
    send(producent, e);
          // pošleme N prázdných položek
while true do                       { konzument }
begin
    receive(producent, m);
           // přijme zprávu obsahující data
    záznam := m;
    send(producent, e);
          // prázdnou položku pošleme zpět
    zpracuj záznam;
end {while}
coend.
```

Komunikující procesy

procesy nemusejí být na stejném stroji, ale mohou komunikovat po síti



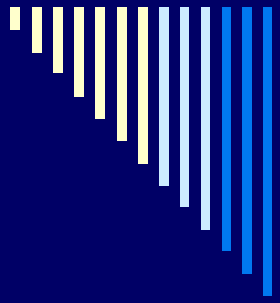


Problém určení adresáta

- dosud – zprávy posíláme procesům
- jak určit adresáta, pojmenovat procesy

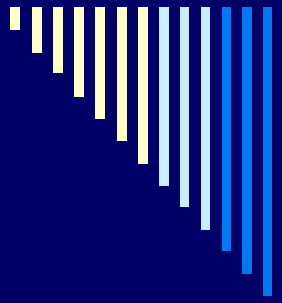
- procesy nejsou trvalé entity
  - v systému vznikají a zanikají
- více instancí stejného programu

- řešení – adresujeme frontu zpráv
  - nepřímá komunikace



Adresování fronty zpráv

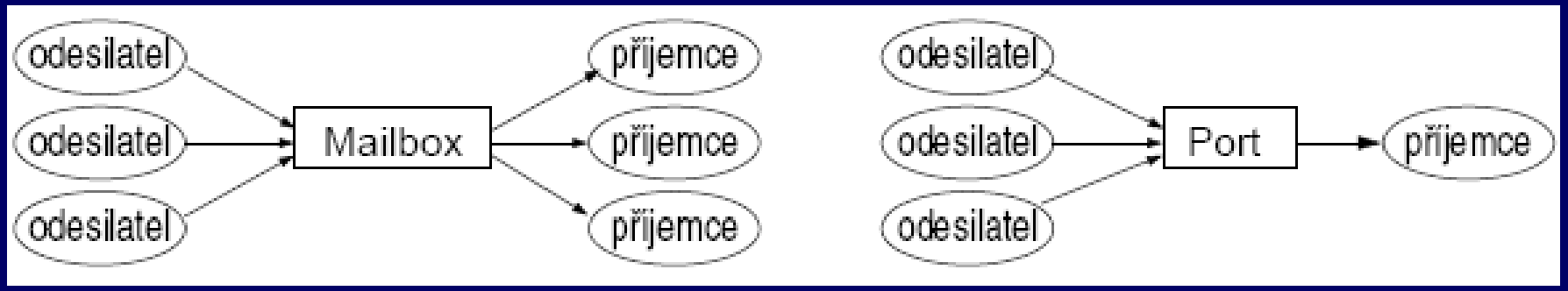
- proces pošle zprávu
  - zpráva se připojí k určené frontě zpráv
- další proces přijme zprávu
  - vyjme zprávu z dané fronty

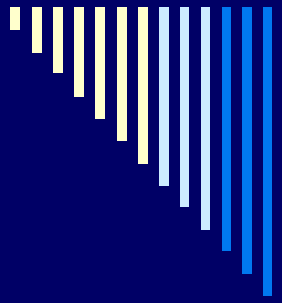


Mailbox, port

- mailbox
  - fronta zpráv využívaná více odesílateli a příjemci
  - obecné schéma
  - operace receive – drahá, zvláště pokud procesy běží na různých strojích
- port
  - omezená forma mailboxu
  - zprávy může vybírat pouze jeden příjemce

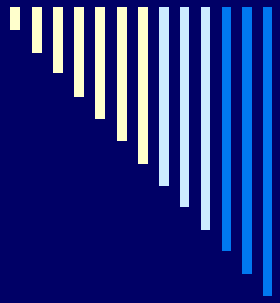
Mailbox, port





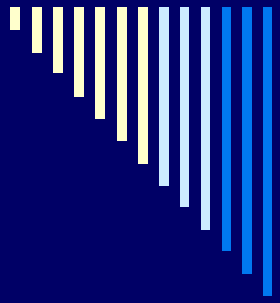
Implementace mechanismu zpráv - otázky

- problémy, které nejsou u semaforů ani monitorů, zvláště při komunikaci po síti
- ztráta zprávy
  - potvrzení o přijetí (acknowledgement)
  - pokud vysílač nedostane potvrzení do nějakého časového okamžiku (timeout), zprávu pošle znovu
- ztráta potvrzení
  - zpráva dojde ok, ztratí se potvrzení
  - číslování zpráv, duplicitní zprávy se ignorují



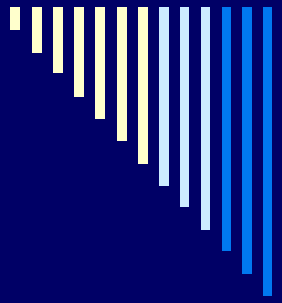
Problém autentizace

- problém autentizace
  - ověřit, že nekomunikují s podvodníkem
  - zprávy je možné šifrovat
  - klíč známý pouze autorizovaným uživatelům (procesům)
  - zašifrovaná zpráva obsahuje redundanci – umožní detekovat změnu zašifrované zprávy
  - Pozn. Symetrické a asymetrické šifrování, podpisy zpráv



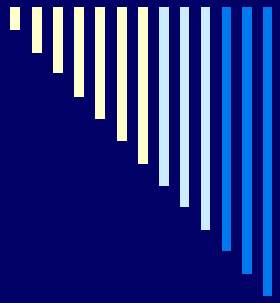
Lokální komunikace

- Na stejném stroji – snížení režie na zprávy
- jednoduchá implementace
  - dvojí kopírování
  - z procesu odesílatele do fronty v jádře
  - z jádra do procesu příjemce
- např. rendezvous
  - eliminuje frontu zpráv
  - send zavolán dříve než receive – odesílatel zablokován
  - vyvolán send i receive – zprávu zkopírovat z odesílatele přímo do příjemce
  - efektivnější, ale méně obecné



Lokální komunikace II.

- např. využití mechanismu virtuální paměti
  - paměť obsahující zprávu je přemapována
    - z procesu odesílatele
    - do procesu příjemce
  - zpráva se nekopíruje

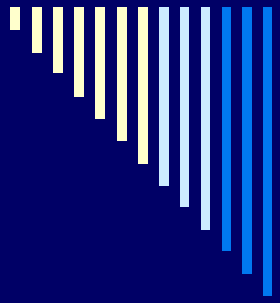


Typické aplikace

- struktura klient – server
  - klient požaduje vykonat práci
  - server práci vykonává
- klienti vyžadují od serveru data
  nebo provedení operace nad daty

- WWW klient x WWW server (Apache, IIS)
- účetní aplikace x databázový systém (Oracle, MySQL)
- klientský OS x souborový server (AFS)

- průběh komunikace

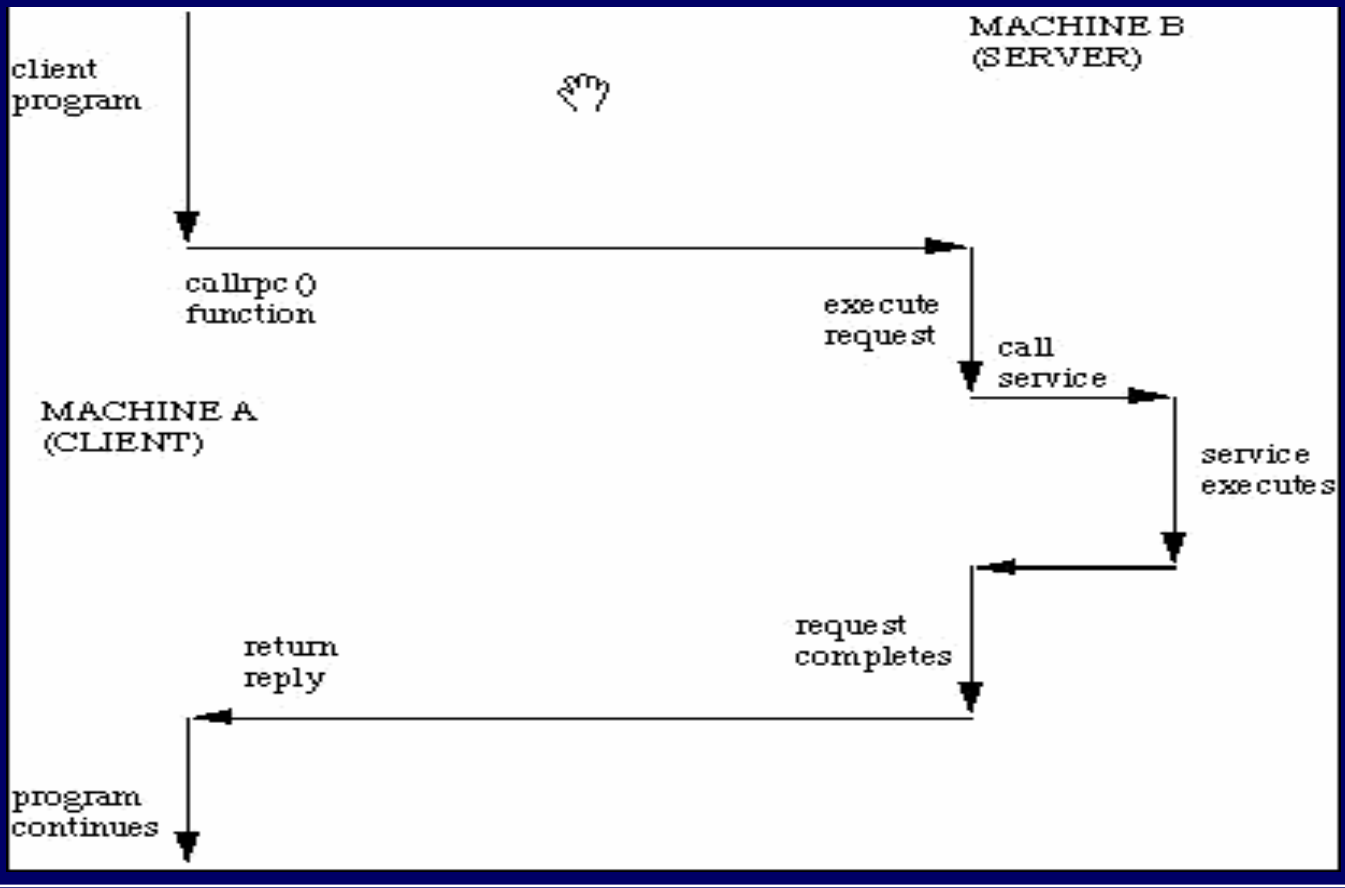


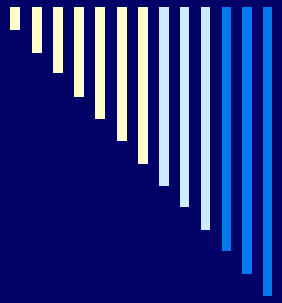
Volání vzdálených procedur (RPC)

- používání send receive – opět nestrukturované
- Birell a Nelson (1984)
- dovolit procesům (klientům) volat procedury umístěné v jiném procesu (serveru)
- mechanismus RPC (Remote Procedure Call)

- variantou RPC je i volání vzdálených metod (Remote Method Invocation, RMI) v OOP, např. Javě
- snaha aby co nejvíce připomínalo lokální volání

RPC

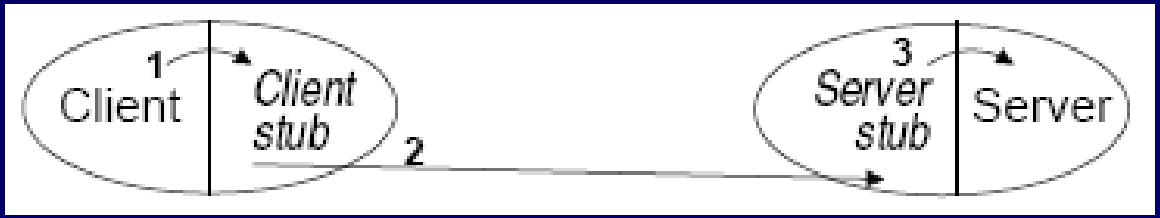

client
program
MACHINE B
(SERVER)
callrpc ()
function
execute
request
call
service
MACHINE A
(CLIENT)
service
executes
request
completes
return
reply
program
continues



spojka klienta, serveru

- klientský program sestaven s knihovní funkcí, tzv. spojka klienta (client stub)
  - reprezentuje vzdálenou proceduru v adresním prostoru klienta
  - stejné jméno, počet a typ argumentů jako vzdálená procedura
- program serveru sestaven se spojkou serveru (server stub)
- spojky zakrývají, že volání není lokální

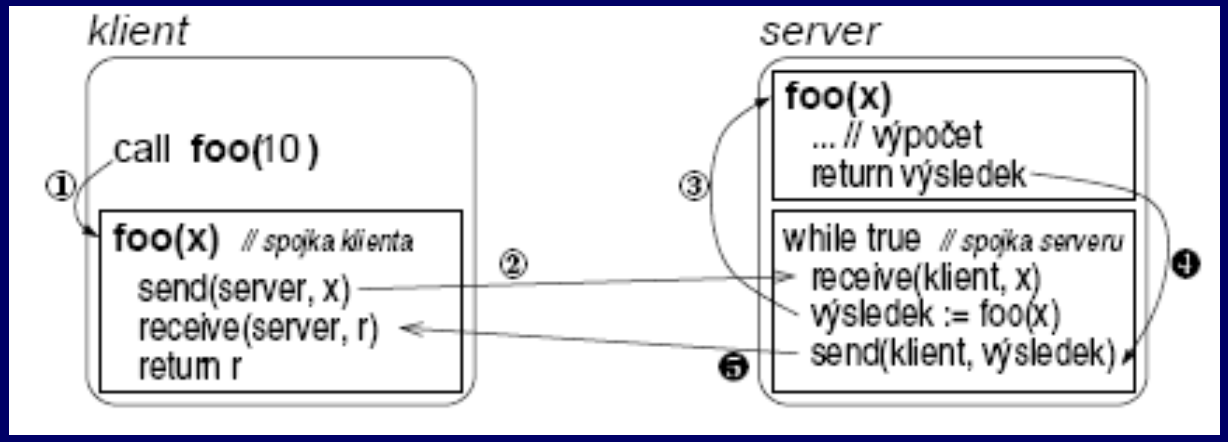
Kroky komunikace



1. Klient zavolá spojku klienta, reprezentující vzdálenou proceduru
2. Spojková procedura argumenty zabalí do zprávy, pošle ji serveru
3. Spojka serveru zprávu přijme, vezme argumenty a zavolá proceduru
4. Procedura se vrátí, návrat. hodnotu pošle spojka serveru zpět klientovi
5. Spojka klienta přijme zprávu obsahující návrat. hodnotu a předá ji volajícímu

Příklad

- volání foo(x: integer): integer



1. Klient volá spojku klienta foo(x) s argumentem x=10.
2. Spojka klienta vytvoří zprávu a pošle jí serveru:

```
procedure foo(x: integer):integer;
begin
send(server, m);   // zpráva obsahuje argument, tj. hodnotu ”10”
```

3. Server přijme zprávu a volá vzdálenou proceduru:

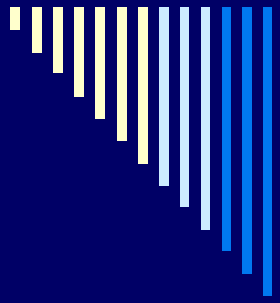
```
receive(klient, x);    // spojka přijme zprávu, tj. hodnotu ”10”
vysledek = foo(x);   // spojka volá fci foo(10)
```

4. Procedura foo(x) provede výpočet a vrátí výsledek.
5. Spojka serveru výsledek zabalí do zprávy a pošle zpět spojce klienta:

```
send(klient, výsledek);
```

6. Spojka klienta výsledek přijme, vrátí ho volajícímu (jako kdyby ho spočetla sama):

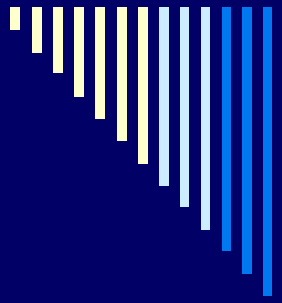
```
receive(server, výsledek);
foo = výsledek;
return;
```



RPC – více procedur

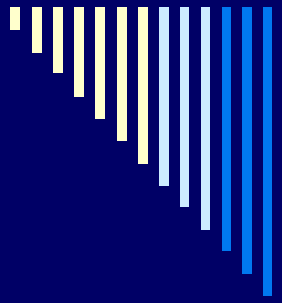
- rozlišeny číslem
- spojka klienta ve zprávě předá kromě parametrů i číslo požadované procedury

```
while true do begin
receive(klient, m); // zpráva obsahující č. procedury a parametry
if (m.číslo_procedury = 1) then výsledek = foo(m.x);
if (m. číslo_procedury = 2) then výsledek = bar(m.x);
...
send(klient, výsledek); // odešli zpět návratovou hodnotu
end
```



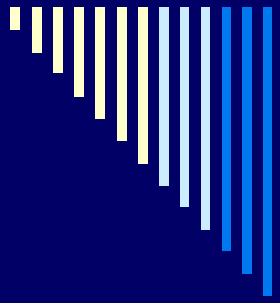
RPC

- dnes nejpoužívanější jazyková konstrukce pro implementaci distribuovaných systémů a programů bez explicitního předávání zpráv
- DCE RPC, Java RMI, CORBA



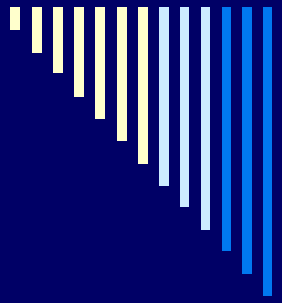
Programování RPC

- Jazyk IDL (Interface Definition Language)
  - Definujeme rozhraní mezi klientem a serverem (datové typy, procedury)
- Kompilátor jazyka IDL
  - Vygeneruje spojky pro klienta i server
- Server sestavíme se spojkou serveru
- Spojka klienta
  - Podoba knihovny
  - Sestavujeme s ní klientské programy



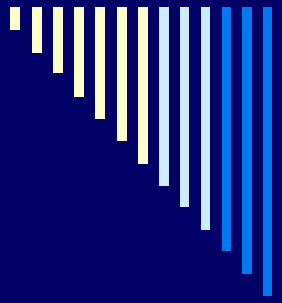
Problémy RPC

- Parametry předávané odkazem
  - Klient a server – různé adresní prostory
  - Odeslání ukazatele nemá smysl
  - Pro jednoduchý datový typ, záznam, pole – trik
    - Spojka klienta pošle odkazovaná data spojce serveru
    - Spojka serveru vytvoří nový odkaz na data atd.
    - Modifikovaná data pošle zpátky na klienta
    - Spojka klienta přepíše původní data
- Globální proměnné
  - Použití není možné x lokálních procedur



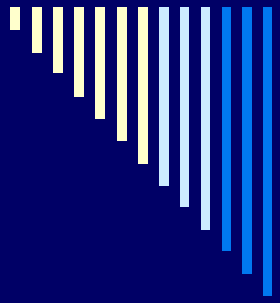
Reprezentace informace

- Společný problém pro předávání zpráv i RPC
- Stroje různé architektury
  - Může se lišit vnitřní reprezentace datových typů
  - Kódování řetězců
    - Udaná délka nebo ukončovací znak
    - Kódování jednotlivých znaků
  - Numerické typy
    - Způsob uložení (little endian, big endian)
    - Velikost (integer 32 nebo 64 bitů)



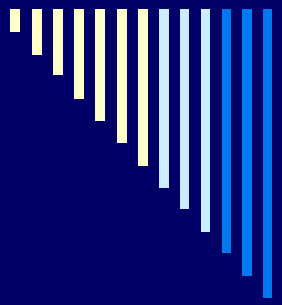
Little & big endian

- Chceme uložit: 4a3b2c1d (32bit integer)
- Big endian
  - Nejvýznamnější byte (MSB) na nejnižší adrese
  - Motorola 68000, SPARC, System/370
  - V paměti od nejnižší adresy: 4a, 3b, 2c, 1d
- Little endian
  - Nejméně významný byte (LSB) na nejnižší adrese
  - Intel x86, DEC VAX
  - V paměti od nejnižší adresy: 1d, 2c, 3b, 4a



Další varianty endians

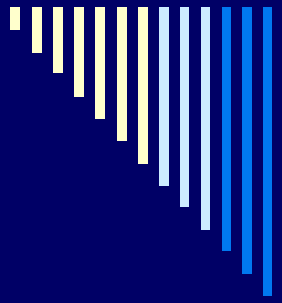
- Bi-endian (bytesexual)
  - Lze nastavit (např. mode bit), jaký formát uložení se bude používat
  - Např. IA-64, defaultně little-endian
- Middle-endian
  - Starší formát, jen poznámka
  - Např. 3b, 4a, 1d, 2c



Otestování sw

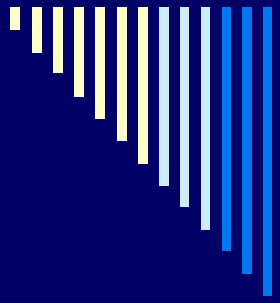
```
#define LITTLE_ENDIAN   0
#define BIG_ENDIAN      1

int machineEndianness() {
    short s = 0x0102;
    char *p = (char *) &s;
    if (p[0] == 0x02)
        return LITTLE_ENDIAN;
    else
        return BIG_ENDIAN; }
```



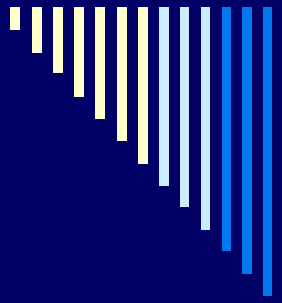
Řešení portability

- Definovat, jak budou data reprezentována při přenosu mezi počítači – síťový formát
  - Před odesláním do síťového formátu
  - Po přijetí do lokálního formátu
- Problém rozdílné velikosti
  - Nová množina numerických typů, stejná velikost na všech podporovaných architekturách
  - Int32_t – integer 32bitů, <stdint.h>, ISO C99



Síťový formát

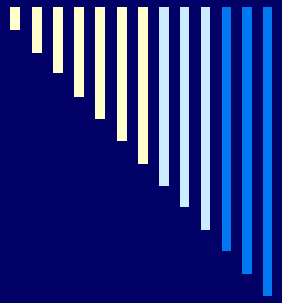
- TCP/IP
  - Network byte order (big endian)
  - Celé číslo – nejvýznamější byte jako první (MSB)
- Konverzní funkce např. <netinet/in.h>
- htonl, htons
- ntohl, ntohs
- (“host to net, net to host, short/long)



Sémantika volání RPC

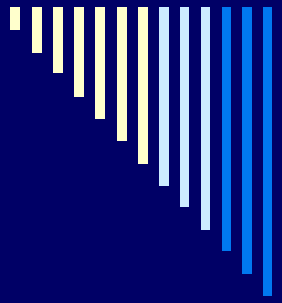
- lokální volání fce – právě jednou

- vzdálené volání
  - chyba při síťovém přenosu (tam, zpět)
  - chyba při zpracování požadavku na serveru
  - klient neví, která z těchto chyb nastala
  - volající havaruje po odeslání zprávy před získáním výsledku



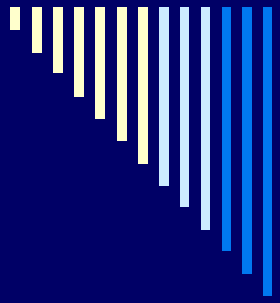
Sémantika volání RPC

- právě jednou
- alespoň 1x
  - opakované volání po timeoutu
  - dle charakteru operace
- nejvýše 1x
  - klient volání neopakuje
  - při timeoutu – chyba, ošetření výjimek



Idempotentní operace

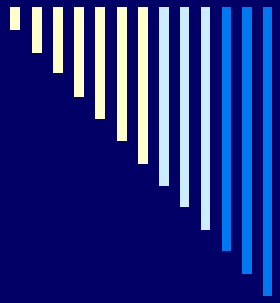
- operace, kterou lze opakovat se stejným efektem, jaký mělo její první provedení
- pro sémantiku alespoň 1x
- x = x + 10 vs. x = 20
- vypinac (zapni), vypinac (vypni) x vypinac (prepni)



Ekvivalenty uvedených primitiv

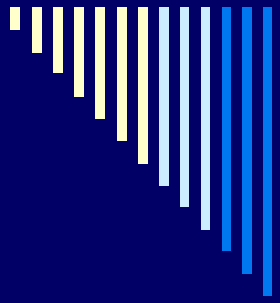
- Implementovat semafory pomocí zpráv
- Zprávy pomocí semaforů, …
- Tj. má obojí stejnou vyjadřovací sílu?

- Zprávy pomocí semaforů
  - Řešení problému producent-konzument
  - Vložení zprávy umístíme do send
  - Vyjmutí zprávy do receive



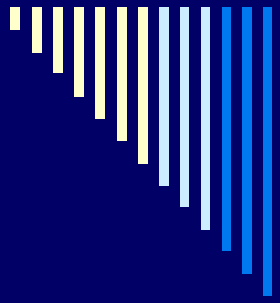
Ekvivalenty

- Semafory pomocí zpráv
- Pomocný synchronizační proces (SynchP)
  - Pro každý semafor udržuje čítač (hodnotu semaforu)
  - A seznam blokovaných procesů
- Operace P a V
  - Jako funkce, které provedou odeslání požadavku
  - Poté čekají na odpověď pomocí receive
- SynchP – v jednom čase jeden požadavek
  - Tím zajištěno vzájemné vyloučení



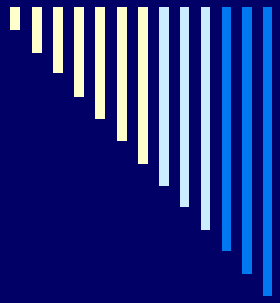
Ekvivalenty

- Pokud SynP obdrží požadavek na operaci P
  - a semafor > 0, odpoví ihned
  - Jinak neodpoví – čímž volajícího zablokuje
- Pokud SyncP obdrží požadavek na operaci V
  - A je blokovaný proces
    - Jednomu blokovanému odpoví, čímž ho vzbudí



Stejná vyjadřovací síla

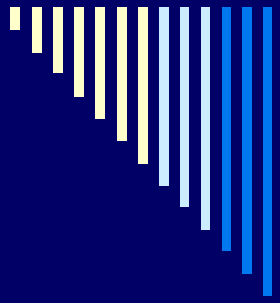
- Lze ukázat, že je možné implementovat
  - Semafory pomocí monitoru
  - Monitory pomocí semaforů
- Všechna dříve uvedená primitiva mají stejnou vyjadřovací sílu
- Platí to i o mutexech? Ano (ale viz dále)



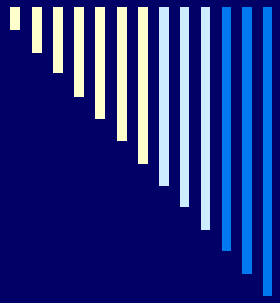
Semafor pomocí mutexů

- Např. Barz (1983)

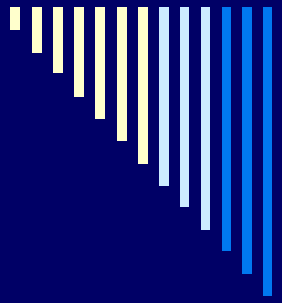
```
type semaphore = record
                        val: integer;
                        m: mutex;        // pro vzájemné vyloučení
                        d: mutex;        // pro blokování (delay)
                 end;
procedure Initsem(var s: semaphore, count: integer);
begin
    s.val:=count;
    s.m :=0;                        // odemčeno
if count=0 then s.d := 1            // zamčeno, někdo musí provést V(s)
    else s.d := 0                   // odemčeno
end;
```



```
procedure P(var s: semaphore);
begin
    mutex_lock(s.d);  // když s.val=0, čekáme
    mutex_lock(s.m); // kritická sekce –
                     // přístup k s.val
s.val := s.val - 1;      // vždy bude platit s.val>=0
if s.val > 0 then
    mutex_unlock(s.d); // další proces do operace P
mutex_unlock(s.m)     // konec přístupu k val
end;
```



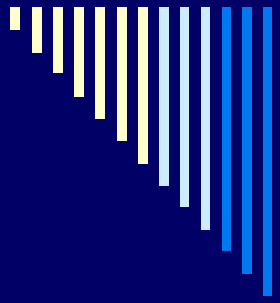
```
procedure V(var s: semaphore);
begin
    mutex_lock(s.m);
    s.val := s.val + 1;
    if s.val = 1 then
          mutex_unlock(s.d)
    mutex_unlock(s.m)
end;
```



Omezení mutexů v některých implementacích

- Z důvodů efektivity někdy omezení mutexů
- Mutex smí odemknout pouze vlákno, které předtím provedlo jeho uzamknutí (POSIX.1)

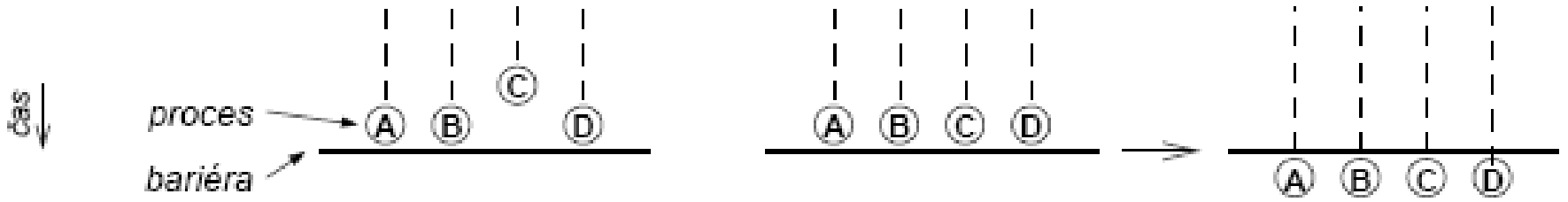
- Nelze použít pro implementaci obecných semaforů
- Slabší než výše uvedená primitiva



Bariéry

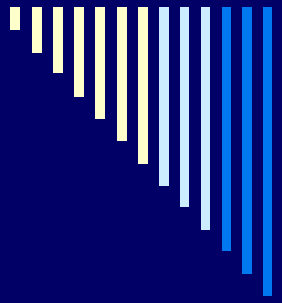
- Synchronizační mechanismus pro skupiny procesů
- Použití ve vědecko-technických výpočtech
- Aplikace – skládá se z fází
  - Žádný proces nesmí do následující fáze dokud všechny procesy nedokončily fázi předchozí
- Na konci každé fáze – synchronizace na bariéře
  - Volajícího pozastaví
  - Dokud všechny procesy také nezavolají barrier
- Všechny procesy opustí bariéru současně

Bariéra



a) procesy přicházejí na bariéru a čekají

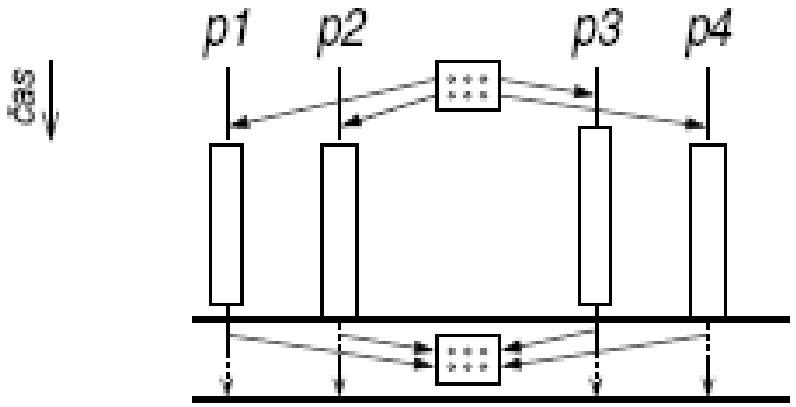
b) po příchodu posledního procesu všechny společně projdou bariérou



Bariéra – iterační výpočty

- Jednotlivé kroky výpočtu
- Matice X(i+1) z matice X(i)
- Každý proces počítá 1 prvek nové matice
- Synchronizace pomocí bariery

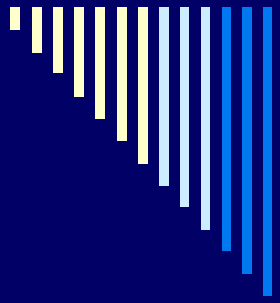
Bariera – iterační výpočty



čtení globálních proměnných

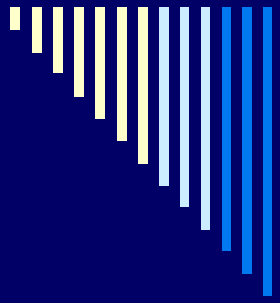
lokální výpočet

synchronizace na bariéře
zápis do globálních proměnných
synchronizace na bariéře



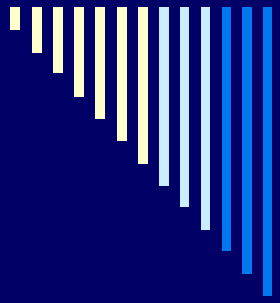
Klasické problémy IPC

- IPC – Interprocess Communication
- Producent- konzument
- Další problémy
  - Večeřící filozofové
  - Spící holič
  - …



Problém večeřících filozofů

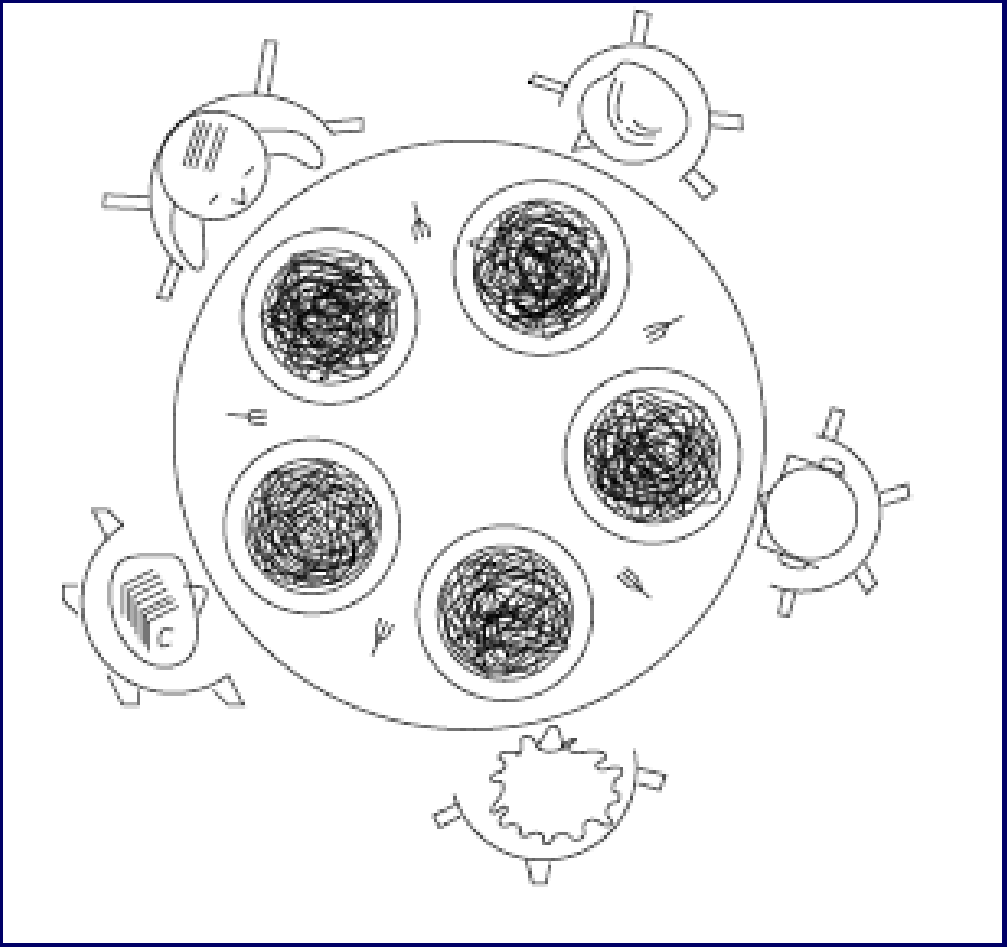
- Dijkstra 1965, dining philosophers
- Model procesů soupeřících o výhradní přístup k omezenému počtu zdrojů
  - Může dojít k zablokování, vyhladovění
- „Test elegance“ nových synchronizačních primitiv

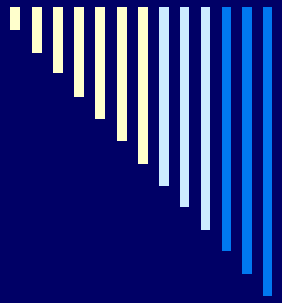


Problém večeřících filozofů

- 5 filozofů sedí kolem kulatého stolu
- Každý filozof má před sebou talíř se špagetami
- Mezi každými dvěma talíři je vidlička
- Filozof potřebuje dvě vidličky, aby mohl jíst
  - Špagety klouzavé ☺

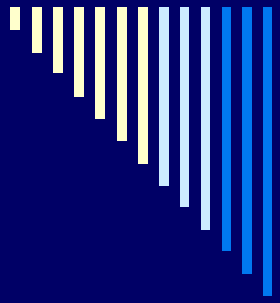
Problém večeřících filozofů





Problém večeřících filozofů

- Život filozofa – jedení a přemýšlení
- Když dostane hlad
  - Pokusí se vzít si dvě vidličky
    - Uspěje – nějakou dobu jí, pak položí vidličky a pokračuje v přemýšlení
- Úkolem
  - Napsat program pro každého filozofa, aby pracoval dle předpokladů a nedošlo k potížím
  - Aby se každý najedl



Problém večeřících filozofů

- Rozbor algoritmu – viz p5rpc.pdf