

DISTRIB. SYSTÉMY

19.2.2013

static mutex_t m = PTHREAD_MUTEX_INITIALIZER

yield()
cancel()

while(1) {

lock(&m)

if(finish)

break;

else

yield();

}

[create
join]

[mutex-lock
mutex-unlock]

[cond-wait
cond-signal
equal]

concurrent programming → programovanie so zdieľanou pamäťou

int c;

EREX → exclusive read exclusive write

CREW → concurrent read exclusive write

ERCW → concurrent read concurrent write

CRCW → toto je jediný možný (správaný) prístup

if(c==0) → toto je tiež čítanie p.čímej
→ taketož inštancie sačne vždy sčítajte mutexom

mutex m;

lock(&m)

{ if(c==0){

unlock(&m)

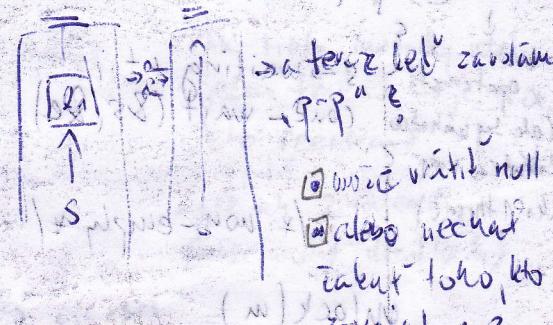
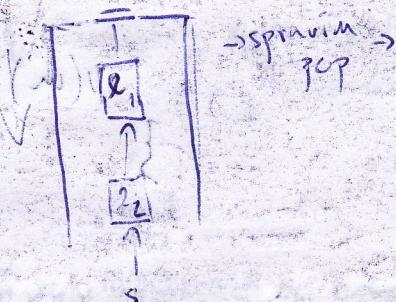
semafory < P(S) lock (ke semaforu zelený → ho uchováva červenú)
< V(S) unlock (zúš. semafor o 1) → sa vypárie z fronty čakajúcich
threadov a upustí ho do kritického

→ ak sú semafory binárne → klasický mutex

→ ak sú n-árne ⇒ potrebujete aj cond-wait a cond-signal

priklad: zdieľaný zásobník

pop(&s, &e)
push(&s, &e)
empty(&s)



→ ak bude nechat
čakajť toho, keď
závisia pop
alebo počas
preruvenia
čakania

② $\text{pop}(\&s, \&e)$
 {
 ~~lock(m)~~ → toto tu musíme učinit aby sa to zavolať v miestach kde sa to ale chce učinit
 najviac pred smarťom pop-om
 while ~~(s->next != NULL)~~
 {
 ...
 }
 else → waiting ++
 cond-wait (&m, &c)
 → ešte si myslím že trba tu decrement waiting --
 }
 unlock(&m)
 }
 }
 → syntakticky nijemoc správne keďže while a else :-)

$\text{push}(\&s, \&e)$
 {
 ~~lock(&m)~~
 if ($s->waiting > 0$)
 cond-signal (&s, &c)
 alebo tu
 unlock(&m)
 }
 }

$\text{pop}(\&s, \&e)$
 {
 ~~p(m)~~
 while ~~(s->next != NULL)~~
 {
 ...
 }
 else
 {
 s->waiting ++
 v(m) → možno, aby nas vedeli zo sveta
 P(c)
 P(m) → to je problem
 }
 V(m)
 }
 → pop (&s, &e)
 {
 lock(m)
 while ~~(s->next == NULL)~~
 {
 {
 opatrosť s->waiting ++
 alebo sa náhodou cond-wait (&s, &e)
 }
 unlock(m)
 }
 /* non-empty */
 unlock(m)
 }

$\text{push}(\&s, \&e)$
 {
 if ($r->waiting > 0$)
 V(c)
 }

$\text{pop}(\&s, \&e)$
 {
 ~~p(c)~~
 ~~p(m)~~
 ...
 V(m)
 }

26.2.2013

(3)

stack.h

push()
pop()

stack.c

static mutex_t m;
push()

pop()

{
 mutex_lock(&m);

...

mutex_unlock(&m);

} mutex-lock(&m);

while(stack is empty)

cond-wait(&c, &ml);

mutex-unlock(&m);

}

→ Java: prišiel som 30 min učiť

- preemptívny scheduling je v jave implementovaný

slide 34

→ server-client

• vytvorenie ConnectionHandler pre každého klienta / novinka: ak nenie klientsky thread ale anebo každý z klijantov má vlastný thread → preferuje decentralizovanú architektúru

• vytvorenie jedného -/-

pre všetkých klientov

→ preferuje centrálnu

blokujúce volania sú problematické v jednom aj v dvoch prípade

java používa MONITOR → bári všeobecnú threadom do jedného krit. úseku

synchronized → thread získava lock na mutex toho objektu

ak by sme napísali takéto kod: (str. 47)

public synchronized withdrawl (float amount) {
 deposit (-amount);

}

tiež synchronizovaná

reentrant

v dôsledku tohože je mutex stale reentrant

je to ok (ale je to ťažké)

Object self

withdrawl

deposit

1 thread to false

nie je to dobré

BankAccount

mutex

• Ak tieto metódy vzia
jeden thread → je to de
aktiv reentrant

• ak ak 2 threads
(može vzniknúť deadlock)

↳ synchronized metódy objektu (tiež sa o nich samozrejme základne nie synch. vzhľadom k tomuže tiež sú statické)

class A

sync m1 {
 ... m2 ()

}

class B

sync m2 {
 ... m1 ()

}

→ zároveň mutex objektu triedy B

class A

object-type A

zatiaľže metóda static a či už ktorému classu

class A

static sync m1(),
sync m2();

zavolať tiež objekt, čiže ak T1 zavola a T2 zavola →
→ uvedie ich oba zavolať lock na class-e
a T2 na objekte

↳ lepej ako sync metody sú synchronized bloky (ktoré si v pôsobstve identického synchronized object) s same code

↳ rôzne možnosti čakame vytvoriť nejaký objekt (objekt), ktor. bude mať

lockovaný

wait() → zodpovedá cond - wait → zmena sa atomicky locked ..

(→ môžu sa jednako uvažovať a jeho znam. premeny
tento objektu na ktorom je wait volaný)

notify() → odpovede cond - signal

→ zmena je prebiehať thread

notifyAll() → zavolať threads, ktor. reagujú na rovnakú condition

DÚ: do tyčina!!!

- p(), b(), wait()

→ treba implementovať class Synchronizer

java.lang

java.lang.parallel

vedľavu

volatile mutacie → hovorí o skutočnej kópii sa nachádzajúcej v pamäti (nie napr. na disk, pričom môže byť následne meniť premene v jednom mieste v niektorom druhom)

mutabilite hovorí, že ROZDÍL v touto premenou časom prebiehať možno, posiaľže

5.3.2013

TRANSAKCIJE

2-tier

log - má to, aby nás nedeli operácií rollback (ale je trans, odovzdaná a pod.)

- alternatívne commit, rollback, no samostatne copy - & database atol cyklické > recovery

slide 7

scheduler - reject - generuje následky

- execute - myšlienky

- delay - predĺži generáciu

napäť, nájdete nás riebať, že akorá transakcia pôjde príma, ale druhá, inakorá nájdete pôsobiť záberom

→ konflikty serverov (não)

→ transakcie, ktoré sú záberom kontrahent, ktoré sú nesúhlasiajú

CENTRALIZOVANÁ ARCHITEKTURA

- na jednom mieste

- stále je jeden server of fi

- 2 tier arch. → ktorý preprináči k database cez server

- ideale je poslat v programátorských systémov (via transakcií)

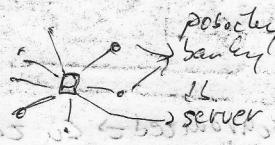
slide 3.1

→ náklad na Architectural complexity → pre distrib. databáze nám je serverovanie 1 ako pri 2+tier

→ klient je jedno, ke ktorému sa pripája

- následky: → database design more complex

Dany sú rôzne možnosti centralizovanej:



- chceme, aby systém bol zhodný

→ zároveň sú všetky resurse k dispozícii v jednom mieste

3-tier ARCHITECTURE

- han viaceri uzelov (slide 34)

- transakcia na meste ako je to tam dole nročené - (nie je s ktorou konzistencie?)

- matočná implementácia „spolu“

(5)

slide 36

→ nie konsistencie dôby (že robia klúč) → my predpokladame, že keď bude pokracovať uzel, tak urobí to
→ este existujú výpadky hneď, ak my budeme mať pravú horúcu ale výpadkovú uzel (je to jednoduchšie)

pozadavky (slide 37)

zo str. 34 Site manager sa musí vedieť opýtať, že aj grant commit alebo abort

38

Atom commit protocol

12.3.2013

M. 39 - ak jeden uzel → celom si odpadne abortom

- ak jeho koordinátor (ktorý vie že je abort → sa už nečaká zjednotiť)

[ak výpadok koordinátora procesu → sa koordinátor o výpadku dozviede zo zprávy užívateľa zo slide 39]

Acknowledgement → potvrdenie že vši sú úspešne zapísané (aby sme vedeli vymazať log file & poslať ACK)

→ snímať zo už zapísaného logu, aby už užitočný výpadok pre zápisor

do DB, aby nás po zápisu

niet viedlo opýtať koordi-

nátora až do skonči-

nia

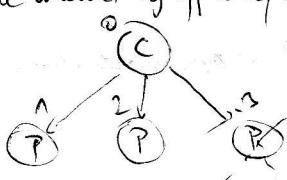
10

slide 40 - ak napr. jeden uzel → on bude môcť zahodiť na tú transakciu
(správni rollback)

→ výpadok na danej linke → abort až po správe od koordinátora

zadanie uviedené aj výpadky (2-fázový protokol súhrne)

→ ak miestny P povie, že je výpadok ⇒ koordinátor ABORT
(vrátiže v sebe info, aby výpadok podľa viedel)



výpadok uzel je môcť možné ťa transakciu zahodiť (správniundo v logu)

→ jediný zlyhavým je povie COMMIT ale potom výpadok (ostatných prípadoch už je "konečný" abortor) → v tomto prípade sa opýta koordi-
natora (ak aj koordinátor je výpadok → P je taká)

teoreticky nebude ďalšie, ak
to nám už aktívne (to neplatí, ak
nemáme problém)

ak by však v zlyhe certifikovať
výpadok / bolo záhanie → ale

→ zrehalo, ale je jeden zlyhavý prípad, keď po plasťaní
postavenie na vlasovanie výpadok C (tiež už obdobie výletok participantov)

→ namiesto správy o rozhodnutí od C mi dôjde správa že
C výpadol

P by mohol spraviť rôzne veci:

1., ak postavil WAS YET potom mohol sa rozhodnúť, že COMMIT, ale nie je to dobré, lebo očakávané uplynutie celého
mohol byť ABORT

2., ak ABORT → tiež výpadok ale všetci blajomali YET (potom lokálny abort je či už)

3., ak by čakal, keďže nelišil, aby mohli čakať na C a keďže je zlyhavý, celá súťaž by sa
blokovala (plasťa 2-fázového protokolu)

→ ale aj tak je to najlepšia možnosť (takže ešte čakať na výkonenie sa koordinátora)

4., aby toto by sa pýtal ostatných participantov, lebo napríklad este dôsti participant mohol
byť výpadok

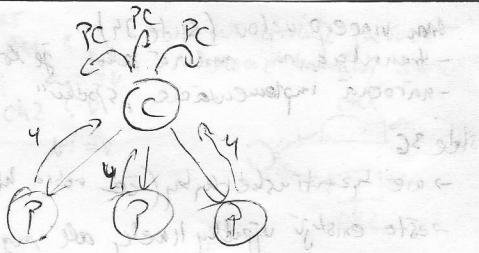
databázka horúci či uzel povie YET alebo NOD (ti nie je vec site managera)

11/42 → stavové diagramy

zadanie 2-fázový protokol nie je to pravé na mesanie dohoty (kvôli tomu možnosť výpadku koordinátora)

6. 2-fazový commit protocol

→ pridanie fáza 1, ktoré sa nazýva rovno "pre-commit"
(posielanie pre-commmitu (PC) nie je ale užívateľom, je PC na
nem je dosť ľahké kvôli zložitosti)



- ak výpredajec pred poslatím PC

- participant sa môže ostatujúcich členov opýtať, či má aktuálny základ pre-committu (takto je výrobca zabezpečený)

- spomedzi ďalších sa výberie nový koordinátor (je vedia ktorí sú jeho súčasťou)
- ak nový koordinátor nie je PC v sieti → sa snazi vysadiť PC do alej siete
- po poslatí nie nový koordinátor, tiež ešte sú v sieti pre-committ a tie sú prijaté ACK
- ⇒ spraví COMMIT a posle vsetkym

ak nový aktuálny v sieti PC je, akce vystih (pýta sa) ostatujúcich či nájde pre-committ

- ak zistí, že nájde → vysadiť vsetkym pre-committ

ak ani on nemá PC a ani aktuálne iný žiaden → ABORT

- zadržíme procesy na počiatku blokovania môžu opýtať, či sú dôvodky transakcie

v 2-fázovom → výpredajec členov sú v parciach, nezasahuje do rozhodnutia

môžu upozorovať aj linky (je to horečka než výpočetný rozsah)

- výpredajec linky:

koordinátor výpredajec pri poslatí napr. pre-comm PC ⇒ nový koordinátor (ten, ktorý dostal PC) →
⇒ nový koordinátor posielá PC ostatným ⇒ tie akt. výpredajec linky? → sú to akt. ktorí rozdelili na podcelky
môžu rozhodovať väčšina (majority protocol)
môžu výpredajec "pre-abort" (tie akt. komponenta/väčšina) načas rozhodnuť, že nový koordinátor
má aktuálny PA

môžu keďže až dolu vylepšili → následuje do existujúcej
→ tiež procesy dokončí

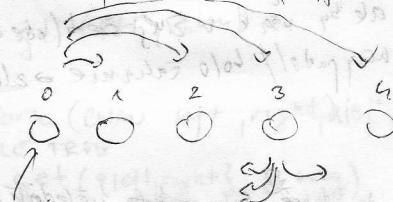
Resumé:

2-fázový → začína sa v prípade keď C výpredajec

3-fázový → rešta prostredom blokovania (keď výpredajec tiež procesy)

→ ak výpredajec patom musíme pridať iné funkcie pravidla)

ako by sme to implementovali (napr. ak tiež my do doc. Plackettu nabil)



→ ak kedy spravi broadcast, ak dostanú od koordinátora správu
(tiež kedy posle svoj was vystih a kedy občerstvi was od výpredajca)

transakcia skončila
o commit end (S)

→ je koordinátorom

19.3.2013

- pripel som až na slide 6

pr. slide 7:

- zameňanie treba spraviť na strane side managerov

Majority 2-fázový locking

- prečo sú ready druhé?



Quorum protocol

- rozdiel medzi väčšinou $w_i \Rightarrow \sum w_i = N$

$$Q_r + Q_w > N$$

$$Q_w = \frac{N}{2} \quad (\text{v prípade zdrobnených väčšin})$$

WFG - wait for graph

- slide 13 → v globalnom WFG môžete je dany cykly, preto musíme riešiť deadlock → abortne negatívne transakcie
 slide 15 → coördinator (jeden z S_1 alebo S_2 ; ten má význam informáciu faktualitácia) alebo proxy
 falloché cykly → abortívne tiež transakcie (keďže nedokážu mal nastaviť deadlock)

Synchronizácia časov

- každý proces má vlastné hodiny súčasne s opatravou
- predložiaci časové predstavy len na základe fyzických hodín je čas (slide 20) → (časopis + interna opatravu)
- slide 21 - existuje možnosť Lamporta aby mohli nesúhlasením sas → možnosť vektorov

$[6, -\infty, -\infty]$

$[6, 10, -\infty]$

$[6, 20, -\infty]$

$[6, 24, 60]$

$[6, 24, 60]$

$[6, 56, 60]$

$[6, 56, 60]$

26.3.2013

`fork()` - proces má napr. aj ďalší proces (v každom systéme mák)

→ v dôsledku vráti id ďalšieho procesu

- jeden zo spôsobov komunikácie medzi dve ďalšie je edielaná pamäť (zdieľajú prístup ako pri threadoch)
- ďalší spôsob : pipe (ten kde v súčasnosti používame) → do 1. časti sa posúva do ďalšej procesu
 int read fd; file na čítanie & pipe
 int write fd; file na písanie & pipe
 → read(fd +m, p, int nbytes) → pointer, kde sa majú dať
 write(fd +out, p, int nbytes) → keďže tu je to sa má písť
 ↓
 FIFO fronta
 ↓
 pointer, kde aktuálne
 do pamäti

KANAL - zabezpečenie pipe-ov

FIFO

- model: kanál je abstrakcia
 pipe je materializácia

interná kapacita pipe-ov je ajm 1 integer

okrajové príslušenstvo:
 type má konkrétnu kapacitu
 ako sa zachovať read ak je pipe prázdný

int read wriat
 int write } kde bývajú bytové sa podľa čo sa písat / čítať

ASYNCHRONNE PIPE -

- read napsáva predtia 100 a 500 byteov, alebo 0 byteov a nabytes
- write nie je možné čítať - blokujúce
- preto treba nastaviť flag: fctrl() → ak je ta pipe má charat

do kanálu sa písú správy / a vysenávajú sa správy
 do je správa? → hoci jakej koncretné postupnosť bytov

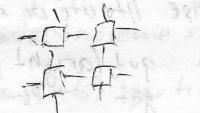
OCCAM - program, ktorý sa využíval z hardvérom s názvom Transputer - procesor

čip so korešidkami

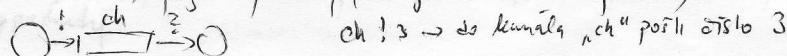
existujúce procesy (primitíve): ostacacie sú zložené z fielde

:= - primitívne a := 2 (process, kt. má v súčase ľahšiu formu)

data sa umiestňujú v rôznych procesoroch



kanál v OCCAME → spája 2 procesy (nie je to iba FIFO fronta)



ch ! 3 → do kanála, "ch" posíli číslo 3

→ kanály majú 0-ovú kapacitu v OCCAME, pretože prenájíci proces reabdukujú, ktorý niesie niektoré trojky (ch 2, v)

ďalšie procesy: IF, WHILE, FOR, SEQ, PAR, ALT

potom osajav skončenia

prijme z ch do skončenia, v"

8.
 SEQ
 $a := 3$ → sekvenčné vykonávanie
 $b := 1$
 ALT
 $a := 3$ → 2 procesy sa vystavávajú paralelne
 $b := 1$
 ALT
 $ch_1 ? v$ → ALT doručuje posielat viacé kanály následne
 $a := v$
 $ch_2 ? v$
 $b := v$

sledujúci filozofi súčasne používajú procesor a pipe-ov (filozof)

→ výkon je jednoduchý

PROC phi1 (CHAN left, right)
 WHILE TRUE
 SEQ
 think()
 left ! 0
 right ! 0
 eat
 left ! 0 → vráti vŕtak
 right ! 0

PROC f (CHAN left, right)
 WHILE TRUE
 ALT
 left ? msg → left ? msg → ak nie pošle zľava, tak vŕtačka vŕtačka na ľavú /akej na ľavú, ale ju filozof vráti
 right ? msg
 right ? msg

SEQ
 ch1 ! v
 ch2 ! v
 ch3 ? v → ak keďž takto
 ch4 ! v napsaný
 ch1 ? v tak HNEDE deadlock
 ch2 ? v keďž kedyž ešte
 ch3 ? v3 na pravú následuje
 ch4 ? v4

→ preto treba 2 procesy prevezť do skupín a tie skupiny bolo nutné kódy (akej aby to fungovalo)

1. put (ch, vn)
 2. get (ch, *vn)

PROC fork (CHAN left, right, aleft, aright)
 WHILE TRUE
 SEQ
 get ({left, right}, &msg)
 if (vn == left)
 { put (aleft, 0);
 get (left, &msg);
 put (aleft, 0);
 }
 else // to čiže vyskutím kanálom
 { put (aright, 0);
 get (right, &msg);
 put (aright, 0);
 }

PROC phi1 (CHAN left, right, aleft, aright)
 WHILE TRUE
 SEQ
 think()
 put (left, 0)
 → get (aleft, &msg)
 put (aleft, 0)
 → get (right, &msg);
 eat ();

→ výprava, že get bude mať robiť z univerzálnej kanálov got({3, -}), hoci aj iba jeden kanál bude v univerzálnej



-ak správę fak, tak tiež musí ~~sie~~ jmenovať faj svoj nový vypočítavý proces
-tieže ak chce správę edit, fak asi teda obnovovať súhlas

1. otáčka



↪ kde zmiene, zo so zmyslom, tak nectime to tak, aby druhý dcíčdal

2. otáčka



↪ myšing, kde sa zahľadí, čo ďalšia miestna (takže ak nikt nechce odiť)

3. otáčka



↪ 2. správa tomu bude blokovať do ukončenia

↪ nesenie: este pred skončením treho, počle príjmom ďalšieho do konca správ
↪ skončení a môže skončiť ⇒ druhý časťou správu o tom
že 1. proces skončil



↪ bude nás spoľahlivý defektív, keď je 1. proces „počas“

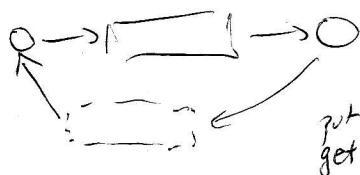
↪ ak 1. skončí, tak tento defektív zapíše správu do konca, že príjde skončil

↪ keď 2. časťou správu, tak ešte potom odblokuje (takže bol časťou blokovaný)

2.4.2013 - Veľká noc

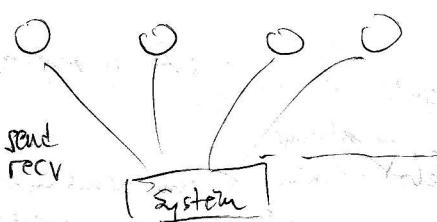
9.4.2013

-príslušnou 26 min. ueskör



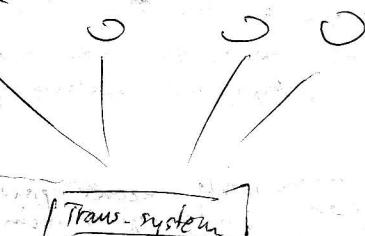
Point-to-point

Procesy



Transakcie

Language binding



Implem.
binding

[Imp.]

send(p, m, R, S)
recv(p, &m, S', S)

možnosť renderovať
od kt. je ochutnávaný proces „P“

[recv(p, m, R, S)] → toto je strošok my zapisatice

(10) Def: $[send, p_s, m, R, s]$
 $[recv, p_r, \&m, S, s_r]$ má zdrožený páru, keď $p_s \in S \wedge p_r \in R \wedge$ redukčné my kresli rovn.
par, keď sú zložky prebiehajúce

message, kde v piatke na 1. operácii môže byť použitý rôzny alebo druhý

Def: System sme výkonavateľ len zdrožené páry operácií (pri point-to-point)

Def: Výkonanie páru $[send, p_s, m, R, s]$, $[recv, p_r, \&m, S, s_r]$ znamená atomický krok:

$\hat{n}_m := m_s$

Os

2. delete (α_s)

message sa kopíruje a je presúvaná (potom pre konsistenciu modelu,
 za prve message zanikol)

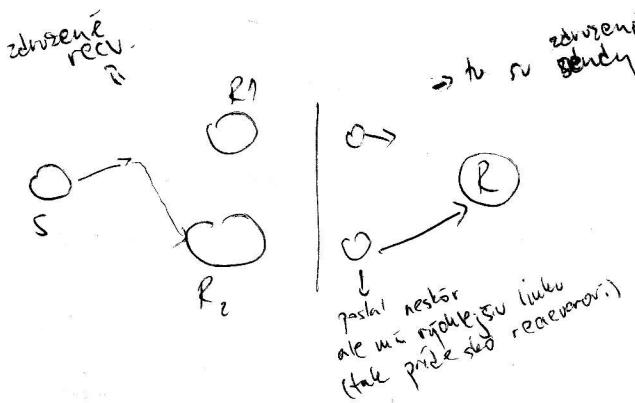
3. if ($s_s == \text{NULL}$)
 signal (s_s)

4. if ($s_r == \text{NULL}$)
 signal (s_r)

5. delete (α_s)
 6. delete (α_r)

Def: Ak v uja kou momente existuje nesúhlas zdrožený páru operácií Os, Or, tak ten system ~~nie~~ niekedy
 výkoná aspoň jednu z operácií Os, Or

$\begin{cases} \checkmark [send, 0, m, \{1, 2, \dots\}] \\ \checkmark [recv, 1, \&m, \{0\}, \dots] \end{cases}$ 1. zdrožený páru
 $\xrightarrow{\text{system vž. tuto garantuje, že aspoň jeden zo zdrož. op. výkoná}}$
 $\checkmark [recv, 2, \&m, \{0\}, \dots]$ 2. zdrož. páru (môže výkonáť a potom 3.)



vezedenie borce

- kresme len (maltevery - akô niečo kresť v predmete)
 - ak by len 4 predstavali, tak sa nenechajme kresť

borce fungujú takto: pri každom prídu borcevi pári (vymárahľia aktivity) → spredajú pacient, že bolo ich predané a kresia a zostavia páry (a pacient zlyšuje násobky ich aktivity)

new(m)
 send(m, \{1\})

není aktívny, že
 toto možno písať aj ~~new send(m, 1)~~, aby hľadanie výrazových názvov bolo ľahšie

recv(\&m, \{0\}) → tu nám nenechá vytvárať nové nazvy (recv to spravi za nás)
 unikajúce meno

new(s)

recv-init(s, 0)

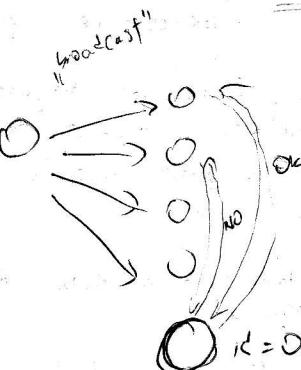
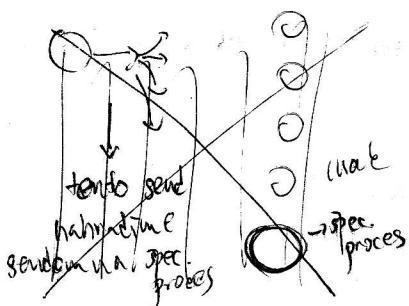
recv(\&m, \{0\}, s)

recv-wait(s)

→ všetky meničie týchto možností písať recv(\&m, \{0\}) a potom, keď je blokovacia

[send, $P_1, m_1, \{P_1, P_2, \dots, P_n\}, m\}$

→ scheme matriční sendem, kde uživatel nemá



procesy po recec. zistia, že ten msg
me je korektný (že je všetky, zusti z hľadisk správ) \Rightarrow sa pyta
spec. proces, on odpovie procesom
či OK alebo NIE OK.

abu súce rozhodnosť (deterministic):

- jedine program posúvajúci 2 procesor (sedla rozdielne na sase, merielajú uči, obrovské súčasne sa nazýva)
- komunikatívny systém (abstrakcia)
- procesy majú ID, posúvajú sa, je ich konečné uči
- pri posúvaní posúvajú aktivity relevantného kanálu (procesy nás prejedú správy)

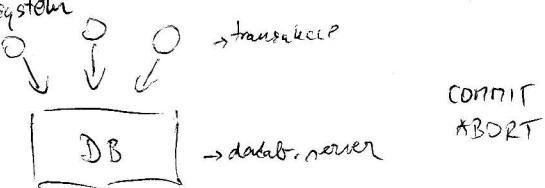
ale u nás je užrovnosť v užrovnosti kanálu,

- aký bude kanál písaný (to nazívame dát posor, či reac. je blokovací
alebo nie peri Δ) \rightarrow reac. neblovací
reac. blokovací

16.4.2013

- prosiel rámec cca 20 minu neskôr
- ako sa nazýva n chybami a distrib. myšlienoch?

základový systém



\rightarrow možno sa pokračovať klient (transaction) alebo linken

malloc

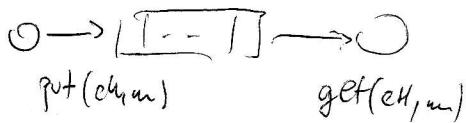
free

open

read

write

\rightarrow je potrebné očesťovať operátorov, ktoro
výnimočne nastaví alebo rôzne procesy nie je
možné v operačnom systéme.



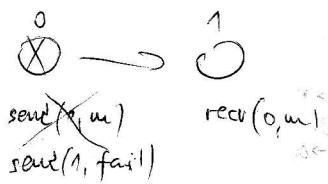
\rightarrow do sa stane, ak myslíme správ - a posielal správu
- neposalal správu

• aktív

• kanál

fail stop chyby

\rightarrow potenciálne využitie množstva (nevziať ostatné komunikácie) \rightarrow myšlienka škodia



\rightarrow ke korektnému bolo, aby O. pred informáciou posielal & procesom, ktorý má ďalej
správu, re "koncovú"

send(1, fail)@

\rightarrow myš. myslí, že nelen "dokončujúci" proces, ale samozrejme s myšlenkou

\rightarrow ak myslíme komunik. médium / kanál \rightarrow to nie je česne

(12) 7.5.2013 - príde súm 15 min neskôr
- pokračovanie z prednášky (J. Janíček) z 30.4.2013

WiFi → kanály (v rôznych krajinach, sú možné používať rôzne kanály)
- reálna rýchlosť WiFi je polovičia reálnej hodnoty, ktoré sú na slide 802.11b ~11 Mbps (užitočnosť cca 6 Mbps)

5GHz - má väčší problém prechodu cez polohy

802.11n → pomereľne rýchle

BSS - mimoľudská stanica, ktorá súčasťou v dorehu (napr. Access Point a veľké rôzne okolné nete)

ESS - 1+ BSS - tiež

- vlastnícky rádiosík s vecmi (infrastruktúrou)

AP
DS - ak sme v jednej BSS a chceme poslat do druhej BSS, tak to viestí DS (postreď spravu AP-ku v tej BSS-kde bolo sa nachádzat pôvodný zdroj)

príklad → distančný rádiosík WiFi a kde vieme, že máme poslat do ethernetu, tak pretočím rádiosík až na ethernet

→ to bola funkčná vrostva

jsietová vrstva

IP packet

TTL → po kažko routeru máte mal. preť (lebo ak nastane cyklus, aby sa po čase zadržal) - Hop

Routovanie

- Ethernet má IP - typického riadiaceho pole

adresa - eth0 → interfejs, oč ktorom poslat
adresa next do ktorého chcem poslat

otáčka: aktívna adresa → a máme získať b, ktorý závisí z routovací tabuľky a posíli
(či adresa "a", pri ktorého riadiaceho pole je maska "m")
return (a & m == s)

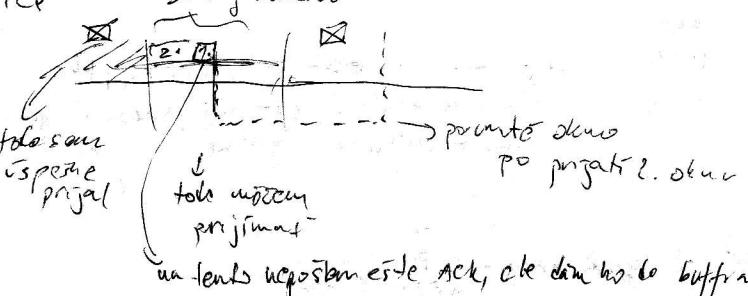
ARP

- ARP cache → ochrana proti záťahu

UDS → normálne spracovanie ako ucastníckej metódy (unicast, ..., 1:1)

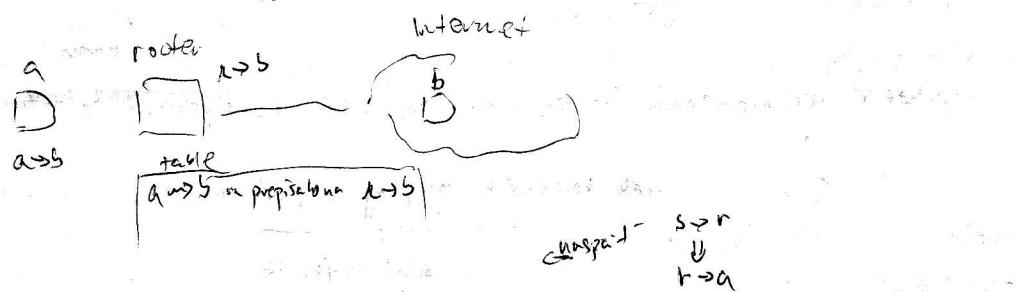
→ koncový siedací aj vlastnícky aj siedací

TCP



NAT

- SNAT →



- DNAT

- ak dešia nás my posílat k serveru

DNS - aktívne je aký súci je aktuálny paralel. IP adresy (najväčšia distrib. databáza na svete)

- posíti chary

NS → chary (ich užívateľov nie je 11 užívateľov na sledoch)