P O Z N Á M K Y K P R O P O J O V Á N Í A S Y N C H R O N I S A C I P R O C E S Ů V J S D

Michal Kretschmer, prom. mat.

Tento příspěvek dále rozvíjí článek /2/, jenž obsahuje výklad Jacksonovy metody vývoje systému. Opírá se přitom o knihu /1/ a zaměřuje se především na více "programátorské" partie látky, které vzhledem k vymezenému rozsahu článku /2/ nebylo možné do něj zahrnout. Zabývá se modelováním vybraných akcí entit, důsledky hrubého míšení na programovou strukturu procesu, postupem pro zabránění paralelního probíhání procesů, speciálními samostatnými synchronisačními procesy a využitím separátních CLOCK procesů pro zabezpečení pozdržení provádění procesů.

1. Modelování jen vybraných akcí entit propojováním procesů pomocí dat

V některých případech omezení na pořadí akcí entity jsou takové, že se nedají dobře vyjádřit v jediné struktuře. V takových případech vyjádříme strukturu entity více strukturami, z nichž každá zachycuje jen některé z akcí. Každé z těchto struktur odpovídá pak proces, přičemž tyto procesy musí být propojeny pomocí dat. Ukažme si to na příkladu :

Nechť entita P provádí akce A, B, C, D a E a nechť přitom po A může bezprostředně následovat některá z akcí B, C, D, E; po B některá z akcí A, C, D, E; po C některá z akcí A, B, C, D; po D některá z akcí A, B, E a konečně po akci E některá z akcí A, B, C. Nechť přitom po každé akci A musí (i když ne nutně bezprostředně) být provedena akce B, podobně po akci C musí někdy následovat akce D a po akci D musí někdy následovat akce E. Při těchto omezeních mohou tedy akce C, D a E být proloženy akcemi A a B a obráceně akce A a B mohou být proloženy akcemi C, D a E. Je zřejmé, že úsilí znázornit v jedné struktuře vyčerpávajícím způsobem všechna možná pořadí akcí této entity, by vedlo ke kom-

plikované, zcela nepřehledné struktuře.

Problém vyřešíme tím, že strukturu entity P vyjádříme pomocí tří struktur, kterým budou odpovídat procesy P, PAB a PCDE propojené pomocí dat. Proces P při výskytu akcí A a B bude zapisovat data pro proces PAB a při výskytu akcí C, D a E bude zapisovat data pro proces PCDE. Procesy PAB a PCDE budou tedy modelovat jen vybrané akce a omezení na jejich provádění. Struktura těchto procesů je znázorněna na obr. 1, 2 a 3. Takové modelování můžeme využít jak na úrovni vstupního podsystému pro zabezpečení kontroly správného pořadí prováděných akcí, tak i na úrovni funkčních procesů, které mohou dále strukturu entity rozpracovávat s ohledem na požadované funkce. Tak v našem příkladě bychom mohli do procesu PCDE přidat před iteraci komponenty C na začátek sekvence CDE ještě komponentu C-PRVNI, čímž bychom od sebe odlišili první výskyt akce C od dalších výskytů akce C prováděné před akcí D (na obr. 3 znázorněno čárkovaně). Procesy PAB a PCDE budou implementovány jako korutiny procesu P invertované vůči předávaným datům.

2. Důsledky hrubého mísení na programovou strukturu procesu

Jak již bylo vysvětleno v /2/, kap. 8, hrubé mísení není zcela deterministický proces, neboť jeho výsledek závisí na relativní rychlosti jednotlivých procesů. Výsledná programová struktura musí tedy zahrnovat všechny případy, ke kterým může dojít, včetně těch, které představují vlastně chybu a vyžadují tudíž zvláštního ošetření. Ukažme si to na následujícím příkladu :

Do procesu ZAKAZKA vstupují dva proudy dat, jež jsou v něm čteny pomocí hrubého mísení. Jeden z nich jsou data vytvářená procesem PARTNER, která obsahují informace o provedených akcích entity PARTNER, a to UDELENI, ZMENA, PRIJEM a ZRUSENI. Struktura těchto dat zapisovaných procesem PARTNER je na obr. 4. Druhý z těchto proudů dat vstupujících do procesu ZAKAZKA je nejvýše jedna věta vytvářená procesem DODAVATEL (viz obr. 5), jež obsahuje informaci o provedení akce ODESLANI. Přitom však dodavatel zboží neodešle, když partner včas zruší zakázku. SSD všech těchto zmíněných procesů je na obr. 6. Entita

ZAKAZKA má společné akce s entitami PARTNER a DODAVATEL.

Struktura procesu ZAKÁZKA by měla vyjadřovat smysluplné vztahy mezi daty udělení zakázky, změny zakázky, odeslání zboží, příjmu tohoto odeslaného zboží a zrušení zakázky. Je zřejmé, že měnit zakázku lze jen do odeslání zboží nebo jejího zrušení a že nelze odeslat zboží po zrušení zakázky, tedy že strukturu tohoto procesu by měl vyjadřovat obr. 7. V důsledku hrubého míšení, stejně tak jako ve skutečném světě vlivem zpožďování předávaných informací, se však může stát, že požadavek zákazníka na změnu přijde po odeslání zboží nebo že požadavek zákazníka na zrušení zakázky přijde až po odeslání zboží. Takové případy musí struktura procesu ZAKÁZKA respektovat, takže jeho struktura bude komplikovanější (viz obr. 8). Setkávají se tu dva pohledy na zakázku : zakázka z hlediska partnera a zakázka z hlediska dodavatele. Změnu, která přišla pozdě, nelze provést, pozdní odeslání zboží (po požadavku partnera na zrušení zakázky) je třeba dále řešit buďto dodatečným příjmem objednaného zboží, které došlo zákazníkovi, nebo jeho odmítnutím a zasláním zpět. Jestliže v důsledku hrubého míšení by přišla informace o odeslání zboží po informaci o příjmu dodávky, nebude to na závadu.

3. Vzájemné vyloučení procesů

Procesy v JSD modelu probíhají nezávisle na sobě rychlostí, která není definována. Při propojení procesů pomocí stavového vektoru vydání instrukce getsv neblokuje žádný z procesů. K blokování procesů dochází jedině při provádění instrukce read, kdy proces musí čekat až budou pro něj k dispozici příslušná data. V některých případech však potřebujeme zajistit, aby procesy v určitých situacích nemohly probíhat současně, aby se jejich skutečné provádění vzájemně vylučovalo v tom smyslu, že v jistých okamžicích může v systému být prováděn pouze jeden z těchto procesů.

Nechť procesy A a B jsou spolu propojeny pomocí dat DB zapisovaných procesy A a zároveň pomocí stavového vektoru VB čteného procesy A (viz obr. 9). Procesy A na základě vstupních dat DA a obsahu stavového vektoru VB procesu B zapisují data pro proces B, na jejichž základě proces B mění svůj stavový vektor.

Jedná se vlastně o analogii známé situace, kdy při možnosti čtení a změn vět z více terminálů hrozí, že mezi čtením a zápisem věty z programu odpovídajícímu některému terminálu by mohlo dojít ke čtení téže věty z programu pro jiný terminál s následnou změnou věty, takže celkový výsledek by pak byl nesprávný. Procesem B může být např. účet, procesy A jednotlivé finanční transakce vztahující se k tomuto účtu. Pseudokód procesů A a B lze zjednodušeně vyjádřit takto :

```
A  itr                          B  itr
   read DA;                        read DB pomocí hrubého míšení;
   getsv VB;                       zpracuj vstup DB;
   zpracuj vstup DA;               updatuj stavový vektor VB;
   write DB;                    B  end
A  end
```

Může se stát, že tentýž nebo jiný proces A získá dvakrát po sobě stejný stavový vektor VB, protože mezitím nebude dokončeno zpracování dat DB v procesu B a tedy stavový vektor VB nebude updatován před vydáním další instrukce getsv VB z téhož nebo některého jiného procesu A. To očividně vede k nesprávným výsledkům, čemuž musíme zabránit. Potřebujeme, aby instrukce getsv VB mohla být vydána až po updatu VB pro předchozí data DB.

K tomu nestačí zabezpečit, aby z procesů A vždy jen jeden mohl provádět sekvenci instrukcí začínající read DA a končící write DB, neboť podle JSD definice procesů není zajištěno, že getsv VB nebude vydáno dříve než byla procesem B zpracována data DB zapsaná dříve prováděným procesem A. Nezbývá než propojení pomocí stavového vektoru B nahradit propojením pomocí dat DI a DS (viz obr. 10). Procesy A zapisují data DI kdykoliv požadují, aby proces B jim jako obsah dat DS předal obsah svého stavového vektoru. Pseudokód procesů A a B lze pak zjednodušeně vyjádřit takto :

```
A  itr                          B  itr
   read DA;                        read DI;
   write DI;                       write DS for A(i);
   read DS;                        read DB(i);
   zpracuj vstup DA;               zpracuj vstup DB včetně změny
   write DB;                                stavového vektoru;
A  end                          B  end
```

Přitom instrukcí read DI v procesu B se rozumí přečtení některých dat DI zapsaných v procesech A. Tato data jsou čtena v pro-

cesu B technikou hrubého míšení, popř. je čte speciální proces provádějící toto hrubé míšení vložený mezi procesy A a B. Data DS jsou zapisována pro ten proces A(i), ze kterého byla zapsána data DA přečtená při hrubém míšení. Z téhož procesu jsou pak čtena předávaná data DB. Proces B nezmění stavový vektor, dokud nepřečte příslušná data DB, tj. dokud příslušný proces A nedokončí zpracování svého vstupu. Podobně proces A nezíská data DS (což je vlastně stavový vektor procesu B), dokud není dokončena změna stavového vektoru procesu B pro dříve obdržené požadavky DI na zpracování. Tím je zajištěno vzájemné vyloučení provádění procesů A a B v jejich rozhodujících fázích.

4. Synchronisační procesy

V některých případech je třeba zavést speciální samostatné synchronisační procesy, které zabezpečují správné pořadí prováděných činností v různých procesech. Nechť např. procesy A zapisují v pravidelných časových intervalech (např. na závěr dne) data DB pro proces B (např. součet denních pohybů), jenž tato data tiskne spolu se závěrečným součtem, a to se stejnou periodicitou. Dosud vytvořený SSD ukazuje obr. 11. Přitom je třeba zabezpečit, aby všechna data DB předávaná procesy A byla procesem B přečtena dříve než bude tištěn závěrečný součet. Proces B však nemusí mít k disposici informaci o tom, kolik těchto předávaných dat DB obdrží v jednom cyklu, neboť data DB mohou zapisovat jen některé z procesů A (např. ty, pro které během dne došel alespoň jeden vstup DA).

Problém nelze řešit tím, že proces B vybavíme vstupem dat časových pulsů, neboť bychom se přitom museli opírat o relativní rychlost procesů A a B, což je nepřípustný postup právě tak jako pozdržení zahájení cyklu zpracování v procesu B o nějaký časový úsek po tom, co všechny procesy A obdrží na závěr periody zpracování signál k tomu, aby zapsaly data DB.

Pro zabezpečení stanovených požadavků zařadíme do budovaného systému speciální synchronisační proces SYNC, jehož propojení s ostatními procesy je znázorněno na obr. 12. Proces SYNC čte data časových pulsů TS a pro každý z procesů A zapíše požadavek TA na vytvoření dat DB. Každý z procesů A obsahuje instruk-

ce write DB (ta může příp. odpadnout) a write RA v tomto pořadí. Synchronisační proces SYNC čeká na přečtení dat RA od všech procesů A a pak teprve předá procesu B signál TB, kterým jej informuje o tom, že přečetl všechna data RA, tedy že všechny procesy A již zapsaly data DB. Proces B po přečtení TB vytvoří závěrečný součet a zapíše data RB pro proces SYNC, který je tak informován o ukončení jednoho cyklu zpracování. Pseudokód procesu SYNC je tento :

```
SYNC  itr
   PERIODA  seq
      read TS;
      write TA pro všechny procesy A;
      read RA pro všechny procesy A;
      write TB;
      read RB;
   PERIODA  end
SYNC  end
```

Přitom hrubé mísení v procesech A a B musí upřednostňovat DA před TA, resp. DB před TB, např. analogicky tomu, jak je znázorněno na obr. 25 v /2/. Všechna data DB pro proces B jsou přitom zapsána dříve než proces SYNC zapíše data TB, takže (z důvodu upřednostňování DB v hrubém mísení před TB) proces B zpracuje všechna data DB dříve než je přečten TB.

5. Pozdržení provádění procesů

V některých případech, zejména u systémů pro řízení technologických procesů, je třeba zajistit, aby provádění procesu bylo na určitou dobu pozdrženo, tj. aby mezi prováděním dvou po sobě následujících instrukcí v procesu uplynul jistý předem specifikovaný čas.

Nabízí se možnost vybavit takový proces dalším vstupem, tj. daty časových pulsů, a mezi dvěma instrukcemi, kdy má být provádění procesu pozastaveno, prostě počítat časové pulsy. Tento přístup je však nevhodný, neboť data časových pulsů přicházejí jako vstup spojitě, tedy je třeba je číst nejen v té komponentě, kde je to žádoucí, ale ve všech komponentách procesu, kde případně neplní žádnou funkci, čímž se komplikuje celá logika procesu. Kromě toho by bylo nutné data časových pulsů číst technikou hrubého mísení a ostatními daty procesu, neboť jinak

během instrukce read by proces mohl být blokován, takže by ani nemohl číst data časových pulsů.

Další možností je vybavit systém jedním centrálním CLOCK procesem, který čte data časových pulsů a jehož stavový vektor obsahuje proměnnou udávající čas. Proces, jehož provádění je třeba pozdržet, lze propojit s CLOCK procesem pomocí stavového vektoru VC. Pozdržení procesu lze pak realizovat jako dynamický stop např. takto :

```
getsv VC;
počáteční-čas:=čas of VC;
CEKACI-SMYCKA  itr (until čas of VC > (počáteční-čas+pozdržení))
   getsv VC;
CEKACI-SMYCKA  end
```

Z hlediska specifikace systému je to korektní řešení, neboť v této fázi vývoje systému předpokládáme, že každý proces má k dispozici svůj vlastní procesor. Při implementaci systému, kdy více procesů musíme sdílet na jednom procesoru, bychom však narazili na potíže. Poznamenejme, že úloha, kterou se zabýváme, není bezvýznamná, ani když systém implementujeme na výpočetní technice s operačním systémem, na němž je k dispozici instrukce wait po specifikovaný čas. Taková instrukce je totiž prováděna na úrovni jedné úlohy z hlediska multiprogramování operačního systému, přitom však obvykle počet paralelně zpracovávaných úloh je menší než počet procesů v JSD specifikaci systému a ném se jedná o pozdržení na úrovni JSD procesu.

Lepším řešením pozdržení procesu A je uspořádání dle obr. 13. Na začátku doby čekání zapíše proces A data DZAC, po uplynutí požadované doby čekání zapíše proces CLOCK data DKON. V procesu A je čekání realizováno po sobě následujícími instrukcemi write DZAC a read DKON. Proces CLOCK musí být přitom specifikován separátně pro každý proces, ve kterém je třeba realizovat pozdržení. Jeho pseudokód může např. být :

```
CLOCK  itr
   read DZAC&TGM;
   SKUP  sel (TGM)
   SKUP  or  (DZAC)
      čítač:=čas-čekání;
      CEKANI  itr (until čítač=ø)
         read TGM;
         čítač:=čítač - 1;
      CEKANI  end
```

```
      write DKON;
   SKUP  end
CLOCK  end
```

Logika procesu CLOCK vychází z předpokladu, že proces A nezapíše další data DZAC dříve než přečetl data DKON, tedy že v procesu A se současně může vyskytovat jen jedno čekání. Takto specifikovaný proces CLOCK nelze tedy použít ve spojení s více procesy, ve kterých během pozdržení jednoho procesu může dojít k požadavku na pozdržení dalšího procesu. Pro více pozdržovaných procesů A bychom tedy měli stejný počet procesů CLOCK. Při implementaci procesů CLOCK bychom nemohli použít separaci stavových vektorů, neboť po přečtení každého TGM potřebujeme odečíst 1 od čítače pro všechny čekající procesy, nikoliv tedy jen pro jeden z nich. Pseudokód takového procesu, který je propojen s více pozdržovanými procesy, může být :

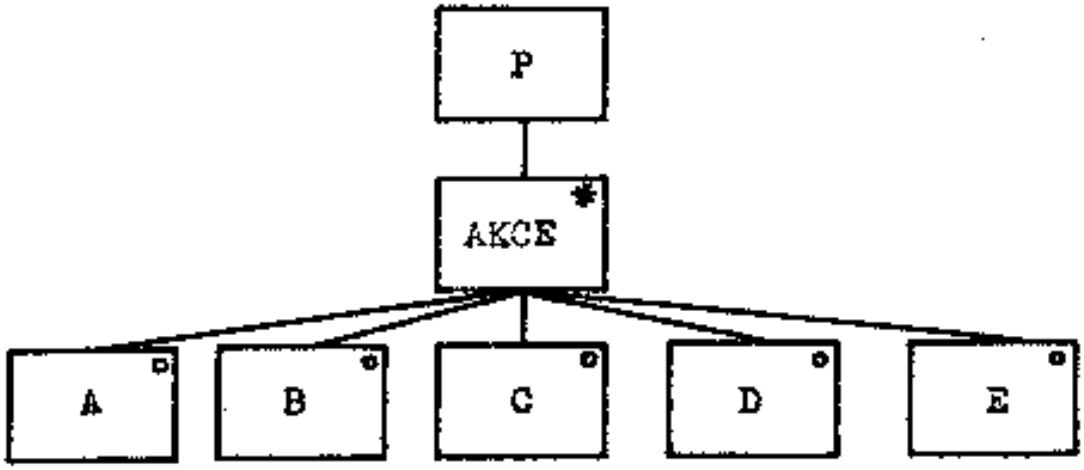
```
MULTI-CLOCK  itr
   read DZAC(TGM;
   SKUP  sel (TGM)
      čti první prvek seznamu;
      SEZNAM  itr  (until konec seznamu)
         čítač:=čítač - 1;
         KON  sel  (čítač=-1)
            write DKON(process-id);
            vyjmi přečtený prvek ze seznamu;
         KON  or
         KON  end
         čti další prvek seznamu;
      SEZNAM  end
   SKUP  or (DZAC)
      vlož nový prvek do seznamu;
   SKUP  end
MULTI-CLOCK  end
```

Data DZAC, jež zapisují jednotlivé procesy na začátku čekání, obsahují identifikaci procesu a dobu pozdržení; každý element seznamu obsahuje identifikaci procesu a čítač, přičemž hodnota čítače je při vložení dat DZAC jako nového prvku do seznamu rovna požadované době pozdržení.

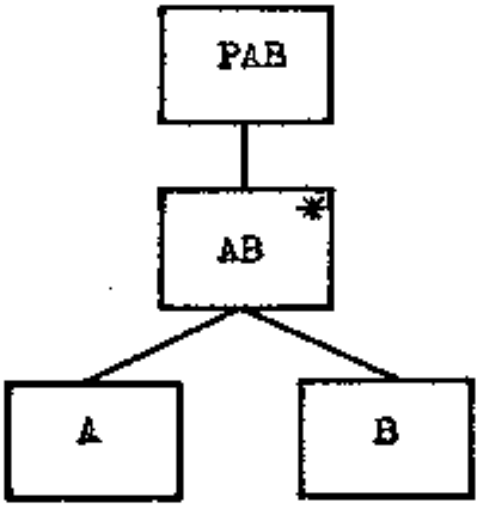
Literatura :

/1/ Jackson M.A.: System Development, Prentice-Hall International, Englewood Cliffs, 1983

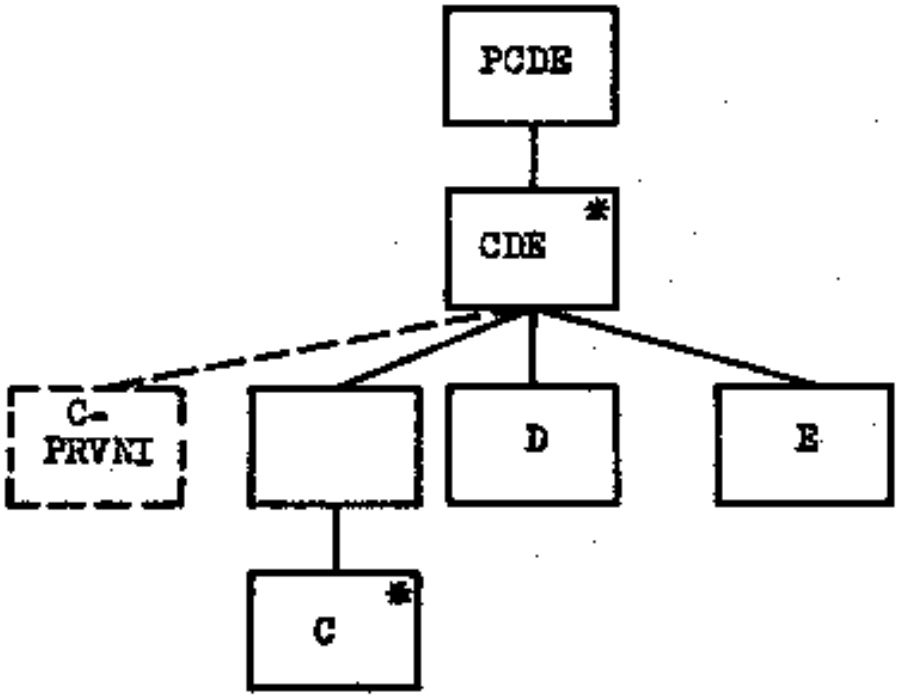
/2/ Kretschmer M.: Jacksonova metoda vývoje systému, MAA, 1986, č. 3-5



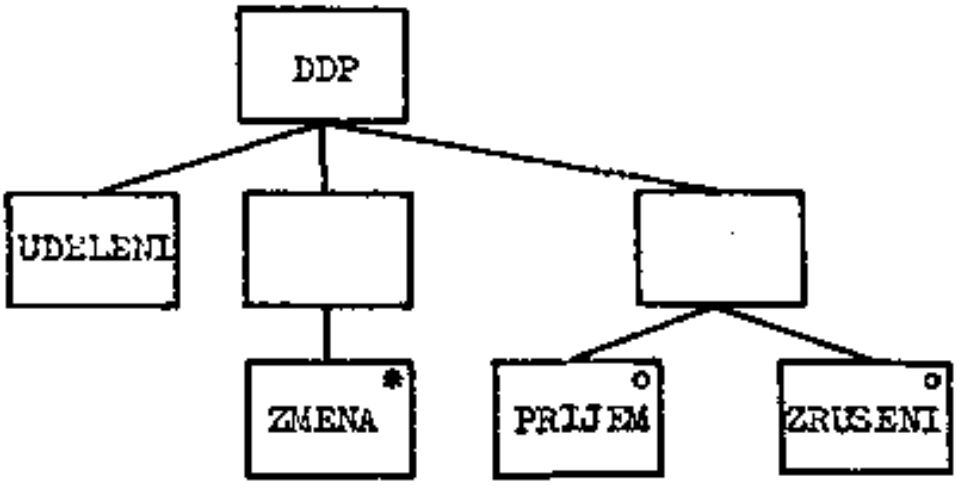
Obr. 1 Struktura procesu P



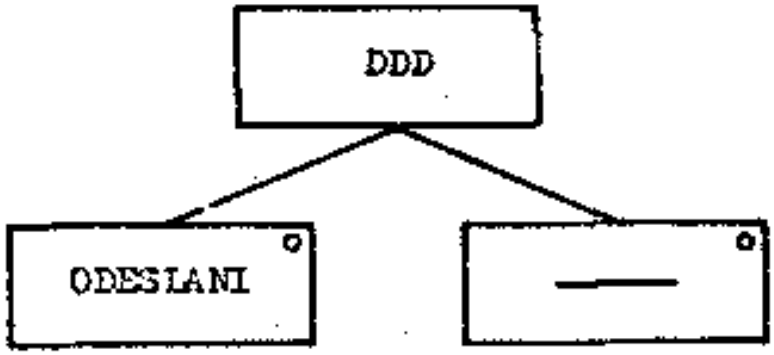
Obr. 2 Struktura procesu PAB



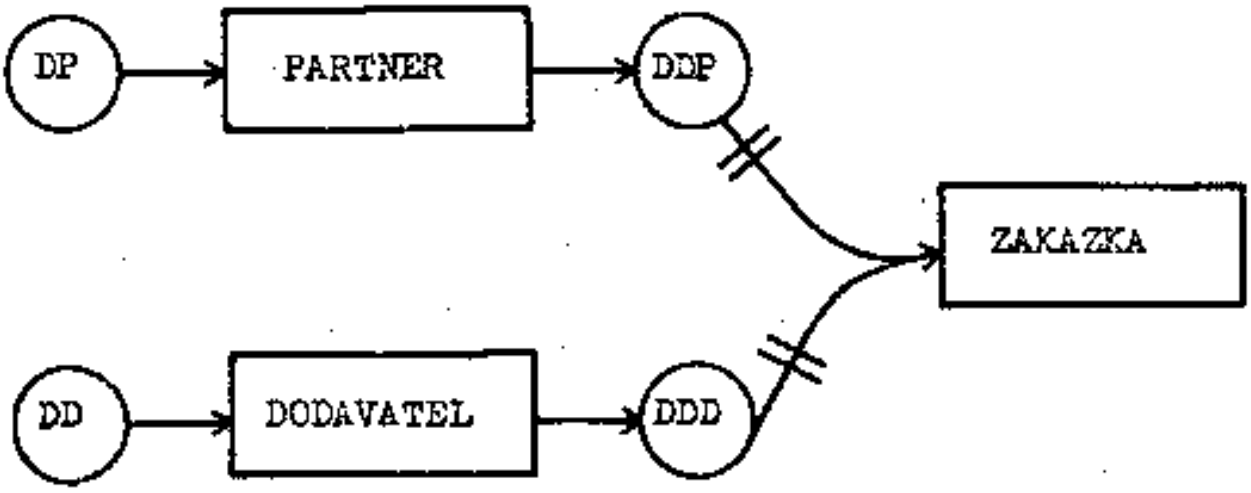
Obr. 3 Struktura procesu PCDE



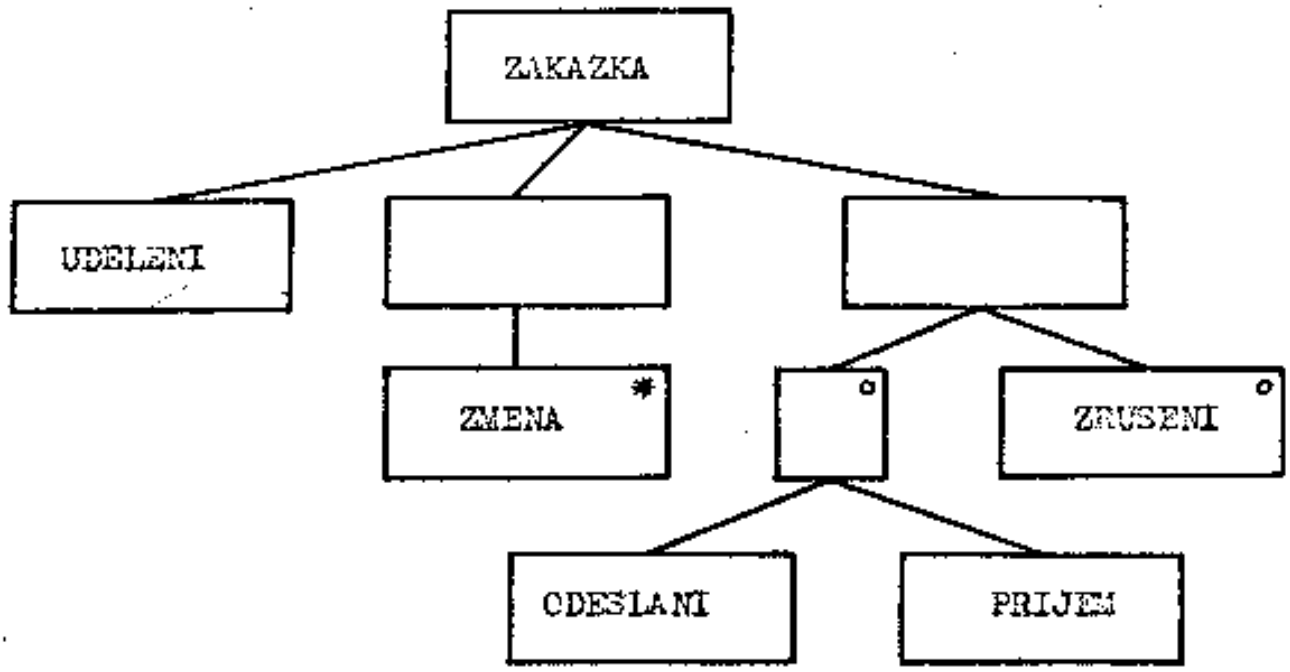
Obr. 4 Struktura dat předávaných procesem PARTNER



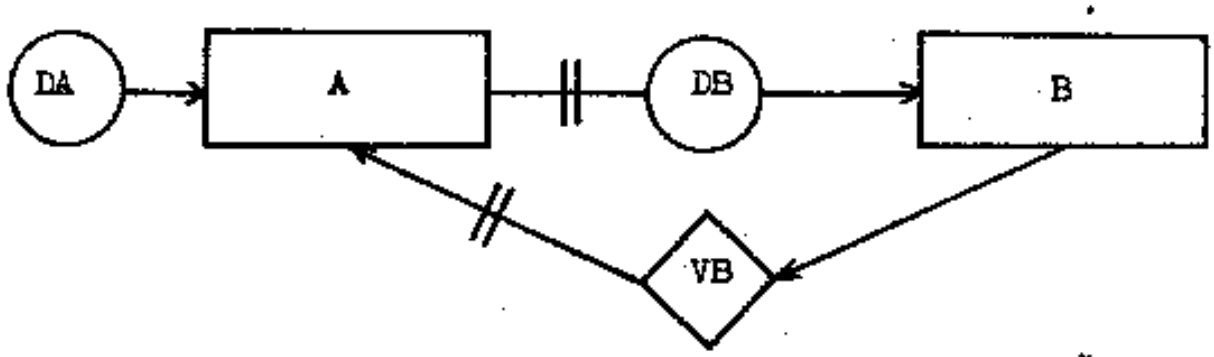
Obr. 5 Struktura dat předávaných procesem DODAVATEL



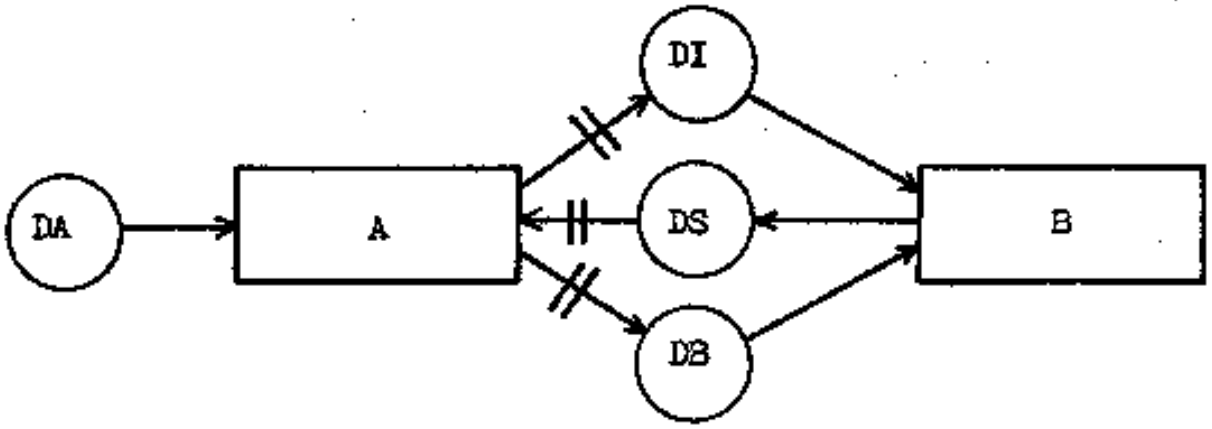
Obr. 6 SSD propojení procesů PARTNER, DODAVATEL a ZAKAZKA



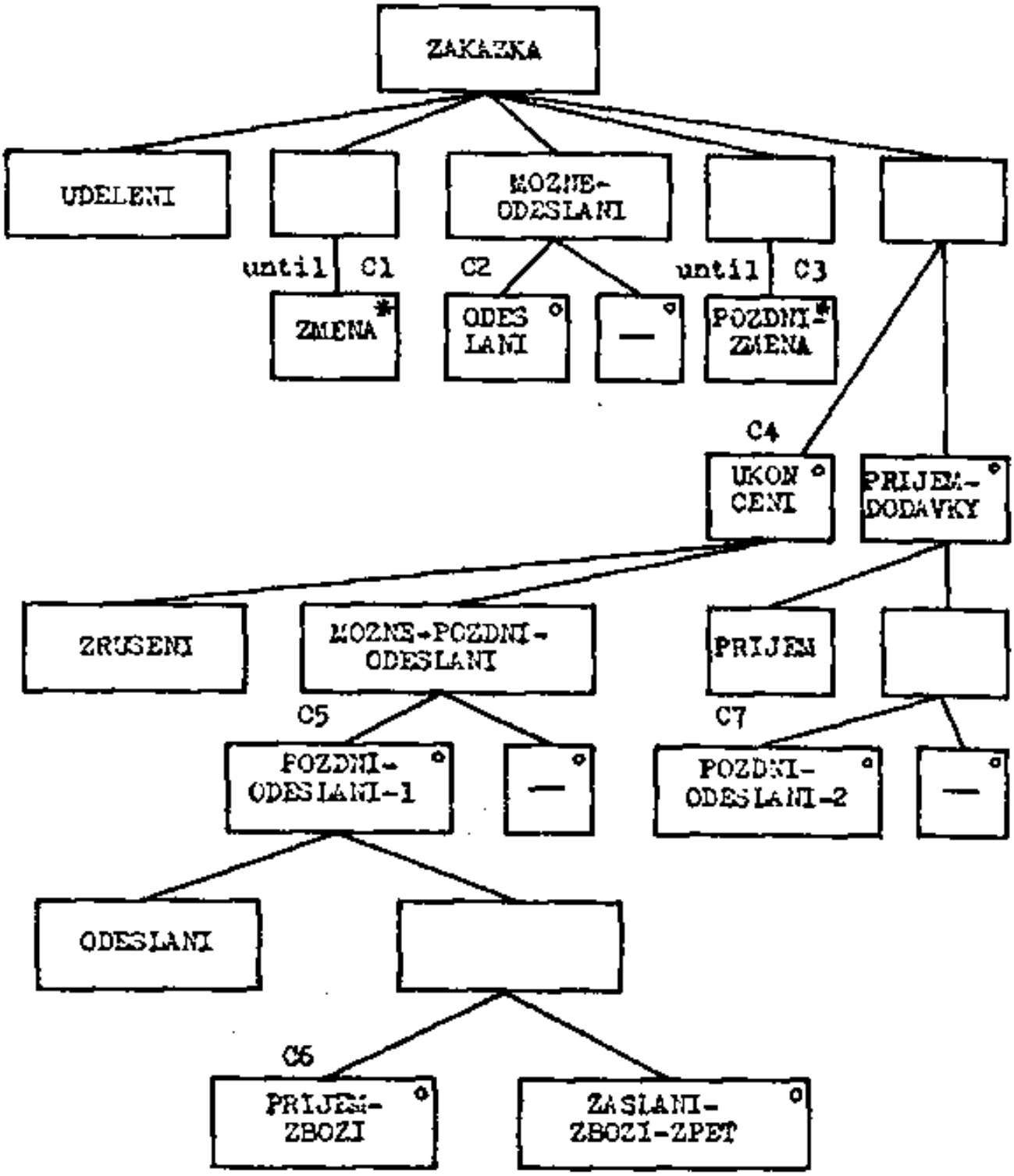
Obr. 7 Ideální struktura dat procesu ZAKAZKA



Obr. 9 SSD propojení procesů A a B, které nezajišťuje jejich vzájemnou synchronisaci



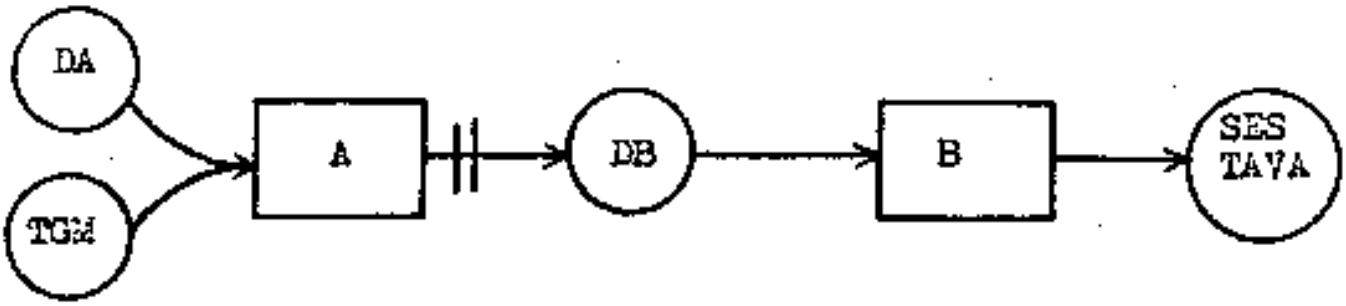
Obr. 10 SSD propojení procesů A a B umožňující vzájemné vyloučení provádění procesů A a B



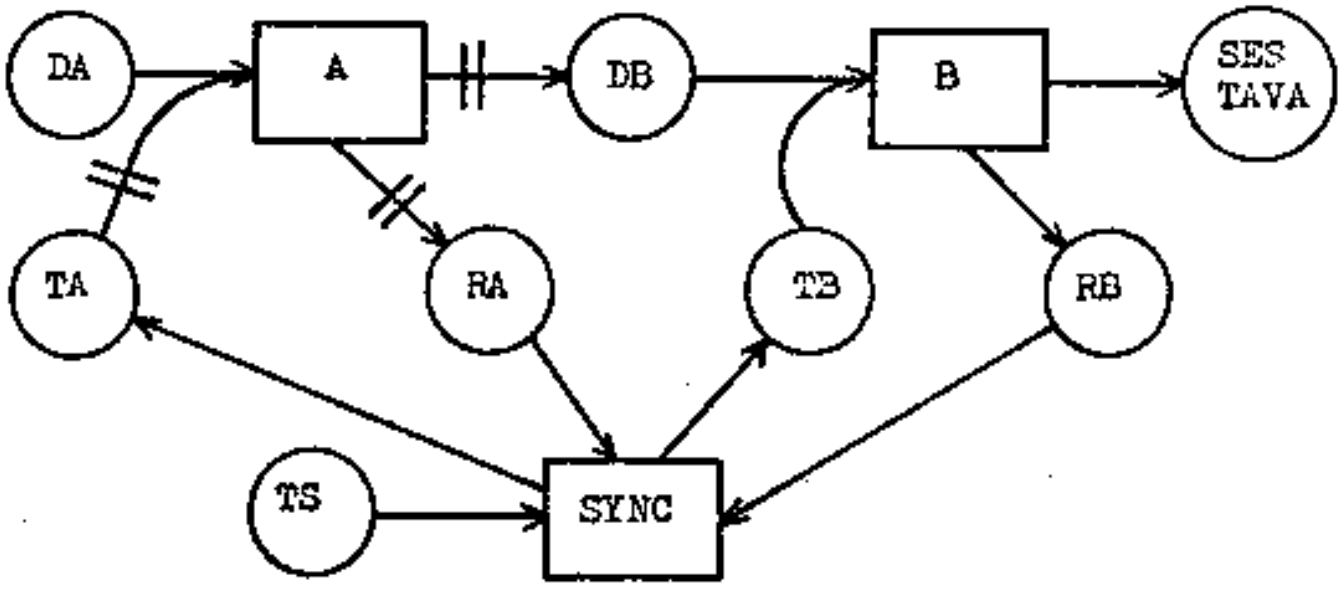
V podmínkách v selekcích a iteracích je uveden název akce, jejíž data jsou předávána procesu ZAKAZKA

C1 ... not ZMENA C4 ... ZRUSENI C7 ... ODESLANI
C2 ... ODESLANI C5 ... ODESLANI
C3 ... not ZMENA C6 ... PRIJEM

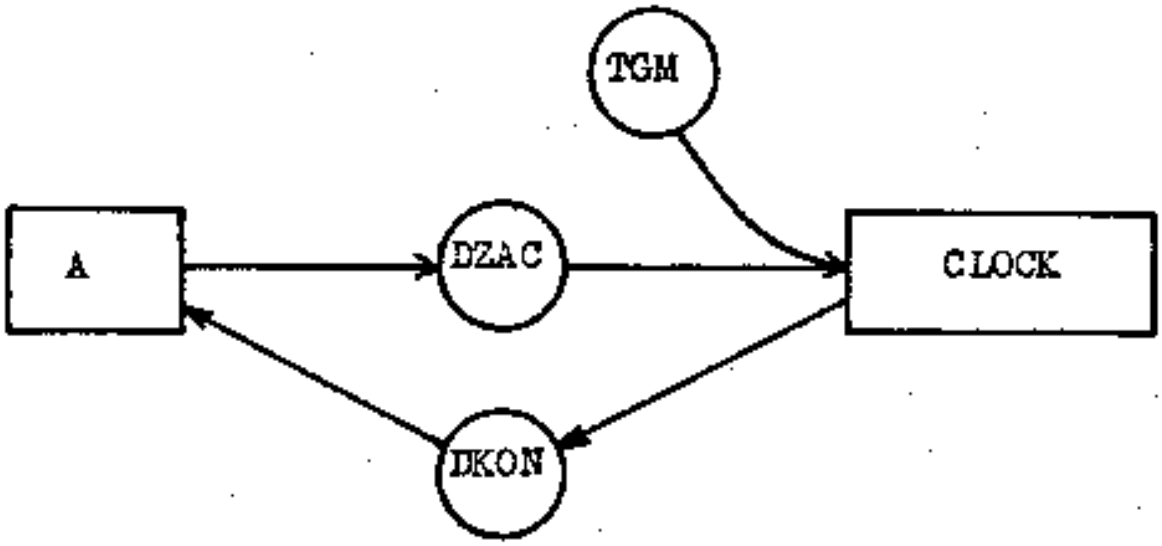
Obr. 8 Struktura dat procesu ZAKAZKA čtených hrubým míšením



Obr. 11 První verze SSD propojení procesů A a B



Obr. 12 SSD procesů A a B včetně synchronisačního procesu SYNC



Obr. 13 SSD propojení procesu A a CLOCK procesem