Z Á K L A D N Í T E C H N I K Y I M P L E M E N T A C E S Y S T É M U V J S D

Michal Kretschmer, prom. mat.

Tento příspěvek navazuje na článek [2] a příspěvek [3], které byly zaměřeny na výklad vypracování specifikace systému Jacksonovou metodou vývoje systému. Opírá se přitom o knihu [1] a zaměřuje se na základní postupy používané při implementaci vypracované specifikace systému, jež byly v článku [2] jen velmi stručně zmíněny. Jedná se zde o postupy specifické pro metodu JSD, které představují řešení dvou základních rozporů mezi předpoklady učiněnými v době vypracování specifikace systému a mezi podmínkami, ve kterých je třeba systém implementovat. Je to rozpor mezi počtem specifikovaných procesů a počtem disponibilních procesorů a rozpor mezi předpokládaným prováděním procesů po celou dobu existence příslušných entit a mezi potřebou tyto procesy přerušovat nebo je implementovat jako dávkové programy. Vzhledem k vymezenému rozsahu se příspěvek zaměřuje jen na základy hlavních technik implementace, jimiž jsou použití programové inverze, oddělení stavových vektorů od programu a rozčleňování procesů. Čtenáři se mohou jevit některé z těchto technik jako jednoduché a známé; nicméně cílem příspěvku je přesvědčivě ukázat, jak lze přejít od představy mnoha bez přerušení probíhajících procesů k tradičním dávkovým nebo interaktivním programům a tak specifikovaný systém implementovat.

1. Použití programové inverze

Při vypracování specifikace systému jsme předpokládali, že pro každý proces máme k dispozici samostatný procesor. Tento předpoklad je obvykle v rozporu se skutečností, a proto musíme usilovat o snížení počtu procesorů potřebných k implementaci systému. Jednou z možností je sdílet více procesů, které jsou propojeny pomocí dat, na jednom procesoru.

Nechť procesy P a Q jsou propojeny pomocí dat D zapisovaných procesem P. Přitom můžeme oba tyto procesy podřídit novému koordinačnímu procesu S, který zajistí střídavé provádění procesu P, dokud ten nezapíše jednu větu předávaných dat, a procesu Q, dokuď ten tuto předanou větu nezpracuje a nevyžaduje číst další větu dat D. Pseudokód procesu S pak bude :

```
S  itr
   CALL P(D-VETA);
   CALL Q(D-VETA);
S  end
```

Přitom je třeba kód procesů P a Q transformovat na invertovanou podobu vzhledem k předávaným datům D. Způsob provádění této transformace je popsán např. v [4]. Graficky znázorňuje tento způsob implementace procesů P a Q SID (System Implementation Diagram) na obr. 1.

Snadno zjistíme, že v tomto jednoduchém případě se můžeme obejít bez koordinačního procesu S. Jeho řídící úlohu může převzít proces P a pak Q bude invertovanou korutinou (SID viz obr. 2) nebo proces Q, kdy P bude invertovanou korutinou. V prvém případě provádění kombinovaného procesu značeného obvykle P‡Q začíná úvodními operacemi P a řízení je po prvé předáno procesu Q, když P dorazí k první operaci write; v druhém případě provádění začíná úvodními operacemi procesu Q a řízení je po prvé předáno procesu P, když Q dorazil k první operaci read. Tento rozdíl by mohl být v některých případech významný, neboť se vlastně jedná o chování systému během jeho startu, kdy některé procesy byly již spuštěny, zatímco jiné ještě nikoliv. Chceme-li se vyhnout z toho vznikajícím problémům, je vhodné dodržovat tato pravidla :

- každý proces přiřadí určitou hodnotu každému prvku svého stavového vektoru dříve než provádí nějakou operaci, kterou komunikuje s jinými procesy v systému (to nazýváme inicializační fáze procesu);
- vytváření procesu musí zahrnovat i provádění vytvořeného procesu až k první komunikaci s jiným procesem;
- stavový vektor procesu nesmí být prohlížen dokud není dokončeno vytvoření procesu včetně provedení jeho inicializační fáze.

Programové inverze lze použít i pro koordinaci více procesů, pro které platí :

- propojení mezi těmito procesy je pouze propojení pomocí dat (ne tedy pomocí stavového vektoru);
- v žádném z těchto procesů se nevyskytuje hrubé míšení;
- pokud existuje přímé nebo zprostředkované propojení mezi libovolnými dvěma z těchto procesů, existuje jen jedna cesta, po které jsou předávána data mezi nimi (tj. graf propojení všech těchto procesů neobsahuje cykl).

Jako řídící proces můžeme zvolit kterýkoliv z takto propojených procesů nebo můžeme je všechny podřídit nově vytvořenému koordinačnímu procesu. Přitom potřebujeme být schopni rozlišit důvod, pro který je provádění některého procesu dočasně přerušeno, tj. proces musí poskytovat informaci o tom, do kterého proudu dat zapsal nebo ze kterého si žádá data přečíst.

Jestliže proces P vytváří data předávaná procesu Q a zároveň data předávaná procesu R, proces Q data předávaná procesu R a proces R užívá hrubé míšení pro čtení obou proudů předávaných dat (SSD viz obr. 3), můžeme proces Q invertovat vůči datům předávaným mezi P a Q. Kombinovaný proces P&Q pak zapisuje v nějakém pořadí dva datové proudy čtené procesem R. Jestliže toto pořadí je přijatelné pro hrubé míšení v R, můžeme R invertovat vůči sloučenému datovému proudu B&D sestávajícímu z dat vytvářených procesy P a Q v pořadí, v jakém jsou tato data zapisována. Graficky tento způsob implementace vyjadřuje SID na obr. 4.

2. Oddělení stavových vektorů od programu

Při vypracování specifikace systému se zajímáme o jednotlivé typy entit a zpravidla abstrahujeme od počtu výskytů entit určitého typu. Každému výskytu entity odpovídá však samostatný proces. V oblasti hromadného zpracování dat máme obvykle více výskytů entity určitého typu (např. zákazníků, dodavatelů), tedy máme i více jim odpovídajících procesů, které mají všechny shodnou strukturu i pseudokód. Liší se pouze tím, že každý se týká jiného výskytu entity, jež se liší svým klíčem od ostatních entit téhož typu, a tím, že každá z těchto entit se může nacházet ve stavu rozdílném od ostatních výskytů entit tohoto typu. Shodný je tedy pseudokód těchto procesů, rozdílné jsou jejich stavové vektory.

Oddělíme proto stavové vektory jednotlivých procesů od textu programu, takže v paměti procesoru bude program přítomen jen jednou, ale stavových vektorů budeme mít právě tolik, kolik máme procesů. Obvykle nebudeme udržovat tyto stavové vektory ve vnitřní paměti procesoru, ale uložíme je do paměti s přímým přístupem tak, aby bylo možné je vyhledávat podle klíče, jímž je identifikace příslušného výskytu entity.

Vstupem do takového programu, který je společný pro všechny výskyty entity daného typu, je sjednocení všech vstupů pro jednotlivé výskyty entity. Pokud tento program bude pracovat v interaktivním režimu, předpokládáme, že data mohou vstupovat z více terminálů, a že máme k dispozici programové vybavení zabezpečující, že všechna data, bez ohledu na to, od kterého terminálu přicházejí, vstoupí do tohoto programu, a dále pak že všechny terminálové výstupy programů jsou směrovány k tomu terminálu, ze kterého vstoupila ta data, na která je výstupní zpráva odezvou.

Oddělení stavových vektorů od programu realizujeme tím, že vytvoříme koordinační program S (scheduler), který vyvolává program P, jenž je společný pro všechny výskyty entit typu P. Činnost koordinačního programu vyjadřuje následující pseudokód :

```
S  itr
   read vstup I;
   loadsv P(klíč of I) into předávací parametr SVP;
   CALL P(SVP);
   storesv SVP into P(klíč of I);
S  end
```

Koordinační program tedy čte všechny vstupy pro procesy odpovídající entitám určitého typu. Na základě identifikátoru výskytu entity určí, který proces má být aktivován, získá jeho stavový vektor (zpravidla čtením věty podle klíče), vyvolá společný program P se získaným stavovým vektorem jako parametrem a po návratu z programu P uloží pozměněný stavový vektor. Při tomto uspořádání stavový vektor musí obsahovat informaci, která určuje, kde při příštím vyvolání společného programu P pro tento proces má program P pokračovat. Program P tedy na začátku své činnosti při každém vyvolání obsahuje skok na pokračovací adresu odvozenou z obsahu předávacího parametru SVP. Jedná se o programovou inverzi známou z Jacksonova strukturovaného programování. Program P je tedy invertovanou korutinou koordinačního programu S.

Při kódování koordinačního programu S musíme se zabývat případem, kdy stavový vektor procesu P pro daný klíč není nalezen, protože příslušný proces ještě neexistuje. To nastává zpravidla při vstupu první akce entity P, kterou bývá vložení základních informací o tomto výskytu entity. Jedná se vlastně o vložení věty do logického souboru stavových vektorů procesů P. V takovém případě program S vloží do předávacího parametru SVP příznak udávající, že program P bude probíhat od svého začátku a zpracovávat tak akci založení výskytu entity. Zároveň si program S poznamená, že operaci storesv SVP musí realizovat jako vložení nové věty, nikoliv tedy jako změnu věty již existující.

Jinou variantou implementace oddělení stavových vektorů od programu je přesunout obhospodařování stavových vektorů jednotlivých procesů z koordinačního programu na začátek a konec společného programu, který na začátku své činnosti získá příslušný stavový vektor, z něho naplní své proměnné a pokračovací adresu, poté předá řízení na tuto pokračovací adresu a na závěr své činnosti uloží nové hodnoty svých proměnných a novou hodnotu pokračovací adresy do stavového vektoru a ten zapíše do vnější paměti. Tato druhá varianta bývá užívána v komunikačním režimu, kdy úlohu koordinačního programu přebírá výrobcem počítače dodávaný komunikační monitor.

Na základě výše uvedených principů lze navrhnout i koordinační program, jehož vstupem jsou data pro více procesů odpovídajících entitám různých typů. V tomto případě vstupní data musí obsahovat příznak, na základě kterého koordinační program rozhodne, kterému z odpovídajících společných programů má být předáno řízení. Stavový vektor je získáván na základě typu entity a jejího identifikátoru. K návrhu koordinačních programů je třeba přistupovat tvůrčím způsobem. Tak např. jestliže dva procesy jsou propojeny pomocí dat a jsou ve vztahu 1:1, lze jejich stavové vektory sloučit do jednoho a tento sjednocený stavový vektor ukládat až po provedení akce obou těchto procesů. Přitom koordinační program může pro jeden vstup buďto vyvolávat oba odpovídající programy po sobě nebo může první z těchto programů vyvolávat druhý z nich (SID viz obr. 3).

3. Rozčleňování procesů

Při implementaci systému se musíme obvykle také zabývat rozčleňováním procesů (process dismembering) do provádění schopných programů, které jsou běžným způsobem zaváděny do paměti a spouštěny. Při specifikaci systému jsme vycházeli z představy, že jednotlivé procesy probíhají (byť různou rychlostí a s čekáním na další vstup) nepřetržitě. Ve skutečnosti se však rozhodneme některé procesy implementovat jako dávkové programy prováděné na vyžádání nebo se stanovenou periodicitou, jiné procesy chceme sice implementovat jako on-line transakce, ale on-line systém nemáme trvale k dispozici, nýbrž zpravidla jen po část dne. Kromě toho můžeme se rozhodnout některé vstupy určitého procesu zpracovávat bezprostředně jako součást on-line systému, naproti tomu jiné, abychom nepřetěžovali počítač během dne, budeme zpracovávat dávkovým programem až po skončení pracovní doby.

Abychom tato rozhodnutí mohli realizovat, nemůžeme provádět celý proces najednou, takže musíme jej rozdělit do dvou nebo více částí. Tak proces, který bude implementován jako dávkový program, se rozpadá na dvě části : první z nich bude prostou iterací jednotlivých provádění druhé části. Schematicky to můžeme znázornit na následujících pseudokódech :

```
P   itr                 Pa  itr                 Pb  seq
    A   seq                     proveď Pb           .
        .               Pa  end                     .
        .                                           .
        .                                       Pb  end
    A   end
P   end
```

Vlevo je pseudokód původního navrženého nerozčleněného procesu. Uprostřed je pseudokód koordinačního programu Pa, který při každém průchodu iterací zabezpečuje jedno provedení programu Pb, jenž pseudokód je znázorněn vpravo. Původní pseudokód procesu P se tedy rozpadl na dvě části tak, že žádná jeho komponenta není ztracena, tj. každá jeho komponenta je obsažena v některé části (Pa nebo Pb). Při této implementaci budou vstupy procesu P čteny postupně při jednotlivých vyvoláních programu Pb, jsou tedy také rozčleněny. Toto rozčlenění vstupů implementujeme tím, že vstupní data uchováváme v sekvenčním souboru vždy od jednoho zpracování

Pb ke druhému. To se může dít buďto tak, že on-line program vstupní data místo jejich zpracování zapisuje do sekvenčního souboru nebo nejčastěji tak, že vstupní data jsou pořizována mimo počítač např. na magnetickou pásku, jež je pak vstupem pro každé provádění dávkového programu Pb. Rovněž tak i výstupy procesu P budou rozčleněny. Vytváří-li např. proces P tiskovou sestavu, bude při každém provádění programu Pb vznikat dílčí tisková sestava, kterou po skončení Pb lze vyjmout z tiskárny a dát k disposici uživateli.

Program Pa implementujeme často manuálně tím, že příslušný pracovník výpočetního střediska obdrží příkaz typu : Prováděj program Pb denně (týdně apod.) a dále pak dle něj skutečně také jedná. Dosud jsme předpokládali, že jednotlivá provádění programu Pb nemají spolu nic společného. Často tomu však tak není; např. sestava produkovaná Pb může obsahovat součet hodnot za všechny vstupy od počátku procesu P. V takovém případě je třeba na konci činnosti každého provádění programu Pb uložit jeho stavový vektor, jenž obsahuje tento součet, a na začátku každého Pb jej obnovit a dále pak zajistit, aby již před prvním vyvoláním Pb byl již stavový vektor vytvořen s počáteční hodnotou nula v tomto součtu. Stavový vektor je společný pro všechny části rozčleněného procesu P.

Schematicky tuto situaci znázorňuje SID na obr. 6. Úplné provedení Pb je znázorněno krátkou čarou kolmou ke spojnici mezi Pa a Pb. Nejedná se zde o to, že by Pb byl invertovanou korutinou Pa, ale o to, že ne každé vyvolání Pb je proveden celý Pb od začátku do konce. Jednotlivé části původního procesu P od sebe odlišujeme malými písmeny za názvem původního procesu. Stejně tak značíme rozčleněné části vstupů a výstupů procesu, takže např. Ib značí ty vstupy I původního procesu P, které jsou čteny při jednom vyvolání Pb. SVP značí stavový vektor procesu P, který Pb zapisuje na disk a z něj čte.

V případě, že zpracování části vstupů procesu P bude implementováno on-line a zpracování zbývajících vstupů jako dávkový program provozovaný po skončení provozu denního on-line systému, je situace poněkud jiná. Budeme mít dvě verze programu Pb a Pc, které nikdy nejsou současně v paměti procesoru, které však mají

společný stavový vektor. Z důvodu úspory paměti můžeme z Pb vyjmout výkonné instrukce týkající se zpracování dávkových vstupů a naopak z Pc vyjmout výkonné instrukce pro zpracování on-line vstupů. Příslušná návěští však ponecháme; to je bezpodmínečně nutné u návěští zavedených z důvodu inverze procesu P. Podmínky v selekcích a příp. i iteracích můžeme v rámci optimalizace Pb a Pc zjednodušit, příp. i některé selekce nahradit sekvencí, jestliže totiž v nich byla volba mezi jedním typem dávkového vstupu a jedním typem on-line vstupu. Údržbu programu je však vhodné provádět v nerozčleněné verzi jeho zdrojového textu a transformace vedoucí k jeho rozčlenění je pak třeba po každé opakovat.

Tento způsob implementace znázorňuje SID na obr. 7. Přitom Ib jsou on-line vstupy pro on-line režim procesu P během jednoho dne, Ic dávkové vstupy pro Pc pro tento den. Ob jsou výstupy Pb zapisované na terminál, Oc tiskový protokol provedených změn v dávkovém režimu při jednom běhu Pc. Pořadí provádění rozčleněných částí Pb a Pc je definováno pseudokódem Pa, který v našem případě bude obsahovat v iteraci vždy sekvenci kompletního provedení jednoho Pb a jednoho Pc.

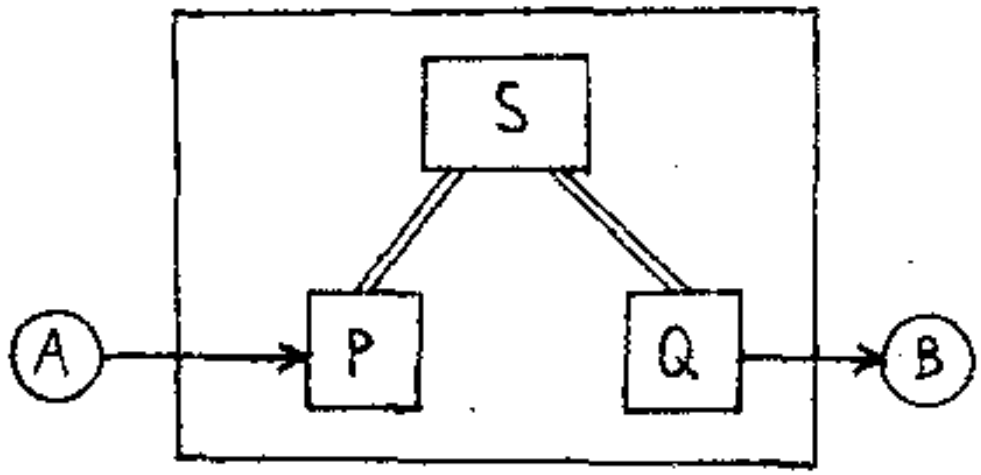
Literatura :

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

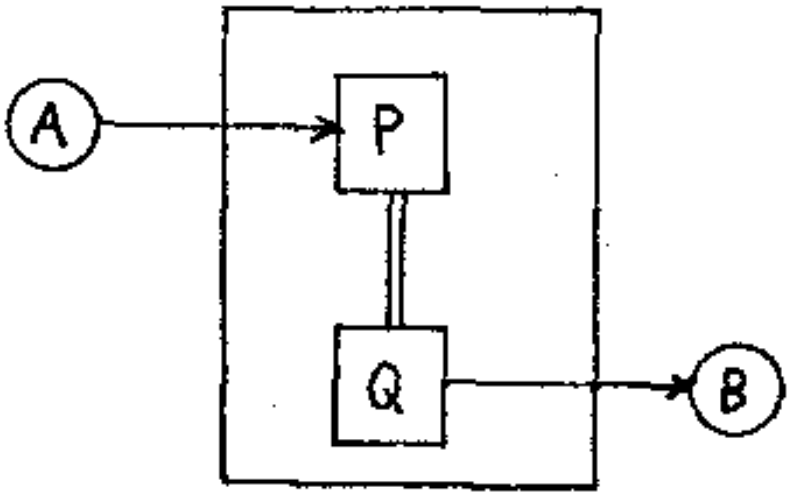
[2] Kretschmer M.: Jacksonova metoda vývoje systému, MAA, 1986, č. 3-5

[3] Kretschmer M.: Poznámky k propojování a synchronisaci procesů v JSD, sborník Programování '86, DT ČSVTS Ostrava

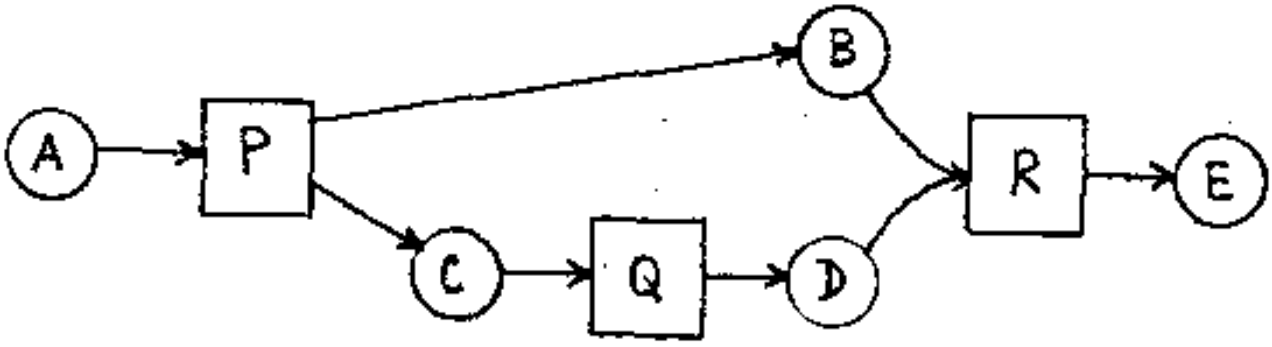
[4] Vondráček B. a kol.: Technologie strukturovaného programování, sborník Programování '80, DT ČSVTS Ostrava



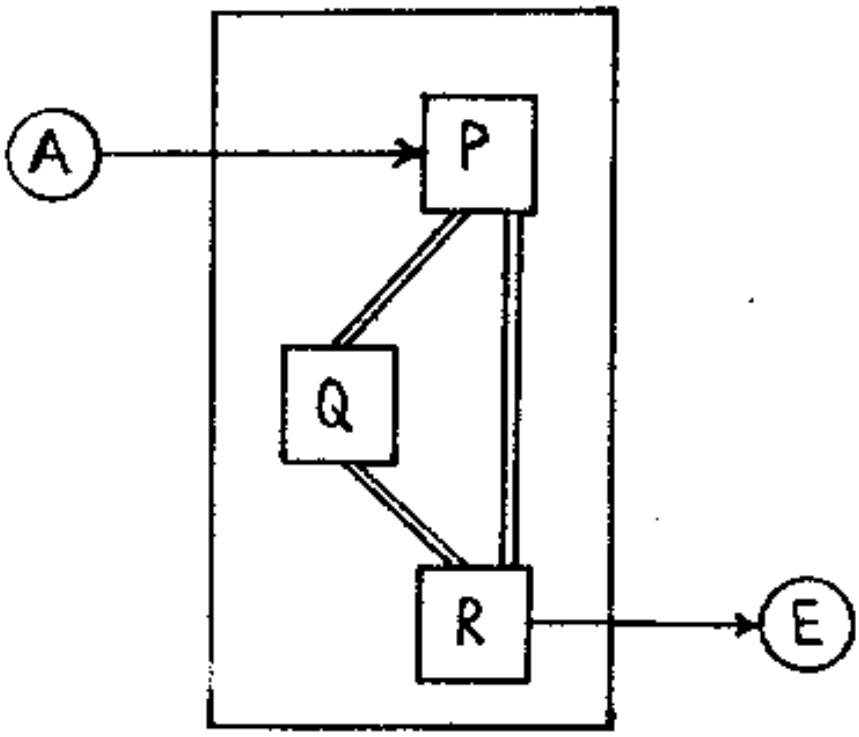
Obr. 1 SID případu, kdy korutiny P a Q jsou podřízeny koordinačnímu programu S



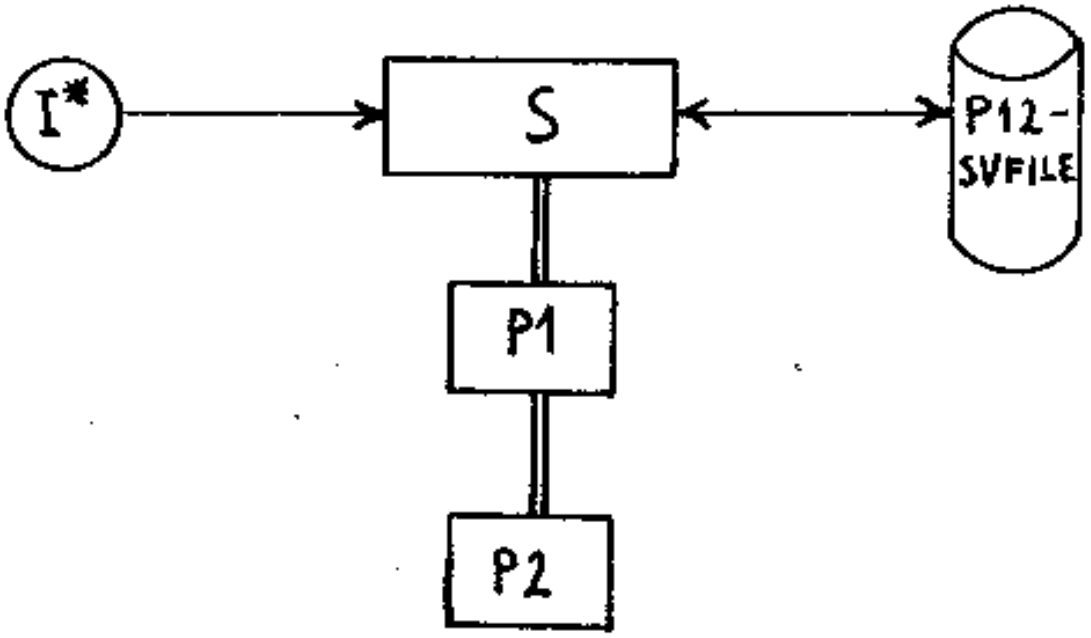
Obr. 2 SID případu, kdy Q je invertovanou korutinou procesu P



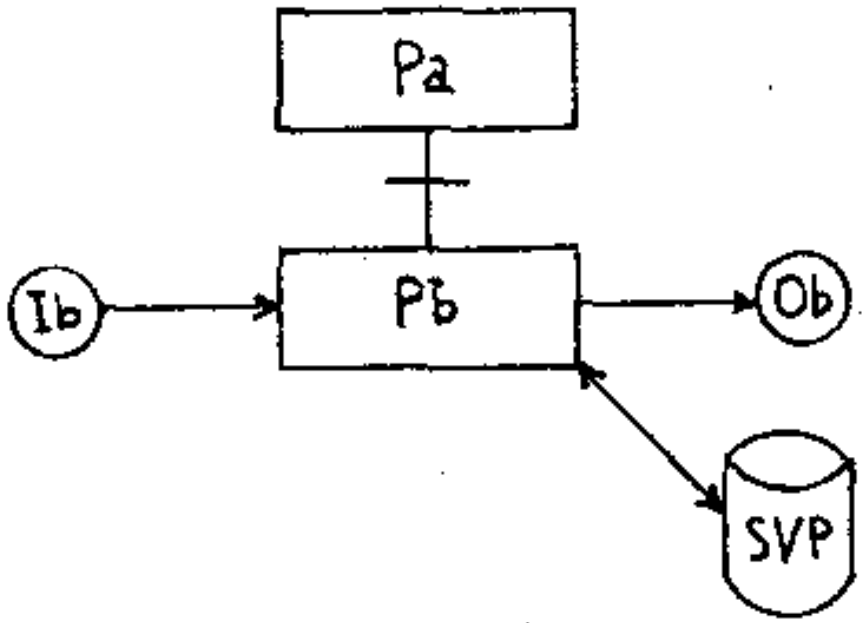
Obr. 3 SSD případu, jehož implementace je znázorněna na obr. 4



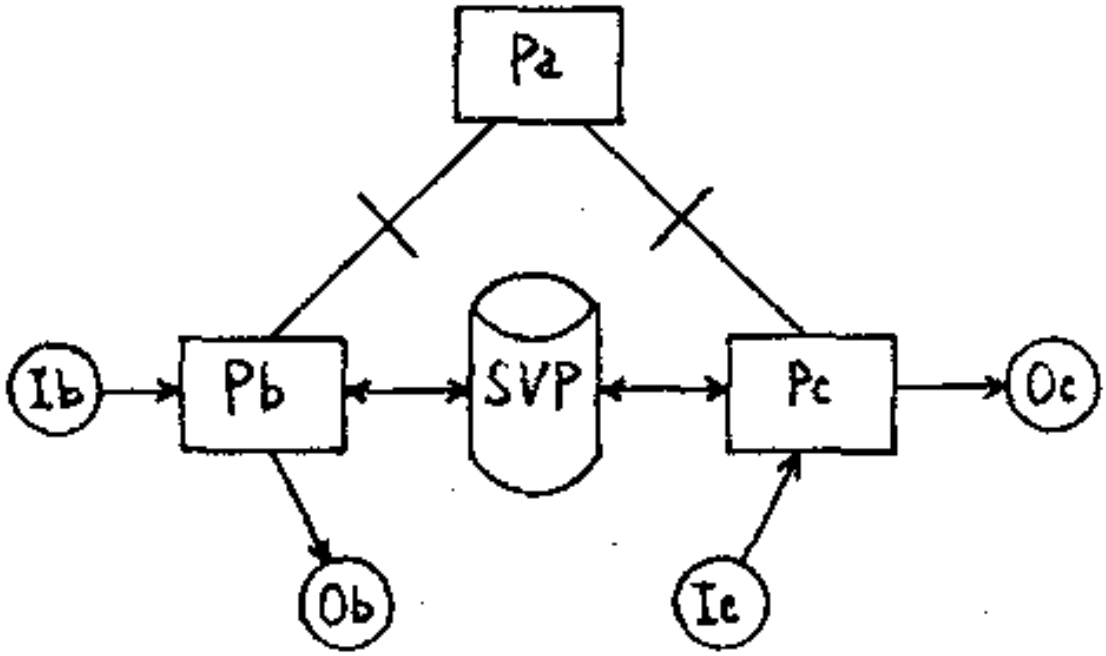
Obr. 4 SID případu, kdy Q je invertovanou korutinou procesu P a R je invertovanou korutinou kombinovaného procesu P&Q



Obr. 5 SID případu, kdy P1 je invertovanou korutinou S a P2 invertovanou korutinou P1, přičemž čtení a zápis stavových vektorů kombinovaného procesu P1&P2 zabezpečuje koordinační program S



Obr. 6 SID případu, kdy proces P je rozčleněn na programy Pa a Pb, přičemž Pa zabezpečuje opakované provádění Pb



Obr. 7 SID případu, kdy proces P je rozčleněn na programy Pa, Pb a Pc, přičemž Pa zabezpečuje opakované provádění Pb a Pc