Michal Kretschmer, prom. mat.
Ing. Bohumil Vondráček
MÚZO Praha

APLIKACE TECHNOLOGIE STRUKTUROVANÉHO PROGRAMOVÁNÍ PRO ŘEŠENÍ ON-LINE PROBLÉMU

1. Úvod

V současné době roste podíl on-line úloh zaváděných na počítačových systémech. Tento článek navazuje na příspěvek "Technologie strukturovaného programování" uveřejněný v loňském sborníku Programování '80 a v dalším je znalost základních principů této technologie předpokládána.

Cílem našeho příspěvku je ukázat, jak navrhovat struktury programů pro on-line zpracování. Ukazuje se, že návrh těchto programů se prakticky neliší od návrhu programů pro dávkové zpracování, i když zde existují jistá omezení. Příspěvek se též zabývá klasifikací on-line úloh, diskutuje vztahy mezi operačním systémem a on-line programem uživatele a v závěru obsahuje jednoduchý příklad.

2. Klasifikace on-line úloh

On-line úlohou rozumíme takové zpracování, při němž dochází k přenosu dat (zpráv) mezi hlavním počítačem a koncovým zařízením v obou směrech. Data od koncového zařízení mohou být odesílána buďto manuálně operátorem (zpravidla prostřednictvím klávesnice) nebo automaticky. Pokud by docházelo např. jen k výstupu dat bez interakce, jednalo by se vlastně o zpracování dávky, která by byla "zapisována" na koncové zařízení. Jakmile však koncové zařízení odpovídá uživatelskému programu

informací o úspěšnosti přenosu, jedná se již o interakci. Tyto úlohy lze programově realizovat na hlavním počítači; je-li však koncové zařízení programovatelné, může být další podpůrný program přítomen též v paměti tohoto zařízení.

On-line úlohy můžeme rozdělit do dvou základních kategorií : dialog mezi programem a operátorem koncové stanice a přenos dávek dat. V některých případech může jít o kombinaci úloh obou těchto kategorií řešených jediným programem.

Pod pojmem dialogové zpracování rozumíme takové zpracování, při kterém program přijme zprávu operátora koncového zařízení, zpracuje ji a jako odpověď odešle ke koncovému zařízení jednu nebo eventuálně více zpráv, přičemž až do vyslání poslední z těchto zpráv program nepřijímá od koncového zařízení další zprávy. Program není tedy během zpracování zprávy operátora přerušen. Jedná se vlastně o jakýsi dotaz, na který program odpovídá, přičemž je vždy aktivován zprávou operátora. Odpovědi jsou obvykle posílány na obrazovku terminálu.

Přenos dávek dat může být aktivován jak operátorem koncového zařízení, tak bez jeho součinnosti. Jedná se buďto o zápis dávky dat na některou z periferií koncového zařízení (tiskárna, disketta apod.) nebo o čtení dat z periferie koncového zařízení a jejich předání programu hlavního počítače k dalšímu zpracování. Koncové zařízení přitom automaticky vysílá zprávy o úspěšnosti prováděných periferních operací, které dostává uživatelský program hlavního počítače k dispozici ve formě speciální vstupní zprávy. Celá dávka dat je postupně vysílána resp. přijímána bez zásahu operátora koncového zařízení (kromě eventuálního prvotního pokynu k zahájení přenosu dávky), přičemž operátor koncového zařízení má zpravidla možnost během tohoto přenosu odesílat další zprávy směrem k programu v hlavním počítači. Tyto zprávy mohou např. požadovat ukončení přenosové funkce, informovat operátora hlavního počítače apod. Program v hlavním počítači musí tedy kromě provádění přenosu dávky dat zpracovávat tyto zprávy operátora koncového zařízení.

3. Vlastnosti operačních systémů s ohledem na řešení on-line úloh

Veškeré přenosy informací mezi programem a koncovým zařízením, právě tak jako mezi programem a libovolnou periferií hlavního počítače, jsou prováděny v součinnosti s operačním systémem. On-line program tedy žádá operační systém o provedení přenosu ke koncovému zařízení resp. žádá operační systém, aby mu předal přenesenou zprávu, je-li tato k dispozici nebo jej alespoň informoval, že zatím žádná zpráva od koncového zařízení nedošla.

Každé koncové zařízení má vlastní tzv. komunikační protokol, tj. souhrn pravidel, jak musí vypadat formát zprávy, řídící znaky, jak se potvrzuje přijetí zprávy apod. Operační systém na různých úrovních zajišťuje disciplinované zachovávání pravidel komunikačního protokolu, tj. např. doplňuje automaticky požadované řídící znaky do zprávy a naopak částečně dekóduje zprávy došlé od koncových zařízení. Rovněž může zajišťovat nezávislost programu uživatele na použitém typu koncového zařízení, tj. zprávy programu trasformuje na formát požadovaný skutečně použitým typem koncového zařízení (obvykle deklarovaným při definici sítě). Operační systém též zjišťuje chyby při přenosu a automaticky zajišťuje opravné akce, takže program uživatele se o těchto chybách nedozví buďto vůbec nebo se o takovéto chybě dozví jako o neodstranitelné chybě po marných pokusech operačního systému o její odstranění.

K hlavnímu počítači je zpravidla připojeno více koncových zařízení, které tvoří tzv. síť. Uživatelský program má přístup ke konkrétní podmnožině této sítě. Může tedy obsluhovat dvě i více techto koncových zařízení. Každá zpráva je operačním systémem vybavena jakousi adresou zařízení, k němuž směřuje nebo od kterého přichází. V paměti hlavního počítače může být v multiprogramovém prostředí současně přítomno několik on-line programů, přičemž distribuci zpráv pro příslušný program zajišťuje

operační systém, což se děje právě na základě adresy.

Program může předávat operačnímu systému zprávy rychleji než je možné vzhledem k přenosové rychlosti linky odeslat. Rovněž tak může dojít k přijetí dalších zpráv od koncového zařízení, aniž by předchozí zpráva byla již programem uživatele zpracována. Z těchto důvodů bývají operační systémy vybaveny možností ukládat vysílané a přijímané zprávy do front. Operační systém předává přijatou zprávu uživatelskému programu až v okamžiku, kdy ten o předání zprávy požádá. Tyto činnosti operačních systémů nemají vliv na návrh struktury uživatelského programu.

On-line program může obsluhovat více koncových zařízení, přičemž může být požadováno provádět pro různá koncová zařízení shodné funkce. Nebylo by samozřejmě dobré, abychom měli pro každé koncové zařízení v paměti kopii celého uživatelského programu. Procedurální část programu lze mít v paměti pouze jednou, data však v tolika oblastech, kolik je koncových zařízení, které program obsluhuje. Data a stavové informace jsou tedy pro každé aktivní koncové zařízení deklarována zvlášť. Oddělením dat od programu je sledováno zajištění reentrantnosti programu, tj. program může být aktivován současně pro více koncových zařízení (multi-threading). Aktivace programu se však musí dít prostřednictvím nadřízené komponenty, kterou může být buďto operační systém nebo uživatelský distribuční program. Nelze-li pomocí oddělení datové a procedurální části programu dosáhnout vlastnost reentrantnosti, měl by program být alespoň následně znovupoužitelný (serially reuseable). Toto řešení také předpokládá existenci distribučního programu, který však nemůže aktivovat právě aktivní program v paměti počítače před jeho dokončením. Distribuční program musí buďto čekat nebo vytvořit v paměti další kopii uživatelského programu, který bude pracovat ve prospěch dalšího koncového zařízení. Při řešení on-line problémů by měla být podle našeho názoru dávána přednost tvorbě reentrant. programů.

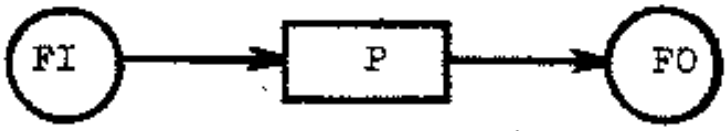
Distribuční program při své činnosti vychází ze souboru stavových vektorů koncových zařízení, jejichž součástí jsou oblasti pro vstupní a výstupní zprávy, pracovní oblasti, v nichž jsou deklarovány lokální proměnná data probíhajícího procesu, a informace o místě pokračování přerušeného procesu. Úkolem distribučního programu je při zahájení uživatelského programu ve prospěch konkrétního koncového zařízení zřídit a inicializovat oblast pro stavový vektor a při každém znovuaktivování již zahájeného programu pro koncové zařízení získat jeho stavový vektor a tento jako parametr předat uživatelskému programu. Uživatelský program musí pak na základě hodnot stavového vektoru aktivovat zpracovávaný proces ve správném místě a před odevzdáním řízení distribučnímu programu předat novou hodnotu stavového vektoru, ve kterém je informace o místě pokračování procesu při další aktivaci uživatelského programu a zpravidla je též ve stavovém vektoru předána zpráva určená k odeslání ke koncovému zařízení.

Máme-li k dispozici dobrý distribuční program, nemusí se uživatelský on-line program zabývat případem, kdy čeká na vstupní zprávu a tato nepřichází, tj. získává informaci od operačního systému, že zpráva není k dispozici. Tento stav je řešen na úrovni distribučního programu a vlastní uživatelský program je distribučním programem aktivován jenom tehdy, je-li k dispozici vstupní zpráva, kterou mu současně prostřednictvím stavového vektoru předává.

Je-li tedy k dispozici distribuční program výše uvedených vlastností, může být uživatelský program koncipován tak, jakoby pracoval s jediným koncovým zařízením, tj. provádí jediný proces. V dalším se budeme zabývat problémem návrhu vlastních uživatelských programů a budeme předpokládat, že distribuční program uvedených vlastností existuje. Důvodem pro to je též, že většina operačních systémů je vybavena komponentami, které svými vlastnostmi odpovídají našemu pojetí distribučního programu.

4. Návrh uživatelského on-line programu

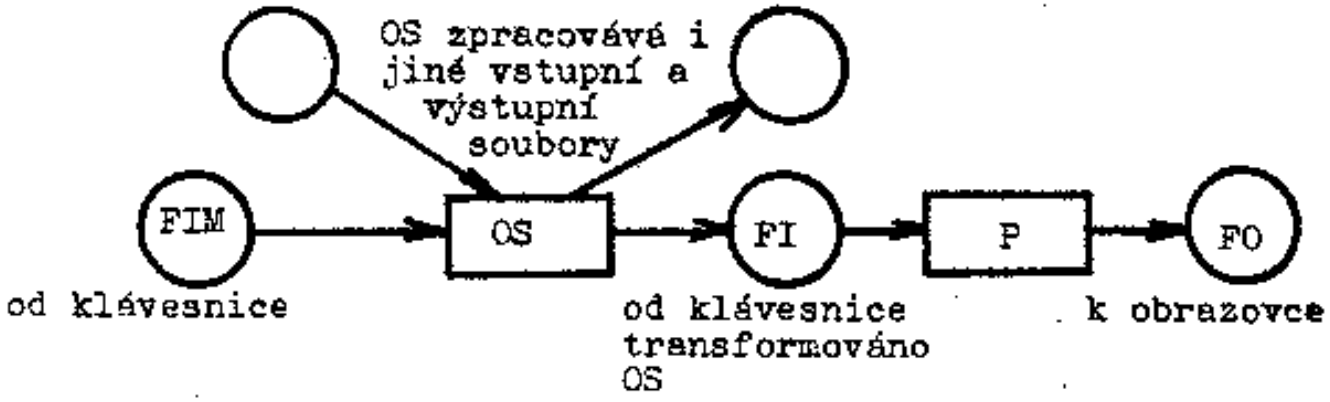
Úkolem je navrhnout uživatelský on-line program P, jenž podporuje jediné koncové zařízení a pracuje v následujícím prostředí:



od klávesnice k obrazovce

kde FI je soubor zpráv přicházejících od koncového zařízení, FO je soubor zpráv vydávaných procesem P k témuž koncovému zařízení. Proces P může samozřejmě používat další soubory nacházející se na médiích hlavního počítače.

Uživatelský program P nepracuje však samostatně, nýbrž pracuje v jistém prostředí speciálně pod operačním systémem a distribučním programem. Dále nebudeme tyto dvě komponenty oddělovat a budeme je vždy nazývat operačním systémem. Tedy proces P pracuje v následujícím prostředí :

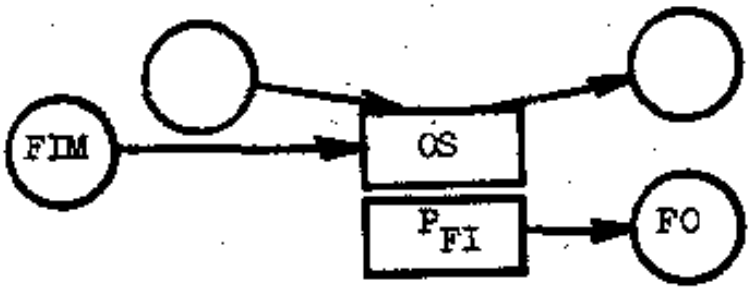


kde FIM je soubor zpráv ze všech koncových stanic v netransformovaném tvaru tak jak jsou dodány na základě komunikačního protokolu. FI je podmnožina těchto zpráv a sice obsahuje ty zprávy, které jsou určeny pro zpracování procesem P, a které jsou navíc již upraveny operačním systémem OS. Operační systém provádí tedy selekci zpráv pro další zpracování procesem P na základě

jejich adresy a někdy i obsahu, tj. nepředává zprávy od koncového zařízení zpracovávané pouze komponentou, která zajišťuje dodržení disciplíny dané komunikačním protokolem.

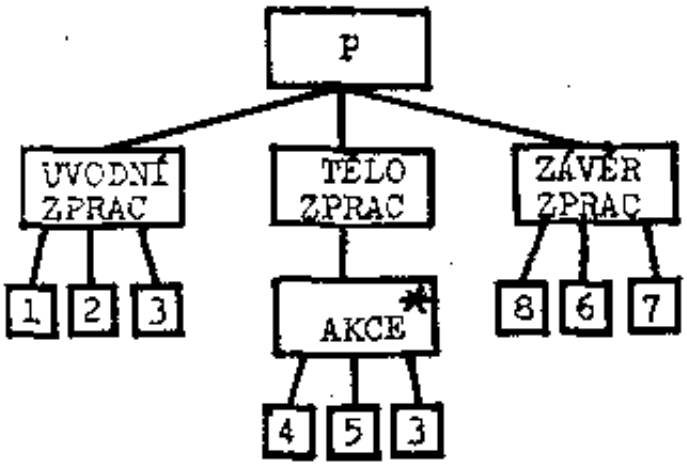
Uvažujme nyní návrh programu P pracujícím ve výše popsaném prostředí z hlediska programátora, který aplikuje principy technologie strukturovaného programování. Prvním krokem této technologie návrhu je popsat strukturním diagramem strukturu všech sekvenčně zpracovávaných souborů. Z tohoto hlediska jsou soubor FI vstupních zpráv určených pro proces P a soubor FO výstupních zpráv určených k odeslání ke koncovému zařízení sekvenční soubory, a tudíž ovlivňují výslednou programovou strukturu. Struktury techto souborů lze popsat strukturními diagramy s použitím základních řídících struktur sekvence, selekce a iterace a spolu se strukturami ostatních zpracovávaných souborů slouží k odvození programové struktury. V podstatě není třeba vidět rozdíl mezi souborem zpráv vztahujících se ke koncovému zařízení a klasickým sekvenčním souborem na magnetické pásce nebo disku.

Z hlediska implementace on-line programu však existuje následující rozdíl. Chceme totiž využít hlavní počítač pro jiné programy, zatímco náš uživatelský program čeká např. na vstup operátora koncového zařízení, který mu dodá další vstupní zprávu. Jestliže si uživatelský program přeje pokračovat ve své činnosti až po obdržení další zprávy, odevzdá řízení operačnímu systému. To však znamená, že program P je vlastně invertovaným programem vzhledem k souboru FI a je realizován jako podprogram operačního systému, který mu dodává data souboru FI a jenž současně odevzdává data i pro jiné programy. Invertovaný program P_{FI} tedy pracuje v následujícím prostředí :



kde P_{FI} je P invertovaný vzhledem k mezisouboru FI

Ve zjednodušeném případě, tj. pro každou vstupní zprávu je požadováno stejné zpracování, jehož součástí je požadavek na odeslání jedné výstupní zprávy, může vypadat programová struktura programu P takto :



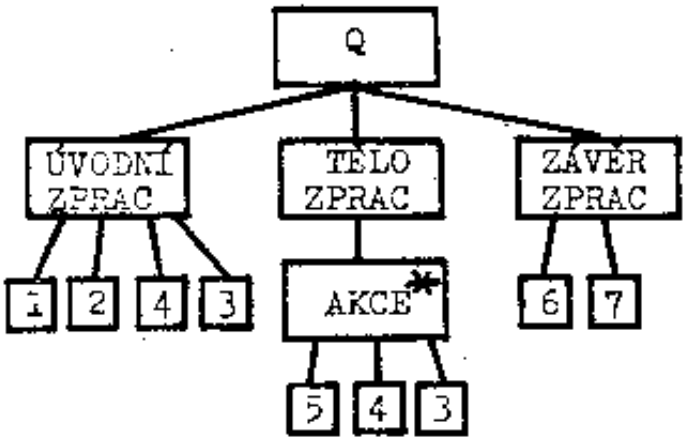
Seznam operací :

1. open FI
2. open FO
3. read FI
4. zpracování věty FI
5. write FO
6. close FI
7. close FO
8. write koncovou větu FO

Pokud by v rámci operace 5 bylo odesíláno ke koncovému zařízení více zpráv, lze na ně pohlížet jako na jedinou, která vznikne složením všech předchozích požadavků na zápis do souboru FO. Proto lze eventuální zpracování mezi operacemi 5 a 3 provést již v rámci operace 4 a tím tedy posunout operaci 5 bezprostředně před operaci 3. Tímto způsobem se v rámci těla zpracování (které obecně má členitější strukturu) vytvoří vždy dvojice operací "write FO" a "read FI", které za sebou bezprostředně v tomto pořadí následují. Operace 2 a 7 představují připojení resp. odpojení koncového zařízení od programu a jsou realizovány jako funkce operačního systému.

Výše uvedený program P byl koncipován jako program hlavního počítače, který pracuje jako korutina operačního systému a který je aktivován vždy při příchodu vstupní zprávy z koncového zařízení. Některá koncová zařízení jsou však programovatelná a tedy vstupní zprávy pro program v hlavním počítači mohou být vytvářeny programem (místo ručního vstupu operátora). V tomto případě vzniká úloha jak navrhnout program pro toto koncové zařízení, jenž bude spolupracovat s programem v hlavním počítači. Program Q koncového zařízení pracuje tak, že sám začíná svou

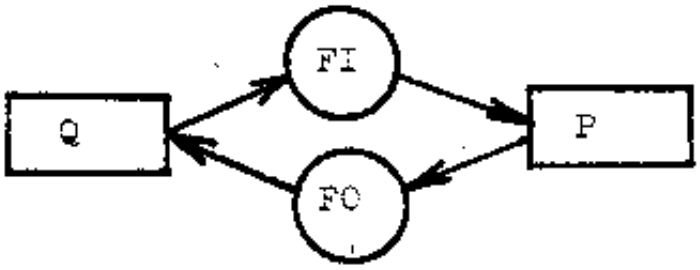
aktivitu, vyšle první zprávu, což lze chápat jako dotaz a vždy pak čeká na zprávu hlavního počítače (odpověď) a pak teprve vyšle další zprávu. V tomto případě by mohla zjednodušená programová struktura programu Q vypadat takto:



Seznam operací :

1. open FO
2. open FI
3. read FO
4. write FI
5. zpracování věty FO
6. close FO
7. close FI

Vztah mezi programy P a Q lze vyjádřit následovně :



Program P se [redacted]
jako "přijímač",
program Q jako
"tazatel"

5. Omezení při návrhu on-line programu

Pro on-line problémy dialogového typu je charakteristické, že uživatel provádí konversaci se systémem. Co je tím míněno ? Znamená to, že každá vstupní zpráva (tj. dotaz) je zodpovězena programem dříve (tj. zapsána na koncové zařízení - odpověď), než může být přijata a zpracována další vstupní zpráva. Tomuto uspořádání odpovídá i výše uvedená programová struktura programu P. Lze tedy říci, že n-tá odpověď musí být zapsána dříve než je kladena (n+1)-tá otázka. To není nic překvapujícího, neboť takovým způsobem jsme zamýšleli program užívat.

Z výše uvedeného vyplývají některé závěry a omezení týkající

se návrhu programové struktury :

a) Spojování otázky a odpovědi vynucuje korespondenci 1:1 mezi každou elementární komponentou vstupní datové struktury a výstupní datové struktury. Vstupní a výstupní soubory zpráv mají tedy přesně shodnou strukturu (liší se pouze formátem jednotlivých zpráv). Proto postačuje znázornit pouze strukturní diagram pro soubor vstupních zpráv.

b) Technika vícenásobného čtení napřed nemůže být aplikována, neboť po n-tém dotazu nemůžeme získat (n+1)-tý dotaz aniž bychom n-tý dotaz zodpověděli.

c) Vstupní zprávy dodává člověk u koncového zařízení. Nelze tedy připustit, aby vedlejší efekty při použití techniky zpětného sledování (backtracking) týkající se souborů vstupních a výstupních zpráv byly nepřijatelné. Lidskou činnost nelze mechanicky vrátit zpět a požadovat např. vstup těchže zpráv znovu je nepřijatelné. Výstupní zprávy, které by byly jako nevýhodný efekt zpětného sledování vyslány ke koncovému zařízení, by jeho obsluhu jenom mátly. Tudíž vedlejší efekty zpětného sledování, které se vztahují k těmto souborům, musí být vždy klasifikovány jako výhodné. Není-li tomu tak, je problém neřešitelný.

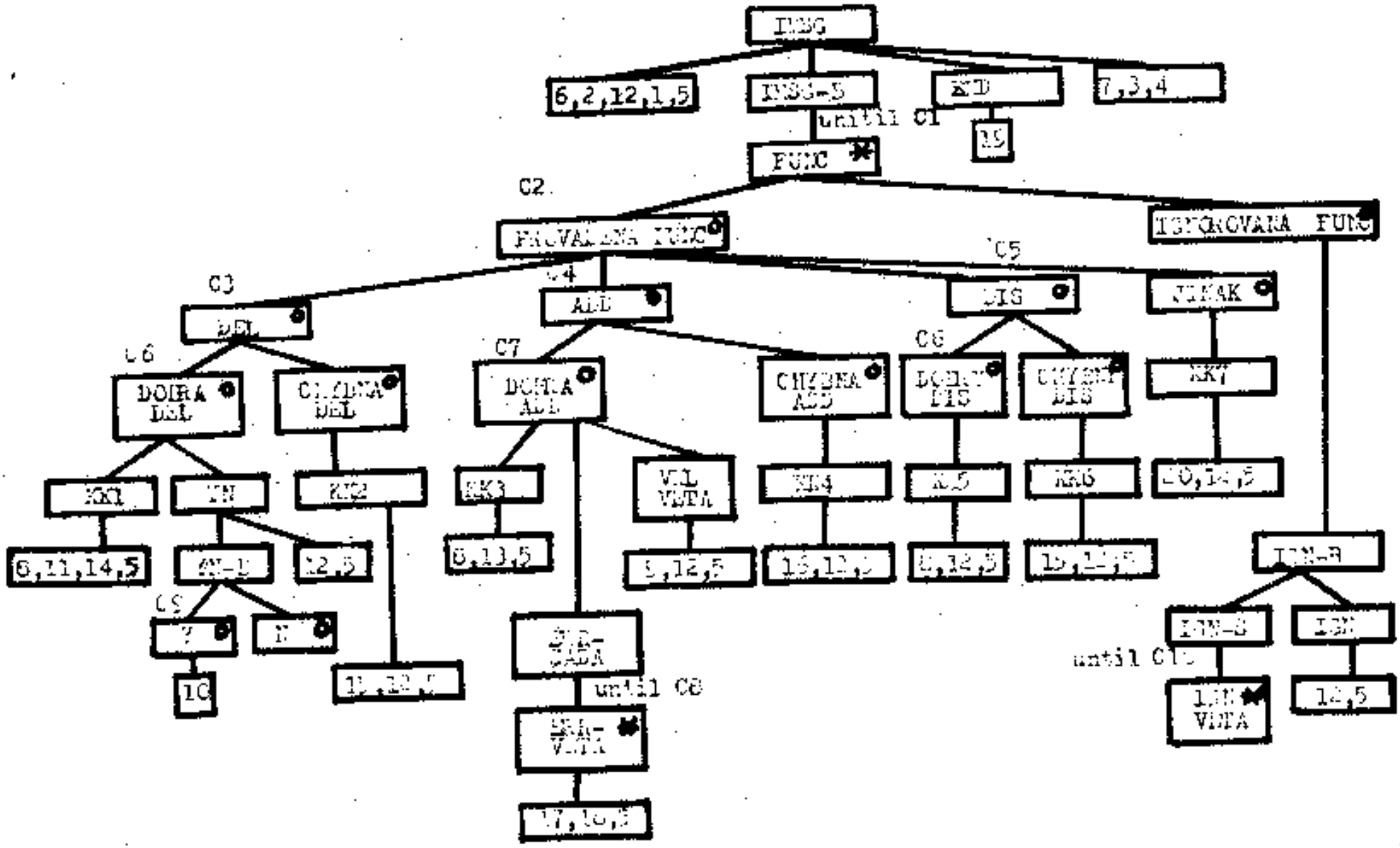
On-line programy pro přenos dat lze navrhovat stejným způsobem, pokud se omezíme na to, že vždy po odeslání datové věty koncovému zařízení čekáme na odpověď koncového zařízení potvrzující příjem této věty. Pak teprve odesíláme další větu. Pokud však operační systém umožňuje vyslat další datovou větu bez potvrzení příjmu předcházející (např. tím, že umožňuje ukládat zprávy do front), nejsou jednotlivé komponenty datových struktur souborů vstupních a výstupních zpráv v korespondenci, operace "read" a "write" se nevyskytují v programu v párech. Sjednocení datových struktur není možné z důvodu rozporu proložení (předpokládá se, že přijetí každé datové věty je někdy později koncovým zařízením potvrzeno). Problém lze řešit zavedením mezisouboru mezi souběžně pracujícími procesy pro vstup a výstup zpráv.

6. Příklad

Máme vytvořit programovou strukturu k programu, který provádí změnové řízení diskového souboru s přímým přístupem dle klíče. Z obrazovkového terminálu jsou požadovány funkce výmaz (DEL), vložení (ADD), opis (DIS). Opravu diskové věty jsme v našem příkladě vypustili z důvodu podobnosti s funkcí vložení. Obsluha terminálu zapisuje svůj požadavek ve formě 3-znakové zkratky funkce a klíče diskové věty. Ukončení programu zadá operátor kódem END. Při výmazu je disková věta o příslušném klíči nejprve zobrazena na terminálu a operátor je dotázán, zda si přeje výmaz skutečně provést. Při požadavku na vložení věty je na obrazovce nejprve zobrazena prázdná maskaodpovídající vstupnímu formátu věty, kterou operátor vyplní údaji a odešle. Program provádí kontrolu správnosti vyplněných polí, v případě chyb vyšle zprávu k terminálu obsahující tuto větu s označenými chybami, bezchybnou větu zapíše do diskového souboru. Program dále testuje takové chyby, jako je pokus o výmaz neexistující věty nebo pokus o vložení věty s již existujícím klíčem. Během zpracování funkce je možné další zpracování funkce ignorovat vysláním zprávy s kódem IGN. V dalším předpokládáme, že protokolaci změn souboru pro jeho případnou obnovu provádí operační systém.

Tento příklad představuje úlohu dialogového typu se dvěma sekvenčně zpracovávanými soubory : vstupní soubor zpráv a výstupní soubor zpráv. Diskový soubor je soubor s přímým přístupem a proto struktura tohoto souboru neovlivní výslednou programovou strukturu. Vzhledem k tomu, že se jedná o dialogový problém, jsou struktury obou sekvenčně zpracovávaných souborů zpráv shodné a tudíž shodné i s programovou strukturou. Na následující straně zobrazujeme proto již výslednou programovou strukturu s přiřazenými operacemi.

Předpokládáme, že program je aktivován operačním systémem na základě transakčního kódu. Jako svou první činnost vyšle program na obrazovku prázdnou masku pro volbu funkce.



Programová struktura s přiřazenými operacemi

Seznam operací :

1. open imsg
2. open omsg
3. close imsg
4. close omsg
5. read imsg
6. open disc
7. close disc
8. read disc (klíč)
9. insert disc (klíč)
10. delete disc (klíč)
11. put omsg (disc věta)
12. put omsg (blank mask pro funkci)
13. put omsg (blank mask pro vložení věty)
14. put omsg ("CHCES VYMAZAT ? (Y,N)")
15. put omsg ("VETA NENALEZENA")
16. put omsg ("DUPLICITNI KLIC")
17. zkontroluj vstupní větu a transformuj ji do diskového tvaru
18. put omsg (vstupní věta s označením chyb)
19. put omsg ("KONEC TRANSAKCE")
20. put omsg ("CHYBNY KOD")

Seznam podmínek :

C1 kód="END"
C2 nelze vyhodnotit - backtracking (příchod IGN)
C3 kód="DEL"
C4 kód="ADD"
C5 kód="DIS"
C6 nelze vyhodnotit - backtracking (věta o požadovaném klíči nalezena na disku)
C7 nelze vyhodnotit - backtracking (věta o požadovaném klíči nenalezena na disku)
C8 nelze vyhodnotit - backtracking (vkládaná věta je dobře)
C9 odpověď="Y"
C10 kód="IGN"

Program bude invertován vzhledem k souboru vstupních zpráv (imsg). Při této inverzi se vezmou v úvahu také vlastnosti operačního systému, pod kterým bude prováděn jako jeho korutina. To znamená, že fakticky odpadnou nejen operace 1 a 3, ale i otvírání a uzavírání souboru výstupních zpráv (operace 2 a 4), neboť inicializaci spojení s terminálem a jeho ukončení zajišťuje operační systém. Ten bude rovněž zajišťovat i otvírání a uzavírání diskových souborů, neboť tentýž soubor může být používán více programy v multiprogramovém prostředí (odpadnou operace 6 a 7).

V seznamu operací se nezabýváme rozvržením výstupních zpráv do jednotlivých řádek obrazovky ani jejím mazáním. Rovněž není zařazena operace provádějící eventuální transformaci vstupní zprávy do pevného formátu (odstranění řídících znaků, doplnění vynechaných mezer na konci řádku apod.), neboť některé operační systémy mohou tuto transformaci provádět před tím, než odevzdají řízení uživatelskému programu.

Operace put omsg jsou vlastně trojího druhu : pouhý přenos výstupní zprávy do bufferu, ve kterém se shromažďuje výsledná zpráva určená k odeslání k terminálu; přenos výstupní zprávy do bufferu a odeslání celého obsahu bufferu k terminálu (za tímto druhem operace následuje vždy bezprostředně v programové struktuře operace read imsg - tj. vznikají dvojice write a read diskutované v textu výše); přenos výstupní zprávy k terminálu s tím, že program současně sděluje operačnímu systému, že ho pro tuto transakci nemá vůbec už aktivovat (operace 19 v našem příkladě). Pokud je tedy v programu více operací put omsg před operací read imsg, znamená to, že se tyto požadavky na přenos zprávy k terminálu skládají v paměti do jedné výstupní zprávy, která se fysicky odešle až bezprostředně před operací read imsg, jež je realizována předáním řízení operačnímu systému v invertovaném programu s požadavkem na výstup zprávy a čekáním na další vstupní zprávu. Při tom musí být zabezpečeno, že při příchodu

nové vstupní zprávy je program znovu aktivován bezprostředně za dvojicí write a read. To je realizováno stavovou proměnnou, která musí být definována pro každý terminál. Vzhledem k tomu, že program by měl být reentrantní, aby mohl pracovat pro více terminálů současně, je zapotřebí před odevzdáním řízení operačnímu systému (před dvojicí write a read) uložit obsahy všech proměnných dat programu do souboru stavových vektorů obhospodařovaných operačním systémem.

Některé podmínky v programové struktuře není možné vyhodnotit, takže je třeba nasadit techniku zpětného sledování (backtracking). Pro nedostatek místa neuvádíme dále celý pseudokód, ale pouze příkazy quit. Tyto příkazy se v našem programu vyskytnou takto :

quit FUNC if kód="IGN" na začátku komponent YN, ERR-VETA a KK7
quit DEL if klíč nenalezen za operací 8 v komponentě KK1
quit ADD if klíč nalezen za operací 8 v komponentě KK3
quit VKL VETA if vstupní věta je dobře v komponentě ERR-VETA
za operací 17

Vedlejší efekt posit části komponenty FUNC je výhodný; iterace IGN-S nebude fakticky nikdy prováděna. Vedlejší efekty posit části komponent DEL a ADD jsou neutrální, neboť pokus o čtení věty z disku podle klíče nevadí. Příkaz quit VKL VETA v iteraci ERR-SADA má výhodný efekt, neboť vstupní věta je již při vstupu do komponenty VKL VETA zkontrolována a transformována do diskového tvaru.

Tento způsob implementace programu není jediný možný. Některé operační systémy umožňují rozdělit program do menších částí, které jsou pak aktivovány vždy na začátku, takže odpadá stavová proměnná.