O TECHNOLOGII TVORBY A ÚDRŽBY PROGRAMOVÉ ZÁKLADNY VÍCEUŽIVATELSKÉHO KOMUNIKAČNÍHO SYSTÉMU (OS POBOS)

Ing. Martin Hron

Na loňském semináři Programování 82 nastínil dr. Suchomel Orgaprojekt, Praha) možné řešení víceuživatelské úlohy ve svém příspěvku /1/. Duch popsaného řešení byl mezitím použit k vytvoření několika víceuživatelských komunikačních systémů. Všechny tyto víceuživatelské systémy úloh pracují pod operačním systémem POBOS na počítačích SM3/10. V tomto příspěvku není uveden podrobný popis systému. Ten byl přednášen v kurzu 815 KSNP a je k dispozici v příručce vydané KSNP k tomuto kurzu.

Systém je problémově orientovaný tzn., že je vytvářen již s ohledem na zpracovávanou problematiku. Může se tedy zdát, že není obecně použitelný a je obtížně aplikovatelný na jinou úlohu, než pro kterou byl původně určen. Přestože systém je rozsáhlý jak z hlediska programového vybavení, tak z hlediska skladby datové základny, pokusím se ukázat, že ani tvorba ani údržba systému a jeho další rozšiřování není pracné ani časově náročné (to se týká i přemosení systému do jiného prostředí - jiný uživatelský problém, jiná konfigurace počítače), zvláště máme-li k dispozici účinné nástroje v podobě výkonného textového editoru a pomocných dávkových programů doplňujících vhodně operační systém.

Z hlediska technologie výstavby je systém složen z několika odlišných oblastí.

1. Rezidentní programové základna

Tato základna zajišťuje spuštění a chod celého systému. Zabezpečuje řízení V/V operací na displejích, řízení V/V operací nad datovou základnou, uspokojuje požadavky na výměnu uživatelských rutin v operační paměti atd. Skládá se z "MAIN" programu (monitor displejové sítě) a ze všech subroutin vykonávajících výše uvedené funkce. Patří sem také stavový vektor a další segmenty (systémová knihovna SYSLIB, zkrácená fortranská knihovna FORTLIB

a další). Celá tato základna má tu důležitou vlastnost, že po celou dobu práce systému zaujímá přesně vymezené místo v operační paměti tak, jak byla zavedena při spuštění do paměti. Oblast za horní hranicí této rezidentní části systému je vymezena vlastním výkonným uživatelským segmentům, které tuto oblast obsazují podle vzniklých požadavků systému. Z hlediska tvorby systému je důležité, že rezidentní programová základna je jedinou oblastí systému, která je závislá na hardwareovém prostředí. Z toho vyplývají dvě skutečnosti. Předpokládejme, že máme vygenerovaný víceuživatelský systém pro konfiguraci čtyř displejů libovolných typů (existující systém předpokládá použití displejů typu VDT, MERA nebo SM7202) a jedné tiskárny. Jestliže celý vytvořený software přeneseme na jiný počítač stejné konfigurace, nemusíme na této rezidentní části systému měnit vůbec nic (je pouze třeba přizpůsobit hardwaru ovladače displejů, ale to je již záležitost týkající se operačního systému, nikoli tohoto aplikovaného systému). Jestliže hardwareové prostředí úlohy zůstane stejné a bude se zpracovávat nová uživatelská problematika (jinými slovy bude se vytvářet systém se stejnou konfigurací pro jiné uživatelské prostředí), vytvoření programové základny nového systému tedy spočívá v prostém zkopírování již existující základny systému starého. Jestliže se rozhodne programátor při generování systému změnit konfiguraci, například použít tří displejů místo čtyř, musí již provést větší zásah - přizpůsobit stavový vektor (SV). Stavovému vektoru je v rezidentní části operační paměti vyhrazena oblast, která i při změně konfigurace musí zůstat zachována. Protože oblast SV je rozdělena na tolik rovnocenných segmentů, kolik je použito uživatelských displejů v systému, musí se provést při změně počtu displejů přerozdělení SV. Například při zmenšení počtu displejů se část SV určená jednomu uživatelskému displeji zvětší a naopak. Složitější situace nastane, jestliže bude systém vygenerován pro více displejů než osm pět. Oblast SV se v paměti nahradí ovladačem stavového vektoru HSV, který zajistí přenos dat přes stavový vektor umístěný na vnějším paměťovém médiu.

2. Výměnné programové segmenty

Toto jsou přeložené programové moduly sestavené s rezidentní části systému a uložené v nepřemístitelném tvaru na vnějším paměťovém médiu v tzv. load modulu. V tomto modulu jsou programové

segmenty umístěny pod svým pořadovým číslem, přes něž je také realizován požadavek CALL příslušného segmentu. Po vzniku takového požadavku je programový segment z load modulu přečten a zaveden do operační paměti do oblasti bezprostředně za rezidentní část systému (ve stávajících systémech je velikost rezidentní oblasti $100_{(8)}$k, velikost překryvných segmentů je $10_{(8)}$k). Společnou vlastností všech překryvných programových segmentů je jejich nezávislost na hardwareovém prostředí. V load modulu jsou uloženy programové složky třech z hlediska funkce i technologie tvorby odlišných typů.

a) Servisní programy

Velká část uživatelských rutin používá standardní funkce, jejichž charakter je obecný. Jsou to například kontroly vstupních údajů z displejů, pomocné programy pro údržbu datové báze atd. Obecnost těchto programů spočívá v tom, že jejich použití není závislé na uživatelském problému a lze je tedy považovat za součást programové základny aplikovaného systému. Z hlediska technologie tvorby jsou servisní programy příbuzné uživatelským výkonným segmentům.

b) Uzlové programy pro volbu další činnosti

Před započetím výstavby vlastního uživatelského systému musí mít programátor jasno v analytickém rozboru uživatelské úlohy (resp. skupiny nebo skupin úloh). Vychází z předpokladu, že po spuštění systému se uživatel bude nacházet v kořeni celého stromu systému, tj. v uzlovém bodě, kde bude mít možnost volit svoji další činnost (oblast činností). Tento bod je tedy prvním uzlem větvení a každá volitelná větev se může dále větvit na další podskupiny programů. Až teprve na konci posledního rozvětvení jsou umístěny vlastní výkonné programy zajišťující zpracování vlastní uživatelské úlohy. Všechny uzly větvení v celém stromu systému jsou realizovány velmi příbuznými programy pro volbu činnosti. Tyto programy jsou specifické pouze počtem možných voleb a obsahem, který vysílají na uživatelský displej, případně také pomocným textem pro nabídkudalší činnosti. Je zřejmé, že má-li programátor jasnou představu, jak celý strom systému bude vypadat, může vytvořit celý strom až po okrajové uzlové programy, aniž by byl v celém systému jediný výkonný uživatelský program. Je-li vytvořena nutná část datové základny, je takto vytvořený systém již schopen spuštění a programátor je jím aktivně veden při vytváření vlastních uživatelských programů tak, že postupně

vytváří funkční oblasti systému a systém tak neustále rozšiřuje. Přitom již dosud vytvořené skupiny uživatelských úloh se mohou zároveň využívat v rutinním provozu. Tato vlastnost systému je důležitá především z těchto důvodů: Uživatel má k dispozici nejdůležitější části své úlohy mnohem dříve, než je systém dotvořen ve své konečné podobě. Záhy tedy může odhalovat nedostatky v analytickém zpracování úlohy vzniklé obvykle nedorozuměním mezi uživatelem - zadavatelem a analytikem. Obvykle také teprve až při pohledu na způsob zpracování problematiky začíná mít uživatel podnětné nápady, z nichž některé mohou původní analýzu dosti značně pozměnit. Protože v těchto chvílích systém není zdaleka dotvořen, má programátor možnost mnoho zamýšlených programů pozměnit bez velkého násilí k obrazu uživatele. Kdy je vlastně systém definitivně hotov? Tehdy, když je vytvořen celý strom uživatelských programů a všechny větve jsou zakončeny výkonnými uživatelskými segmenty ? Nikoliv. Uživatel kdykoli může přijít s požadavkem na vytvoření celé další skupiny úloh nebo na rozšíření stávající skupiny úloh o další funkci. Znamená to na konci některé větve nahradit koncový výkonný program uzlovým, který posune tento výkonný program na vyšší úroveň a vytvoří prostor pro začlenění další skupiny programů do systému - systém je otevřený vůči dalšímu rozšiřování. Kdyby programátor nebyl omezen kapacitou kazetových disků používaných na počítačích SM3 a svým časem, mohl by strom systému rozšiřovat teoreticky neomezeně. (Pro ilustraci:jeden z existujících systémů obsahuje ve fázi, kdy je dotvořen asi z jedné třetiny původně zamýšleného záměru, zhruba 300 uživatelských programů). Podobnost uzlových programů dovoluje jejich efektivní vytváření, má-li programátor k dispozici příslušný nástroj. Tímto nástrojem je dávkový program, jehož vstupem je obecná maska vytvářeného programu a specifické hodnoty příslušného uzlového programu (počet větví uzlu, čísla a názvy volaných programů. číslo pomocného tisku s nabídkou další činnosti). Pomocí tohoto nástroje lze v e velmi krátké době vytvořit všechny potřebné uzlové programy ve zdrojovém tvaru.

c) Koncové výkonné programy

Tyto programy jsou již vesměs specifické. Jejich úkolem je zajistit vlastní zpracování uživatelské úlohy. V systému jich je největší počet a efektivita práce na výstavbě systému je značně

závislá na efektivitě tvorby těchto jednotlivých programů.

Na takto rozsáhlém systému programů většinou pracuje tým několika programátorů. Z toho důvodu je nezbytně nutné, aby programátoři měli "společnou řeč". Touto společnou řečí není pouze programovací jazyk, ale především metoda programování. Pak nezáleží na tom, který pracovník jaký program vytváří - všichni rozumí programům ostatních spolupracovníků (to je důležité i v rutinním provozu systému s hlediska údržby programů).

V praxi se při výstavbě takovéhoto systému plně osvědčila metoda strukturovaného programování (tak jak je přednášena v kurzu 815 v KSNP podle základů podaných v /2/). Při přísném dodržování jejích zásad nelze na první pohled poznat autora programu. Metoda zároveň dovoluje vytvořit logické schéma celého programu za stolem bez zdlouhavého psaní obvyklých podrobností programu. Navíc zkušený programátor je schopen vytvořit si logické schéma jednoduššího programu v hlavě a přípravnou fázi tak dokáže zkrátit na minimum.

Všechny uživatelské překryvné segmenty jsou vytvářeny jako invertované subroutiny (viz /3/). Inverzi těchto subroutin zajišťuje stále se opakující sekvence standardních příkazů, které je možno sloučit do jednoho makropříkazu, zdrojový tvar programu se pak značně zkrátí. Zdroj můžeme zkrátit také o standardní deklarace popisující strukturu záznamu použitého datového souboru. Takto vytvořený zdroj musíme před vlastní kompilací a dalším zpracováním nejprve předzpracovat do formálně správného fortranského tvaru. K tomu nám poslouží program PREFOR (Precompiler Fortra), který má několik funkcí:

a) Provede rozvoj maker. Na počátku programu zapíše sekvenci pro "OPEN" (počáteční nastavení stavové proměnné). V místě požadavku "READ", "WRITE" nebo "CALL" označí stav programu ve stavové proměnné, zapíše RETURN a označí místo návratu specifickým návěštím. Na konci programu zapíše PREFOR podle počtu realizovaných maker počítaný přepínač GOTO podle stavové proměnné programu.

b) Spojí všechny vstupní segmenty zdrojového textu do jednoho výstupního souboru v pořadí jejich zadání. V této fázi zpracování

můžeme tedy vlastní specifický zdroj spojit se všemi obecnějšími moduly (deklarace, commony, ext. funkce ...) .

c) Dokáže nahradit zvolený znakový řetězec z textového souboru novým. Tato funkce mj.dovoluje velmi rychle vytvářet zdrojové tvary nových programů, jestliže se tyto programy jen velmi málo liší od nějakého již vytvořeného programu.

Vytváření nového programu programátorem fakticky končí napsáním "zhuštěného" tvaru těla zdroje. Ostatní činnost počínaje aplikací PREFORu a konče začleněním programového segmentu do systému je automatizována pomocí systémového programu pro řízení dávek BATCH. Všechny programy, jejichž zdroje nevyžadují žádné specifické zdrojové moduly, je možno zpracovat jedním universálním programem pro režim BATCH. V tomto režimu bude vytvořen úplný zdroj programem PREFOR, zdroj bude zkompilován překladačem FORTRA, dále může být zařazeno vytvoření dokumentačního listingu zdroje a následuje sestavení programu. Velký rozsah programového systému si při sestavování programů vyžádal zvláštní postup. Systémový program LINK dokáže sestavit max. 80 programů. Každému novému programu přiřadíme stejné jméno. Takto označený modul pak sestavíme s rezidentní základnou systému. Mapa sestavení dvou různých programů pak bude až do adresy $100_{(8)}^{k}$ shodná, na této adrese bude uloženo standardní globální jméno výměnného segmentu. Takto tedy můžeme zpracovávat každý segment nezávisle na ostatních. Zpracování nového programu je ukončeno jeho zařazením do zásobníku programových segmentů - load modulu. To provede program PRESAV. Protože rezidentní část všech programových segmentů systému je shodná, bylo by zbytečné ukládat do load modulu celý sestavený program. PRESAV proto celou rezidentní část od segmentu odtrhne a segment uloží podle jeho pořadového čísla do load modulu. Od tohoto okamžiku je nový programový segment použitelný v rámci rutinního provozu systému.

3. Datová základna

Kapacita operační paměti počítače SM3 ani kapacita vnějších pamětí nedovoluje vytvořit datovou základnu s vlastnostmi databanky. Přesto se z uživatelského hlediska vlastnosti databáze musí blížit vlastnostem databanky. Všechna vložená data musí být okamžitě po vložení do databáze k dispozici uživateli (tj. všem uživatelům), a

ze všech hledisek třídění. Třídění informací nesmí být porušeno ani při vyřazení jakékoli položky z databáze. Programové vybavení systému musí tedy zabezpečit update databáze okamžitě v interaktivním režimu, a to tak, aby nebyl uživatel časově omezován. Databáze také musí být schopna přijímat data hromadným náběhem v dávkovém režimu. Některé informace se do datové základny dostávají až jako vedlejší produkt činností probíhajících nezávisle na práci systému. Při update databáze v interaktivním režimu zabezpečují propojení informací servisní programy v okamžiku vzniku takového požadavku uživatelského programu. Při hromadném náběhu (např. z DP) nejsou data z hlediska potřeb systému setříděna. Setřídění se provede opět dávkově, čemuž slouží dva nástroje :

a) Program SORT třídí datové soubory menšího rozsahu tak, že třídění provádí v operační paměti (pro pracovní soubor je v OP vyhrazena oblast o velikosti 8 kB).
b) Program SM (SORT-MERGE) je určen pro třídění souborů velkého rozsahu. Vytváří si pracovní soubor na vnějším paměťovém médiu. Velikost tříděného souboru je omezena pouze kapacitou paměťového média, na němž je soubor uložen. Protože třídění rozsáhlých souborů je časově náročnější, vytváří si SM ještě pracovní soubor check pointů uložený na magnetickém disku. Použití tohoto souboru dovoluje pokračovat v třídění souboru od okamžiku přerušení předchozího nedokončeného třídění, jestliže k takovému přerušení dojde.

V databázi jsou obsaženy soubory několika typů.

a) Textové soubory. Programy systému často používají pro výpisy standardní někdy i velmi rozsáhlé texty,které se mnohdy opakují (pomocné informace,nabídka činnosti v uzlech volby, atd.). Tyto texty jsou umístěny ve zvláštním datovém souboru a výpis příslušného textu provádí zvláštní servisní program. To má také tuto výhodu: Jestliže dojde ke změně obsahu některého textu, není třeba opravovat zdrojový program a znovu jej začleňovat do systému (přitom by takových programů mohlo být i více), stačí pouze opravit textový soubor a text je okamžitě k dispozici všem programům, které jej používají. Zvláštní charakter textových souborů spočívá v tom, že práce uživatele nemá na jejich obsah vliv.

b) Soubory číselníků. Toto jsou seznamy textů, jimž jsou přiděleny uživatelem zvolené krátké kódy. Uživatel při komunikaci se systémem používá pouze těchto kódů, systém sám vyhledá v případě potřeby v číselníku příslušný text. Stejně tak v datových souborech se použitím těchto kódů zkracuje délka použitých záznamů. Číselníky vytváří a udržuje sám uživatel přímo v komunikačním režimu systému. Dávkově se provádí pouze začlenění nového číselníku do databáze a případný tisk seznamů na tiskárnu.

c) Vlastní datové soubory uživatelských informací. V těchto souborech jsou uložena všechna ostatní uživatelské data. Každý datový soubor je jedinečný, třídění takového souboru je provedeno pomocí zvláštních pointrových souborů. Každé hledisko třídění vyžaduje jeden až dva pointrové soubory (podle velikosti datového souboru). Hledání položky update souboru se provádí pomocí systému servisních programů z load modulu, které využívají právě pointrových souborů. Veškeré změny v třídicích klíčích datového souboru se během práce systému musí okamžitě promítnout i do pointrových souborů. Jaké činnosti nad datovým souborem předpokládáme?

ca) Vložení nového záznamu. Nový záznam se vloží do prvého volného místa v souboru. Takové místo se najde velmi rychle pomocí propojení volných adres v datovém souboru indexy. Toto propojení se provede dávkovým programem před prvním použitím datového souboru. Po vložení záznamu do datového souboru je třeba zajistit nové zřetězení zbylých indexů volných záznamů a provést propojení pointrových souborů pomocí servisního programu pro insert prvku.

cb) Vyřazení záznamu. Po nalezení a vyřazení záznamu se v datovém souboru provede propojení indexů volných záznamů a pomocí servisního programu pro delete prvku se zřetězí pointrové soubory (přemostí se vyřazený záznam).

cc) Nalezení a oprava položek v záznamu. Pokud oprava záznamu nezasáhne třídicí klíč, na propojení pointrových souborů se nic nezmění. Jestliže se bude opravovat klíč, je nutno provést update pointrových souborů. V tomto případě je to vlastně spojení činnosti delete starého klíče a insert klíče nového. Na propojení indexů volných záznam. v datovém souboru se v tomto případě ni nezmění.

Tvorba takovéhoto rozsáhlého programového systému si vyžádala také vytvoření mnoha pomocných dávkových programů, které mají většinou obecnější použití a rozšiřují tak možnosti vlastního operačního systému. Pro jejich popis není v tomto příspěvku prostor. Účinným nástrojem k tvorbě programů systému je také Textový editor nahrazující systémový EDIT (autory ZN jsou dr.Suchomel a Ing.Matějková). V současné době je tento editor nahrazován víceuživatelským textovým editorem, který pracuje na podobném principu jako výše popisovaný víceuživatelský systém.

Literatura

/1/ Suchomel Jiří, dr.: K problému paralelního zpracování, Sborník Programování '82, DT ČSVTS

/2/ Jackson M.A.: Technologie strukturovaného programování (nevydaný český překlad)

/3/ Suchomel Jiří, dr.: Technologie strukturovaného programování příručka KSNP, 1982