UŽITÍ DATABÁZOVÝCH PŘÍSTUPŮ V ŘÍZENÍ DÁVKOVÉHO ZPRACOVÁNÍ ÚLOH HZD A VE VTV APLIKACÍCH

Ing.Zdeněk Rusín,
Správa operačních systémů, VÍTKOVICE k.p.

1. Katalogový uzel uživatelova objektu v dávkové úloze HZD

V článku /1/ jsme popsali koncepci výpočetního systému, v němž obecně známý pojem adresáře datových souborů přerůstá v ústřední všudepřítomnou databázovou strukturu - systémový katalog, který obsahuje deskripci všech objektů uživatelské i systémové povahy. Jeden objekt je popisován jednou katalogovou větou, tzv. uzlem. Z hlediska uživatelského jde skutečně především o adresář souborů a medií, správce systému zde popisuje veškeré údaje o hardware, hierarchii uživatelů a magnet. mediích a o centrálně sdílených datových souborech včetně sw knihoven. Katalog má obě z našeho hlediska nejdůležitější vlastnosti databáze: reentrantnosti kódu ekvivalentní mnohonásobnou sdílitelnost dat a přirozené oddělení fyzického uspořádání dat od aplikační sféry, realizované databázovým rozhraním. Význam prvé vlastnosti je evidentní, důležitost druhé skutečnosti podstatně vzroste tehdy, je-li aplikována jako obecný způsob přístupu k datům v daném systému. Pak jde o požadavek oddělení popisu fyzického uspořádání dat od uživatelského programu, tedy o zcela jinou koncepci implementace IO systému, k čemuž se vrátíme v závěru článku.

Zobecněním katalogového uzlu dojdeme k pojmu uzlu uživatelova objektu, dále jen UON - user object node, uzlu, jehož interpretaci si provádí uživatel sám. Pomocí UON jsou organizována ta systémová data, jež nemohou nebo alespoň by neměla být konstantami systému. Uvažujme dávkovou egendu HZD s pravidelnými a občasnými chody, kde kromě věcné obsahové správnosti zpracování nutno hlídat řadu organizačních záležitostí jako posloupnost chodů a přiřazení správných dat. Je zřejmé, že porušení formální správnosti zpracování vede k nesprávnosti věcné. Je ale rovněž zřejmé, že logickou formální správnost lze zajistit, snad až na pokyn k započetí zpracování konkrétního chodu, automatisovaným přiřazováním dat v rámci řídicích programů

úloh. Potřebné organizační údaje jsou pak obsahem UON agendy. UON tedy obsahuje odkazy na katalogové uzly souborů či médií, záznam o posledním úspěšném kroku v rámci konkrétního chodu, ale též další údaje, jež jinak vstupují do zpracování jako parametry příkazů řídicího jazyka úloh, jako parametry uživatelských programů nebo by musely být dodány dialogem programů s operátorem. UON může obsahovat údaje pro zobrazení pokynů pro operátory, či obsluhu agendy v případě chybových stavů atd. Význam UON vzroste tehdy, jsou-li procesy jednotlivých úloh separovány od operátorské aktivity, což je nezbytné v systémech s komunikacemi, kdy reálný čas systému sdílejí řádově desítky uživ.procesů. Uzlem lze ostatně výhodně řídit i interaktivní aplikace typu "selekce z menu". Pokusme se nyní definovat vhodnou strukturu dat v UON a minimální uživatelské rozhraní.

Přirozenou se ukazuje samodefinující struktura tvaru typ-délka-data /TLD/. Při obvyklé organizaci paměti postačí 1 byte pro typ a 1 byte pro délku k zobrazení UON jako virtuální stránky o 255 řádcích a 255 sloupcích, kde celková délka dat, včetně TL údajů, je limitována rozměrem katalogové věty. Uživatelsky orientované systémové rozhraní může být tvořeno šesti činnostmi : selekce UON (existujícího či potenciálního - tedy jen reservace místa v katalogu), vytvoření UON, likvidace UON, zpřístupnění uzlu jinému uživateli, čtení dat z UON, aktualizace dat v UON. Selekce má zásadní význam pro opakovaný přístup k UON, jejím smyslem je lokalizace katalogové věty pro úsporný fyzický přístup ke katalogu (procesy simultánně aktualisují různé katalogové uzly, především uzly souborů). Příslušné systémové procedury jsou volatelné jak v rámci řídicího jazyka úloh tak, alespoň potenciálně, též z každého vyššího jazyka implementovaného výpočetním systémem. Řídicí jazyk systému obsahuje dostatečný aparát pro práci se znakovými řetězy a konversi čísel v binárním a znakovém tvaru. Kódování řídicího programu dávkové úlohy selektuje UON, zpracuje jeho obsah spolu s aktuálními hodnotami parametrů řídicího programu do tvaru parametrů jiných příkazů řídicího jazyka, především příkazů pro přiřazení datových souborů, validuje existenci těchto souborů, provádí větvení podle výsledkových kódů příkazů řídicího jazyka i uživatelských programů, s nimiž komunikuje, provádí neztytnou komunikaci s operátorem při neobvyklých stavech, aktualizuje průběžně data v UON. V zásadě všechny tyto činnosti jsou dostupné

v rámci každého uživatelského programu. Zkoumejme, zda střetelné výhody UON lze přenést do výpočetních systémů JSEP, v nichž katalogová struktura chybí. Nejde o samoúčel, motivací je snaha zachránit co nejvíce uživatelského komfortu při přechodu k JSEP po zániku existující instalace.

Pomineme-li jednorázovou inicialisaci UON, redukuje se problém na generování řídicích programů úloh v rámci kódování vyššího jazyka, neboť dosavadní řídicí jazyky patrně nejsou schopny zajistit operace nad řetězy znaků a binárními řetězy, jimiž jsou údaje v UON. Tento problém byl již úspěšně řešen, viz /2/ a /3/.

Myslitelné je ale i zásadnější řešení, spočívající v náhradě interpretátoru JCL novým kódováním pro rozšířený či ještě lépe zcela nový řídicí jazyk. Podmínkou implementace na úrovni jednotlivé instalace je pochopitelně přístup ke zdrojovým textům modulů operačního systému, přítomnost assemblerového jazyka a možnost generování systémů. Např. instalace ICL-4-50, pracující ve Vítkovicích od roku 1968, všechny tyto podmínky splňuje po celou dobu své existence, přičemž provozovaný operační systém je srovnatelný s DOS a assemblerový jazyk je totožný.

Neméně důležitou otázkou je implementace mohonásobné sdílitelnosti katalogu. V opačném případě je UON redukován na jakýkoli datový soubor s přímým přístupem, v němž aktualizace věty znamená aktualizaci fyzického bloku okamžitě. Toto zúžení je nepřípustné všude tam, kde souběžnost chodů spolu s eventuální synchronisací některých činností dávkových úloh je nezbytná třeba z důvodů časových, když řazení chodů za sebou by přesáhlo časovou kapacitu instalace. Problém lze řešit rozšířením primitivní instrukce SVC o funkci fyzického přístupu ke strukturám typu UON a předáváním nezbytných dat ve vhodných obecných registrech. Pak je ekvivalent katalogu přiřazen supervisoru, tedy přístupný z každé úlohy. Procedury uživatelského rozhraní pro práci s UON lze pak chápat jako rozšíření toho vyššího jazyka, v němž je implementován generátor řídicích programů dávkových úloh, nebo jako součást interpretátoru modifikovaného řídicího jazyka či jako obojí. I zde jde o generování operačního systému dle potřeb instalace. Domníváme se, že odhlédneme-li od nejrůznějších

nepřekonatelných překážek administrativně-právní povahy, je navrhovaná maximální koncepce realizovatelná i jako vhodné volitelné rozšíření konkrétní verze DOS, například tedy DOS 4 pro připravované počítače EC 1027.

2. Databáze, uzel uživatelova objektu a VTV aplikace

Zkoumejme užití UON v úlohách vědecko-technické povahy. Nejde nám samozřejmě o aplikaci za každou cenu. Užití UON je smysluplné např. tehdy, je-li úloha natolik časově náročná,že je nutno programové vybavení koncipovat tak, aby výpočet byl přerušitelný/restartovatelný. Test podmínky přerušení, ať už je dána překročením výpočtového času či operátorským zásahem z jiných důvodů, není problémem v úlohách s cyklickým algoritmem. Je známa široká třída algoritmicky podobných problémů mechaniky kontinua, elektromagnetismu, hydrodynamiky či mechaniky hornin, jež metodami variačního počtu převádějí řešení soustav parciálních diferenciálních rovnic na opakované sestavování a řešení soustav rovnic lineárních, kde řešení jednoho kroku výpočtu je počátečním stavem dalšího procesu. Zde nalezneme v každém kroku dvě přirozená místa přerušení procesu : po sestavení soustavy rovnic a po jejím vyřešení. Uživatelský program po indikaci přerušení v těchto místech provede "úschovu" momentálního stavu výpočtu do vhodné uživatelské filestore a ukončí činnost. Řídicí údaje pro restart lze umístit do UON. Užívá-li úloha diskové soubory s přímým přístupem, nemusí být v každém operačním systému použitelný mechanismus checkpointu, jenž bývá omezen na sekvenční soubory, i když je myslitelný takový systém, kdy se souborem s přímým přístupem je spřažen sekvenční soubor se záznamem změn prvého a pak je checkpoint realisovatelný. Pak nezbývá než vybavit uživatelský program "implicitním" checkpointem. Toto řešení je dokonce výhodné tehdy, lze li standardní checkpoint aplikovat nikoli na stav procesu, ale pouze, a zároveň nezávisle na procesu, na soubor samotný. V složitějších případech tedy uživatelský program uchovává historii průchodu svými moduly v oblastech svých pracovních dat /např.v common oblastech ve fortranu/, jež kopíruje do filestore. Nezbytné minimum řídicích údajů pro rekonstrukci stavu výpočtu po restartu je zapisováno do UON. Zde vícenásobná sdílitelnost UON není podstatná, je tedy popisovaný princip realisovatelný v podmínkách JSEP okamžitě.

Komplikace vyvstávají spíše při řízení IO operací v rámci úklidu, kdy je nežádoucí, aby doba úklidu byla srovnatelná s dobou jednoho kroku výpočtu. Toto nebezpečí hrozí, jsou-li IO operace prováděny standardními prostředky vyššího programovacího jazyka. Přihlédneme-li k tomu, že jazyky fortran, algol, pascal umožňují povětšině nanejvýše operace se soubory typu direct serial s pevnou délkou věty, narážíme zde na největší tradiční slabinu těchto jazyků, jež se jeví anachronickým omezením drasticky snižujícím účinnost mnoha úloh VTV povahy. Při diskretizaci prostorového kontinua popisujeme takové topologické útvary jako body, čarové, plošné a prostorové elementy, jež se vymykají unifikovanému zobrazení do vět fixní délky a program buď musí drobit data do mnoho různorodých souborů, nebo plýtvá místem ve filestore i počtem přenosů při kompromisní volbě délky věty. Všechny naznačené problémy svým způsobem řeší radikální zvrat v koncepci VTV úlohy, kdy jsou všechna data s výjimkou výstupů organizována v (hierarchické) databázi, s níž uživatelské programy komunikají. Zde i UOF je nahrazen databázovou větou. Problém se tedy redukuje na existenci databázového rozhraní pro tyto jazyky.

Implicitní checkpoint je ovšem nezbytností i v úlohách, jež nemají zřetelný makrocyklický charakter. Uvažme problém řešení vlastních frekvencí rozměrové prostorové konstrukce dobývacího velkostroje, kde jde o několik maticových transformací časově extrémně náročných. Nyní je nezbytností přerušitelnost doslova kdykoli. I zde postačíme s již popsaným řešením doplněným databází. Představme si třeba algoritmizaci řešení soustavy rovnic eliminací po submaticích, kdy databázová věta obsahuje dva stavy submatice - před prováděnou operací a po operaci. Program v pracovních datech a současně v jejich databázové kopii udržuje indikaci operace a informaci o aktuálním pruhu submatic, s nímž pracuje. Test podmínky přerušení lze pak umístit v mikrocyklu přechodu na nový pruh submatic nebo po každé n-té zpracované submatici v pruhu. Toto řešení je imunní i vůči selhání výpočetního systému, neboť databáze obsahuje veškeré navigační údaje pro restart až k operaci, jež byla před přerušením prováděná a submatici, jíž se týkala, zavede-li se indikace platného stavu submatice v rámci její databázové věty. Není-li dostupný databázový systém, je možno jej simulovat soubory s přímým přístupem. Tím se

vracíme k otázce přístupu k souborům všech implementovaných uspořádání ze všech implementovaných programovacích jazyků na dané instalaci. Jí věnujme poslední část článku.

3. Universální mechanismus přístupu k datovým souborům

Připomeňme, že katalogový uzel souboru dat obsahuje údaje o fyzickém uspořádání a konkrétním umístění souboru na magnetických mediích. Nemělo by být žádným zvláštním problémem, kromě realizace samotné, pochopitelně, rozšířit v tomto smyslu každý existující adresář souborů. Je tedy možno všechny činnosti závislé na konkrétním souboru provést až ve chvíli uživatelského otevření souboru. Máme na mysli hlavně selekci typu souboru a přiřazení IO zásobníků, ev. validaci smysluplnosti programem požadovaného přístupu. Dokonce lze učinit ústupek v tom, že je-li to nezbytné vzhledem k neexistenci katalogu v systému, dodá tyto údaje uživatelský program. Toto tvrzení lze ovšem vyslovit i takto: ve všech existujících výpočetních systémech druhé a vyšší generace lze organizovat IO systém tak, že IO operace dané definicemi vyšších programovacích jazyků mohou být bez dalších programátorských zásahů ve všech implementovaných jazycích rozšířeny o universální IO systém pracující jednotným způsobem pro všechna uspořádání dat, pomocí něhož jsou naopak IO systémy vyšších jazyků implementovány. Ve zmíněném systému ICL-4-50 není problémem napsat v assembleru rutinu, jež po validaci parametrů sestaví definiční tabulku souboru a poté provede ostatní činnosti funkce "open". Assemblerový jazyk zde užívá týchž makroinstrukcí DTF.. a DTFEN jako systémy JSEP či IBM, náš přístup je pouze nahrazuje jinak uspořádaným vlastním kódováním. Tato koncepce má další výhodu v tom - nebrání-li tomu operační systém - že k témuž souboru může uživatelský program současně přistupovat vícekrát, a to event. vždy různým způsobem. Věnujme se důkladněji struktuře našeho IO systému.

Povětšinou se jeví účelné rozdělit systém na sféru fyzického přístupu k magnetickým mediím (předpokládáme důsledný SPOOLING na vstupu i výstupu) a na sféru uživatelského rozhraní.

Sféra fyzického přístupu obsahuje následující rutiny: pro diskové

media postačí čtení bloku, zápis nového bloku, aktualizace existujícího bloku, pro magnetickou pásku čtení bloku, zápis bloku, zápis tape mark, přeskok k následující tape mark, reposice o libovolný počet bloků a extense souboru o nový kotouč. Páskové rutiny musí zabezpečit pohyb vpřed i vzad, diskové rutiny zase selekci bloku relativně od aktuální posice oba směry.

Uživatelské rozhraní pak musí pomocí sféry fyzického přístupu realizovat každý smysluplný přístup k datovému souboru daného uspořádání. Např. pro ISAM organizaci musí uživatelské rozhraní zabezpečit přístup po větách pro počáteční naplnění souboru (tzv.load mode), pro běžnou aktualizaci s rušením a vkládáním vět, pro seriové čtení datového souboru, ale rovněž přístup po blocích jak k datům, tak k oblasti indexů, který je efektivní pro kopírování souborů. Podle typu souboru a typu přístupu, jenž uživatelský program popíše v parametrech rutiny pro otevření souboru, je touto rutinou vybrán vhodný mechanismus přístupu k datům, dále jen DAM - data access mechanism. Rutina pro otevření souboru je tedy selekcí DAMu a vrací uživatelskému programu vhodným způsobem referenci na příslušnou rutinu uživatelského rozhraní, jež zabezpečuje přenos dat ze sféry fyzického přístupu k uživatelskému programu. V assembleru lze tento problém řešit technikou dynamicky alokovaných segmentů kódu s pohyblivou adresou (floating segment), umožňuje-li tuto konstrukci link editor. Poznamenejme, že tímto způsobem supervisor podle potřeby přivolává neresidentní rutiny. Uživatelský kód volá DAM pomocí další rutiny, jež obdrží referenci DAMu jako parametr spolu s parametry typu činnosti a referencí uživatelského zásobníku, jímž může být kterákoli datová položka. Protože se běžně vyskytuje vícenásobné opakování téže IO operace, pak s ohledem na syntaxi algolu či pascalu je vhodné mít další rutinu pro opakování předešlé operace DAMu. Je výhodné tyto tři viditelné rutiny uživatelského rozhraní - selekci DAMu, nastavení a provedení operace DAMu, opakování operace DAMu - kódovat jako celočíselné funkce, kde hodnota funkce je výsledkovým kódem DAM operace.

Zbývá charakteristika parametrů DAM rozhraní. Některé lze organizovat do pole celočíselných údajů, nejlépe po dvojicích tvaru typ-hodnota. Jsou to např. typ přístupu k souboru, počet zásobníků

fyzické sféry, požadovaná operace, relativní vzdálenost nové větynového bloku od okamžité posice atd. Jiné parametry mají charakter reference a s výjimkou pascalu je vhodné je zpracovat samostatně. Sem patří reference na zásobník dat uživatele, reference klíče u souborů s přímým přístupem po větách, reference proměnné aktuální délky přenášených dat, reference DAMu a pochopitelně reference samotného souboru při selekci DAMu (může jít o znakovou položku se symbolickým jménem souboru). Uzavření DAMu je provedeno jako operace DAMu a nevyžaduje zvláštní rutinu.

Popsaná koncepce DAMu řeší problémy IO operací na každé myslitelné úrovni uživatelských aplikací bez ohledu na jazyk v němž je aplikace implementovaná. Posledním výrokem máme na mysli i to, že programátorský tým, obeznámený s databázovými principy má možnost pomocí DAMu pro blokový přístup k diskovým souborům implementovat v kterémkoli jazyce vhodnou databázovou strukturu pro svůj specifický (vědecko-technický) problém, aniž by řešení bylo degradováno omezeními konvenčního programovacího jazyka. Naopak, výběr jazyka může být podřízen jiným podstatným aspektům řešeného problému, v citovaných rozměrných aplikacích maticového počtu například především otázkám efektivnosti adresovacích mechanismů jednotlivých jazyků při práci s vícedimensionálními číselnými poli.

Domníváme, se, že koncepce DAMu je plně slučitelná s principy dnešních operačních systémů počítačů JSEP a její implementace ve tvaru volitelného rozšíření konkrétní verse operačního systému by byla uživatelsky velmi účinným rozšířením těchto systémů.

Závěrečné poznámky :
reálným výpočetním systémem inspirujícím tento článek je počítač ICL 2960 s oper.systémem VME/B, pracující ve VS Vítkovice od r.1978.
Užití UON zde popisované je samostatným výsledkem instalace, nezávislým na firemní literatuře. DAM je zobecněním koncepce tzv.record access mechanismu, RAMu, plnícího ve VME/B částečně popisovanou funkci. Vše ostatní jsou názory autorovy.

Literatura

/1/ Rusín Z., Výpočetní systémy příštích let a jejich dopad na profesní sféru, sborník Programování 82

/2/ Kaniok J., Generování příkazů JCL v oper.systému OS, sborník Programování 81

/3/ Boleslav J., Systém řízení úloh JSC, sborník Programování 82

/4/ Referenční příručka uživatelských procedur, TPVOOl, udržovaná Správou operačních systémů VS Vítkovice