Ing. Jiří Balusek
Mechanizační ústředna zahraničního obchodu, Jindřišská 24, Praha 1

Zkušenosti s programováním transakčních úloh

1. ÚVOD

Více než dva roky je v rutinním provozu terminálový informační systém na Federálním ministerstvu zahraničního obchodu. Je využíván zhruba pěti odbory ministerstva a s terminály pracuje okolo jednoho sta zaměstnanců ministerstva. Je řešeno několik oblastí informací, a to různým způsobem. Naše zkušenosti s programováním a provozem jsou obsahem tohoto příspěvku.

2. Transakční úlohy

2.1. Krátce o transakčních úlohách

Předpokládám, že téměř všichni se s pojmem transakční úloha již setkali, ale přesto bych rád zopakoval několik základních skutečností.

Tak především, co je to transakční úloha? Typickým příkladem transakční úlohy je systém prodeje místenek a jízdenek na místenkové autobusy. Obsluha u terminálu pomocí dialogu s počítačem zjistí volná místa na požadovaný spoj, zadá, která místa si zákazník kupuje a pomocí tiskárny vytiskne jízdenky s odpovídajícími údaji.

Dialog probíhá po tzv. dialogových krocích, což je interval mezi odesláním zprávy z terminálu a obdržením odpovědi. Může pracovat více uživatelů s terminály, dotazy a odpovědi počítač zpracovává tzv. transakční systém, který zároveň zabezpečuje přístup k datům.

Transakční systém především musí:
- přijímat požadavky všech komunikačních partnerů
- rozlišovat, od kterého partnera přišel dotaz
- zpracovat dotaz partnera
- zajistit odeslání odpovědi tomu partnerovi, který ji požadoval
- udržet bezpečnost a konzistenci dat i při výskytu chyby
- mít možnost řídit aplikaci administračními povely.

Protože s transakční úlohou pracuje více uživatelů, transakční systém musí zabezpečit, aby pokračování dialogu u jednotlivých uživatelů na sebe navazovalo a zároveň, aby čas přemýšlení uživatelů využil k řešení dalších požadavků. Musí mít také možnost rezervovat prostředky k vyřšení jednotlivých požadavků až do skončení transakce, to znamená např. nepovolit práci s obsazením jednoho autobusu ze dvou terminálů, pokud první transakce není uzavřena.

2.2 Transakční monitor UTM

2.2.1 Krátce o transakčním monitoru UTM

Výstavba transakčního systému není jednoduchou záležitostí. K tomu, aby se tento problém podstatně zjednodušil, slouží transakční monitory. Jedním z nich je i transakční monitor UTM. Pomocí UTM lze vytvořit UTM-aplikaci, která se skládá z

- aplikačních programů, které vytváří tvůrce aplikace
- generovaného modulu KDCROOT
- administrátorových programů
- souborů, do kterých UTM odkládá data uživatele a svá řídící data

Uživatel v podstatě tvoří pouze aplikační /neboli akční/programy, které zpracovávají dotaz z terminálu a tvoří odpověď. Akční programy lze psát v jazyku ASSEMBLER nebo COBOL. Vše ostatní je generováno na základě požadavků uživatele. Uživatel má k dospozici dvě paměťové oblasti, jejichž velikost lze určit při generaci. Je to komunikační oblast a pracovní oblast. Komunikační oblast slouží k předávání údajů mezi jednotlivými aplikačními programy. Její obsah lze obnovit do stavu, ve kterém byl při skončení aplikačního programu, který předcházel ve zpracování. Je použitelná až do ukončení transakce. Obsah pracovní oblasti je definován pouze pro běh jednotlivého akčního programu, po jeho skončení v dalším dialogovém kroku není definován.

Akční programy jsou volány jako podprogramy z modulu, který celé zpracování řídí. Jednotlivé funkce transakčního monitoru se volají jako podprogramy z aplikačních programů. Funkcí je více,

ale já se zmíním pouze o několika nejdůležitějších:

- INIT - zahájení aktivního programu, obnova komunikační oblasti
- MGET - přečtení zprávy z terminálu
- MPUT - odeslání zprávy na terminál
- PEND - ukončení aplikačního programu, určení programu, který bude zpracovávat další dotaz.

S terminály lze pracovat jak v řádkovém módu, což se v podstatě nepoužívá, nebo ve formátovém módu.

2.2.2. Struktura aplikačního programu

Každý dialogový krok je řešen jedním aplikačním programem. Aplikační program ke aktivován v okamžiku, kdy byla odeslána zpráva z terminálu. Pokud je právě aktivní jiný aplikační program, je požadavek uložen do fronty a program je aktivován, až na něj přijde řada.

1/ Call "INIT" USING komunikační oblast Parametry
Inicializace aktivního programu, obnovení komunikační oblasti. V parametrech je uvedena např. délka komunikační oblasti aj.

2/ Call "MGET" USING komunikační oblast Parametry Zpráva
Přečtení zprávy uživatele z terminálu. V parametrech se udává, zda se jedná o řádkový či formátový mód, délka zprávy atd. Přečtená zpráva je uložena do oblasti Zpráva.

3/ Kontrola požadavků uživatele
Je potřeba zkontrolovat, zda požadavky uživatele jsou korektní.

4/ Zpracování požadavků uživatele
Vyhledání požadovaných dat.

5/ Vytvoření zprávy pro uživatele.

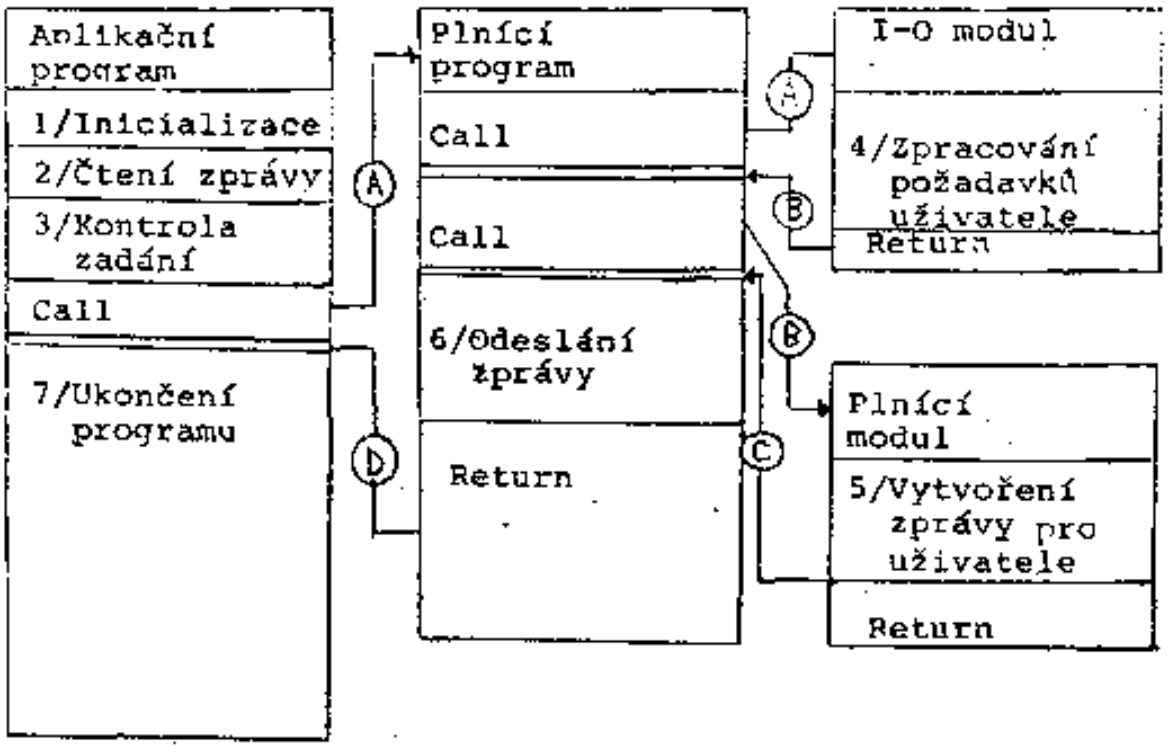
6/ Call "MPUT" USING komunikační oblast Parametry Zpráva
Data z oblasti Zpráva jsou odeslány na terminál uživatele. Parametry obsahují jméno formátu, délku zprávy atd.

7/ Call "PEND" USING komunikační oblast Parametry
Ukončení aplikačního programu. V parametrech se udává identifikace aplikačního programu, který bude zpracovávat právě odeslanou zprávu.

2.2.3. Návrh výstavby aplikačních programů

Jak bylo uvedeno, aplikační program zpracovává jeden nebo více dialogových kroků. U většiny dotazů, pokud pomineme netypické příklady, je obsluha dialogového kroku poměrně složitá a protože je naše aplikace neustále rozšiřována /v současné době je v systému 30 aplikačních programů, z nichž některé obsluhují i několik obrazovek/, bylo nutné z důvodů údržby, změn, opravitelnosti a přehlednosti aplikačních programů přistoupit k rozdělení jednotlivých aplikačních programů na moduly. Toto bylo nutné už z toho důvodu, že většina aplikačních programů je psána v jazyce COBOL a obsluha jednoho dialogového kroku vyžaduje průměrně kolem 2000 řádků programu /nejsložitější až 4000-5000/, a program se v tomto případě stává nepřehledným. Postupem času se nám podařilo vytvořit systém, kde jednotlivé moduly mají přesně určené funkce a je definované rozhraní mezi jednotlivými moduly

Aplikační program má následující strukturu:



Popisy rozhraní

Ⓐ požadavky uživatele
Ⓑ data vyžadovaná uživatelem
Ⓒ naplněná výstupní zpráva pro uživatele
Ⓓ identifikace akčního programu, který bude zpracovávat následující zprávu /je podmíněn vypsanou zprávou/

Poznámky k jednotlivým částem aplikačního programu:

1/ Inicializace
Komunikační oblast uživatele, pokud není aplikační program aktivní, je uložena na disku. Aby se omezil počet I-O operací, je nutné zvážit délku obnovení komunikační oblasti a obnovit jen nutnou část.

2/ Přečtení zprávy
Dávat pozor na délku zprávy.

3/ Kontrola požadavků uživatele
Je nutné požadavek co nejlépe zkontrolovat, a to jak syntakticky, tak sémanticky, aby nedošlo ke zborcení aplikace v důsledku chyby /např. nenumerická položka v číselném poli atd./

4/ Zpracování požadavku
Je závislé na způsobu uložení dat, nejvíce ovlivňuje rychlost odezvy /viz kapitola 3/.

2.2.4. Výhody navrženého řešení

Navržené řešení výstavby aplikačního programu, tzn. jeho rozdělení do disjunktních částí řešících jednotlivé části dialogového kroku, má několik výhod

- jednotlivé moduly řeší části, které jsou nezávislé na ostatních modulech, jediným styčným bodem jsou přesně definované parametry
- zpracování jednoho typu požadavku a vytvoření výstupní zprávy provádí pouze jeden "plnící program" a "plnící modul", které

lze volat i z jiných aplikačních programů /zpětný chod, nekorektně vyplněné údaje a tudíž opakování dialogového kroku.

- při ladění lze funkci jednotlivých modulů /plnící program a plnící modul/ simulovat
- přehlednost, lehčí orientace

Uvedené principy jsme použili při dvou posledních rozšířeních aplikace /cca 10 aplikačních programů/ a jejich použití nám umožnilo daleko pohodlnější a rychlejší práci, i když se jednalo o velice složité dialogové kroky. Tento postup se ale vyplatí i při jednoduchých aplikačních programech, neboť pomáhá při analýze dialogových kroků.

3. Data při transakčním zpracování

Organizace dat a přístup k nim patří k stěžejním problémům každé transakční úlohy, neboť zásadně ovlivňuje jeden z nejdůležitějších parametrů - střední dobu odezvy.

Dobou odezvy rozumíme časový interval v reálném čase, který uplyne uživateli u terminálu od okamžiku, kdy odešle dotaz, až do okamžiku, kdy se mu na terminálu objeví odpověď. To znamená, že doba odezvy je totožná s dobou běhu aplikačního programu. Doba odezvy velice ovlivňuje komfort zpracování a podle posledních zjištění by neměla přesahovat 4 vteřiny. Pokud je zpracování dialogového kroku časově náročnější, je potřeba dát uživateli a terminálu na vědomí, že zadal časově náročný požadavek, že zpracování probíhá a eventuelně jak je daleko. Zároveň je nutné zabezpečit, aby dlouho běžící aplikační program neblokoval dlouho práci z ostatních terminálů, protože pokud je aplikace realizována jedním procesem, může být aktivní pouze jeden aplikační program. Je tedy nutné zpracování dlouhých dialogových kroků nějakým způsobem přerušovat a nebo navrhnout tak dlouhé dialogové kroky, aby doba odezvy zůstala v rozumných mezích.

Návrh organizace dat, vhodné pro ten určitý problém, je velice náročný a důležitý a jeho řešení by přesáhlo zaměření tohoto příspěvku. Je nutné si uvědomit, že návrh organizace dat z největší míry ovlivňuje úspěšnost navrženého řešení, které s ním v podstatě stojí a padá.

V našem případě jsme použili několik možných řešení a naše zkušenosti uvádím v dalších odstavcích.

4. Chování aplikace při chybě

I když je snahou každého programátora napsat program bez chyb, málokdy se to podaří. Je samozřejmé, že v aplikaci, která je velmi složitý objekt, může k chybě dojít. Chyba v aplikaci je nepříjemná z několika důvodů. Tak především: výskytu chyby je přítomen uživatel, to znamená člověk bez hlubších znalostí programování, který často není ani schopen popsat akci, která ke zhroucení aplikace vedla. Dalším důvodem je skutečnost, že možnosti nějaké diagnostiky chyby jsou v podstatě vyčerpány analýzou dumpu, který aplikace při svém abnormálním ukončení vytvoří.

Mnohé chyby jsou způsobeny tím, že ne všechny důležité údaje jsou v aplikačních programech uschovány do komunikační oblasti. Odhalit tuto chybu je velice náročné, neboť se projevuje naprosto neidentifikovatelně pouze v některých případech, a to pouze tehdy, kdy pracuje dva a více terminálů.

Chyby lze rozdělit do dvou základních skupin:
chyby zachycené
chyby nezachycené.

Nezachycené chyby - k nim patří například
- chyby v práci se soubory /např. čtení neotevřeného souboru, otevření neexistujícího souboru/. Tyto chyby sice vedou ke zhroucení aplikace, ale nejsou tak časté a jsou poměrně lehce odhalitelné a odstranitelné.

- chyby při běhu programu /nenumerická data, chyba adresace, alokace nepřidělené paměti apod./. Tento druh chyby patří mezi nejhůře odhalitelné, neboť jediným vodítkem je adresa, kde k chybě došlo. Vlastní příčiny chyby, zvláště pokud se jedná o asynchronní chybu, způsobenou prací z více terminálů, je nutno velice obtížně a za použití náročných prostředků analyzovat, což je mnohdy mnohadenní záležitost. Také tyto chyby vedou k okamžitému zhroucení aplikace. K jejich zachycení lze instalovat exitovou rutinu, ale zatím jsme to nepoužili.

Zachycené chyby - zachycenou chybou rozumíme chybu, která vznikla v chodu aplikačního programu a byla kontrolou odhalena. V žádném případě chybou nenazývám to, že uživatel špatně vyplnil údaje nutné pro zpracování, toto umí ohlídat aplikační program. Jedná se tedy o chybu uživatele a ne aplikace. Chybami aplikace lze nazvat pouze chyby, ke kterým dojde tím, že nebyly zachyceny chybně zadané údaje a nelze požadovanou akci provést a nebo odhalenou chybu v programu /např. nemám naplněny ty údaje, které očekávám/. K těmto chybám patří také neočekávaný návratový kód po volání funkcí UTM.

Při těchto chybách nastává problém, jak situaci, která k chybě vedla natolik dobře zdokumentovat, aby bylo zřejmé jak a kdy k chybě došlo a důvody chyby. Je také nutné připravit uživateli možnost pokračovat v práci, tzn. umožnit mu její opětovné zahájení. Zpráva, která byla uživateli předána musí obsahovat všechny údaje nutné k analýze chyby. Musí být vypsána v srozumitelném tvaru včetně návodu pro uživatele, jak s údaji zacházet a komu je předat.

5. Organizace dat při transakčním zpracování

Je nutno říci, že kritickým místem celé aplikace je ta část, která řeší přístup k datům na megnatickém disku. Co se týče obsluhy obrazovek, neměli jsme s programováním žádné problémy, i průměrný programátor je schopen po krátkém zaškolení psát aplikační programy. Poněkud jiná je situace při návrhu organizace dat.

Pro dosažení únosné doby odezvy je třeba minimalizovat počet diskových I-O operací. Velice zjednodušeně můžeme říci, že existují čtyři možné způsoby organizace dat

- sekvenční soubor - tento způsob lze použít pouze při velmi malém objemu dat a nebo sekvenčním prohledávání, to znamená, že najde své uplatnění ve velmi malém počtu řešení. Tento způsob organizace jsme nepoužili.
- indexsekvenční soubor - k datům lze přistupovat podle klíče i sekvenčně. Lze takto realizovat poměrně složité struktury dat. Výhodou je jednoduchost a také to, že není třeba školit programátory. Na druhé straně tento způsob poskytuje malý programátorský komfort, musí se programovat vše, včetně přípravy dat, ochrany a obnovy dat apod. V našem případě jsme tento způsob použili pro vytvoření upravené stromové struktury. Můžeme říci, že pokud je zřejmé, že požadavky uživatelů mají podobný či stejný charakter, je použití indexsekvenční organizace dat vyhovující a co se týče doby odezvy, i poměrně rychlé.
- soubor s přímým přístupem do paměti - v tomto případě sám programátor určuje, kde jsou na disku data umístěna. Ze všech uvedených způsobů organizace je přístup k datům nejrychlejší. Na druhé straně existují značné nevýhody. Data jsou "šitá na míru" problému a jakákoliv změna či rozšíření je velmi programátorsky náročné. Protože jakýkoliv systém časem vyžaduje třeba i malé změny a údržbu, je potřebné zvážit, zda je tato organizace nutná. Tento způsob jsme použili v jedné části, která je poměrně malá a přesto s ním byly a jsou problémy. Programy jsou velice málo čitelné a jakákoliv úprava vyžaduje úplné pochopení programu, což nebývá jednoduché.
- systém řízení báze dat - v našem případě je to UDS /Universal database system/, který je založen na hierarchicky - síťovém modelu. Pro tento způsob hovoří jeho schopnost implementovat s transakčním monitorem UTM, že existuje značný programátorský komfort, že jsou k dispozici prostředky pro ochranu a obnovu dat, další podpůrné programy jako generátor sestav, programy na údržbu báze dat apod. Na druhé straně -

vyžaduje zkušené a vyškolené programátory a bez hlubších znalostí jej nelze úspěšně využívat. Kritické je hlavně správné vytvoření struktury báze dat, neboť zásadně ovlivňuje dobu přístupu k datům. Tento způsob jsme použili ve dvou částech a dá se říci, že je velmi vhodné ho použít všude tam, kde se pracuje se stejnými daty v různých souvislostech a pohledech.

Velice důležité je udržet konzistenci dat, pokud se pracuje s jejich změnami. Pokud během zpracování dotazu dojde k abnormálnímu přerušení programu /výpadek sítě, porucha počítače/ je nutné mít prostředky pro obnovu původního stavu.

Také je nutné se zmínit o návrhu datové základny. Návrh by měl být prioritní a vycházet z požadavků uživatele, teprve na jeho základě by měl být navržen dialog. Opačný postup zpravidla neodolá dodatečnému požadavku uživatele, což ve svých důsledcích vede ke přeprogramování části, řešící obsluhu dat a někdy celé transakční úlohy.

Literatura:

[1] Martin J. 1973 Design of Man-Computer dialoges, Printice-hall 1973

[2] UTM Programmschnittstellen Beschreibung, Siemens AG 1982

[3] UTM Generierung und administration, Siemens AG 1982