# Úvod do prostředí ILE

Vladimír Župka

2008-08-28

## Historie

Koncept ILE (Integrated Language Environment) vznikl jako odpověď na stále se stupňující požadavky k otevřenosti systému AS/400. V roce 1994 do operačního systému přibyly funkce, které k tomu přispěly podstatnou měrou. Šlo o soustavu funkcí umožňující aplikačním programům provádět operace běžné v unixových systémech. Tím byl položen základ k převodu aplikací z prostředí Unix do prostředí AS/400. K převodům aplikací také skutečně došlo. Za všechny lze uvést aplikační systémy SAP/R3 nebo Baan. V té době systém (AS/400) neprovozoval přímo některý z unixových systémů, ale jen převedené aplikace. Funkce API (Application Programming Interface) byly totiž voleny tak, aby splňovaly specifikaci "Single Unix" (dříve Spec 1170) a specifikaci POSIX. Také nevyčerpávaly veškeré definice těchto specifikací, ale jen ty, které byly vhodné pro komerční výpočty. Od doby, kdy byly zavedeny tzv. logické oddíly (logical partitions) a sjednocen hardware s řadou RS 6000, provozují se na systému i pravé operační systémy AIX a Linux, ale to je jiné téma.

Hlavním programovacím jazykem v unixových systémech je jazyk C, proto byl podstatně zlepšen kompilátor tohoto jazyka, aby vyhovoval normám ANSI a ISO. Zároveň s tím bylo do systému zavedeno prostředí ILE (Integrated Language Environment - integrované jazykové prostředí). Toto prostředí bylo použito zejména ke spojování odděleně kompilovaných modulů, a to nejprve pro jazyk C.

Z ostatních jazyků se programy kompilovaly rovnou do objektu typu *PGM, který se dá spustit příkazem CALL (tzv. dynamické volání). Nebylo třeba žádného spojování modulů, poněvadž žádné neexistovaly. Tento způsob tvorby programů je označován jako OPM (Original Program Model). Podobný způsob vytváření programů (compatibility mode) trvá i nadále v rámci prostředí ILE.[1]

Po úspěšném využití s jazyky C a C++ byl koncept ILE zaveden s různým stupněm využití také do dalších programovacích jazyků, tedy RPG, Cobol a CL. Přišlo se totiž na to, že po určitých úpravách lze v těchto jazycích využívat bohatou zásobu unixovských funkcí obsažených systému. Z nich nejvíce možností poskytuje jazyk RPG, méně Cobol a nejméně CL.[2]

## Struktura ILE programu

ILE program vzniká spojením modulů (jednoho nebo několika), které jsou do něj buď přímo vkopírovány nebo volně připojeny v podobě servisního programu. Je důležité rozlišovat mezi programem (objektem typu *PGM) a servisním programem (objektem typu

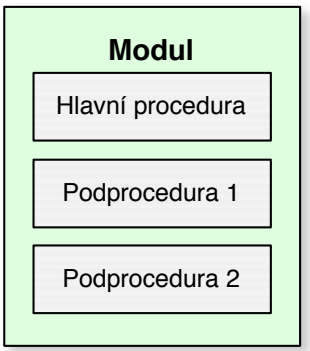
---

[1] Ve skutečnosti před zavedením programového modelu ILE existovaly dva modely: OPM (Original Program Model) a EPM (Extended Program Model). Programový model OPM se týkal jazyků RPG, Cobol a CL, zatímco model EPM se týkal jazyků Fortran, Pascal a C. Zatímco jazyky Fortran a Pascal ze Systému i prakticky zmizely, jazyk C zůstal a přibyl jazyk C++.

[2] Jazyk Java nepatří do prostředí ILE, protože jeho zdroje (.java) a třídy (.class) jsou zcela odlišné objekty. Nejsou uloženy v knihovnách, ale v adresářích IFS a kompilují se pomocí jazyka Qshell převzatého z unixových systémů.

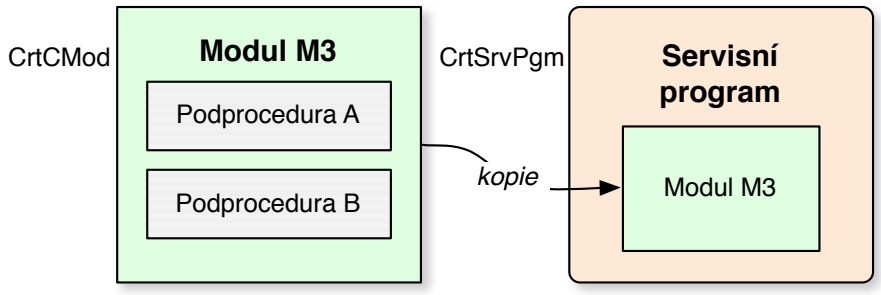
*SRVPGM). Servisní program, jak už je z názvu patrné, slouží k tomu aby poskytoval servis.

**Modul**
Modul je výsledkem kompilace a skládá se z procedur. Modul obsahuje obecně hlavní proceduru a vedlejší procedury, čili podprocedury (o nich níže). Podle okolností může chybět hlavní procedura nebo mohou chybět podprocedury. Modul je objekt typu *MODULE a je základním stavebním kamenem při sestavování programu nebo servisního programu. Vzniká kompilací zdrojového textu daného programovacího jazyka. Kompilační příkazy se liší podle programovacího jazyka: CrtRpgMod (RPG), CrtCblMod (Cobol), CrtCMod (C), CrtCppMod (C++), CrtClMod (CL).

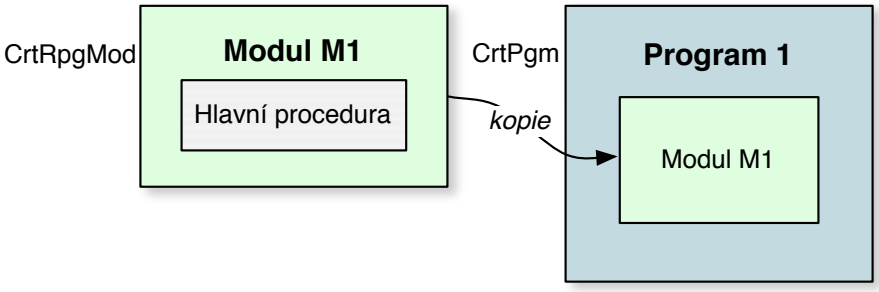



Aby modul mohl fungovat, je třeba jej zakomponovat nějakým způsobem do hlavního programu. Děje se tak buď přímým kopírováním nebo nepřímo prostřednictvím tzv. servisního programu.

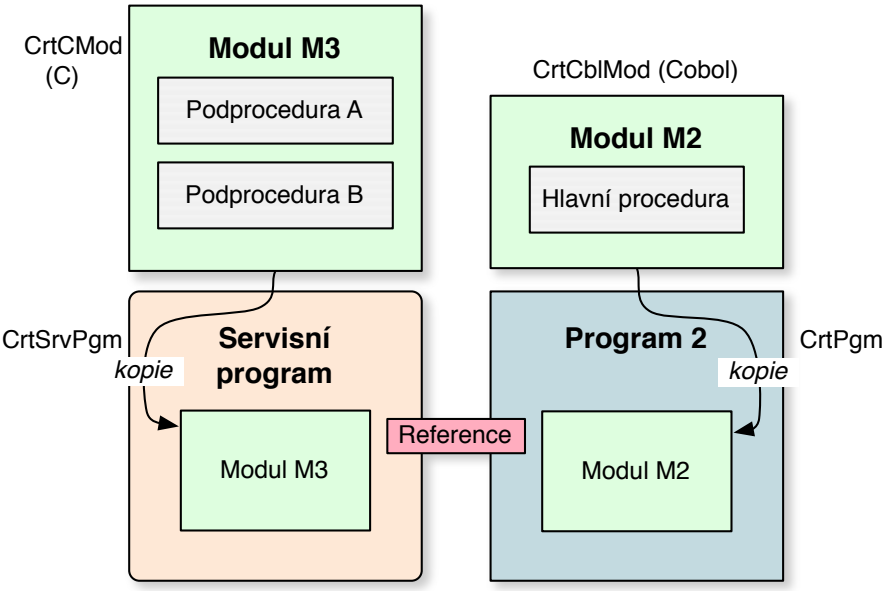
**Servisní program**
Servisní program je objekt typu *SRVPGM a vzniká spojením modulů. Vytváří se čili spojuje příkazem CrtSrvPgm. Na obrázku je jen jeden modul vzešlý z jazyka C, který obsahuje dvě podprocedury. Příkaz CrtSrvPgm vytvoří servisní program tak, že do něj modul přímo včlení.




**Program**
Program je objekt typu *PGM a skládá se z modulů. V jednodušším případě se moduly kopírují (bind by copy) přímo do vytvářeného programu. Na obrázku je jen jeden modul vzešlý z jazyka RPG a obsahující pouze hlavní proceduru. Příkaz CrtPgm vytvoří program tak, že modul do něj přímo včlení.



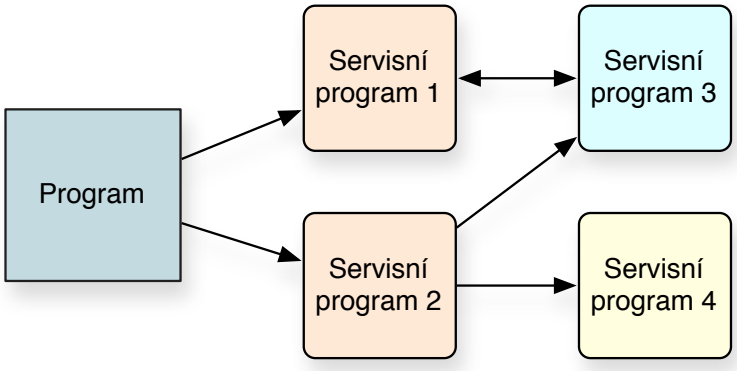

Obecně se moduly připojují ke vznikajícímu programu ještě prostřednictvím servisních programů. Servisní programy se do programu nekopírují přímo, ale vážou se pomocí referencí (bind by reference). Program se vytváří čili spojuje příkazem CrtPgm. Jméno příkazu je stejné, ale jeho parametry určují také servisní program, který se má připojit. Na obrázku program vzniká přímým připojením modulu M2 (z Cobolu) a referenčním připojením servisního programu z dřívějšího obrázku.



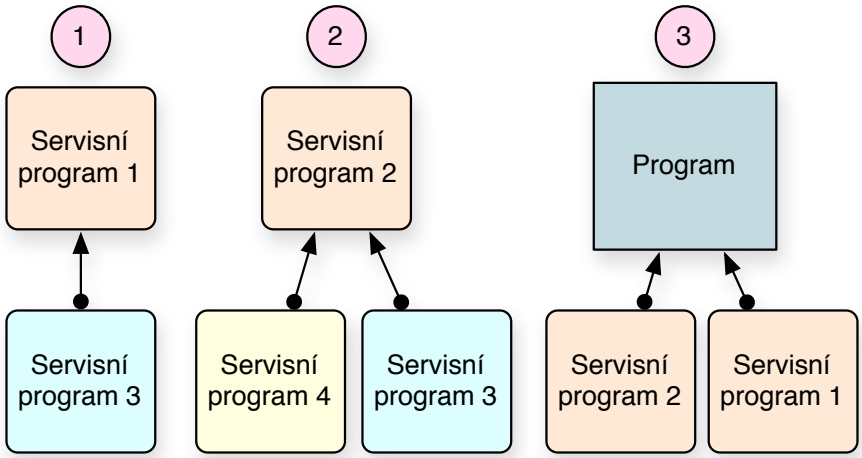

V době vytváření programu příkazem CrtPgm se do programového objektu dostanou důležité údaje z objektu servisního programu. Jsou to údaje o skladbě procedur a verzi servisního programu, které jsou důležité v době, kdy se program spouští.

**Kombinace servisních programů**

Do programu lze připojovat nejrůznější kombinace servisních programů. V obrázku program volá procedury v servisních programech 1 a 2, procedury v servisních programech dále volají procedury v ostatních servisních programech.

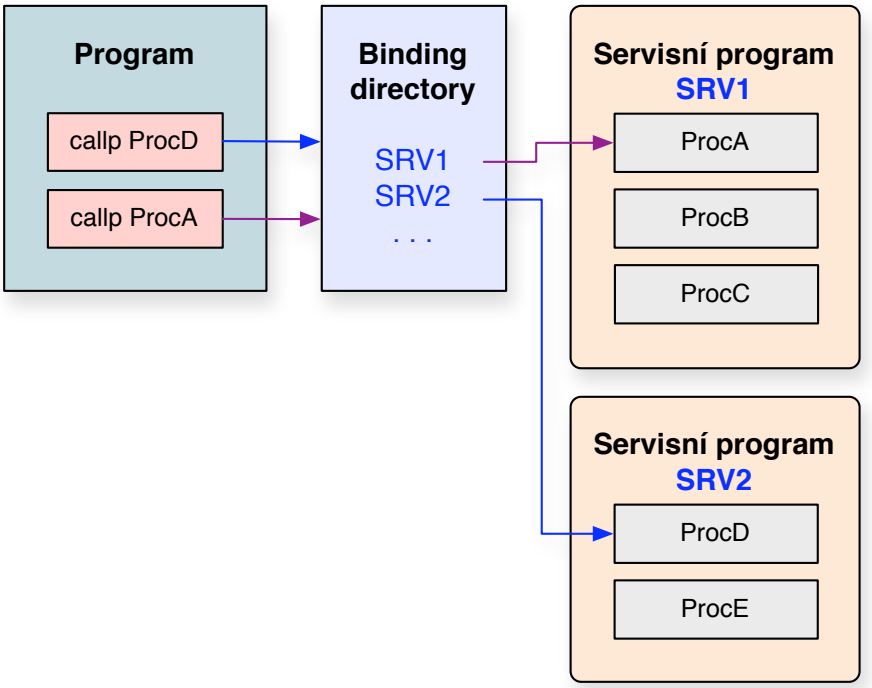



Postup spojování programu z obrázku je naznačen na dalším obrázku. Nejprve je třeba vytvořit servisní programy 1 a 2 ze servisních programů 3 a 4, pak teprve program ze servisních programů 1 a 2.




**Spojovací seznam - binding directory**
Kromě jednotlivých modulů a servisních programů je dalším zdrojem procedur tzv. spojovací seznam (binding directory). Jde o objekt typu *BNDDIR, který se vytváří zvláštními příkazy (CrtBndDir, AddBndDirE) a obsahuje seznam servisních programů a modulů. Při spojování příkazem CrtPgm nebo CrtSrvPgm lze zadat jeden nebo více spojovacích seznamů, z nichž se vyberou jen ty položky, v nichž se vyskytují volané procedury.

V následujícím obrázku znamená callp statické volání procedury, kterému se v souvislosti se spojováním říká import. Proceduře v servisním programu, která je poskytnuta k dispozici navenek, se říká export.




**Spojovací program - binder**
Příkazy CrtSrvPgm a CrtPgm jsou varianty spojovacího programu (binder). Spojovací program je už nezávislý na programovacím jazyku, což umožňuje snadno spojovat moduly vytvořené v různých jazycích. Tomu odpovídá název ILE - prostředí pro integraci jazyků.

# Procedury, parametry a prototypy

Procedura je nový pojem v ILE. Rozlišuje se hlavní procedura a podprocedura. Tam, kde to nevadí, se tyto pojmy nerozlišují a mluví se prostě o procedurách. Hlavní proceduře se také říká hlavní program.

Pojem procedury zhruba odpovídá pojmu funkce v jazyku C. Tam se také rozlišuje hlavní funkce, která se dokonce musí jmenovat main, od ostatních funkcí. Hlavní funkce odpovídá hlavní proceduře v názvosloví ILE. Ostatní funkce jsou podprocedury. Zápis funkce v jazyku C se nazývá definice. Popis jejího rozhraní, tj. parametrů a funkční hodnoty se nazývá deklarace nebo také prototyp. Zdrojový soubor může obsahovat několik funkcí, nebo jen jedinou. Hlavní funkce main není povinná. Stejné je to pochopitelně i v jazyku C++.

V podstatě stejné je to v jazyku RPG, kde pro popis podprocedury je nutné zapsat kromě její definice také prototyp. Zdrojový program může obsahovat několik procedur nebo jen jedinou. Hlavní procedura není povinná.

V jazyku Cobol odpovídá  podproceduře nested program, tedy vnořený program. Hlavní program odpovídá  hlavní proceduře v názvosloví ILE a musí být vždy přítomen. Vnořené programy jsou podprocedury. Hlavní program i vnořený program se nazývají také ILE procedure na rozdíl od cobolské procedury - procedure division. Zde může docházet k záměně pojmů. V jazyku Cobol není zaveden popis prototypu na rozdíl od jazyka C nebo RPG.

V jazyku CL je program hlavní procedurou i podprocedurou. Zdrojový program může totiž obsahovat jen jednu proceduru. Podle toho, v jakém kontextu se použije, se stane buď hlavní procedurou nebo podprocedurou. V jazyku CL rovněž není zaveden popis prototypu.

**Účel prototypu v jazycích C, C++ a RPG**
Prototyp slouží k tomu, aby se již při kompilaci modulu zkontrolovala správnost volání podprocedury (tedy složení a typy parametrů a vracené hodnoty). To znamená, že prototyp musí být zapsán v každém modulu, kde se daná podprocedura vyvolává.

**Dynamické a statické volání**
Program voláme způsobem, který se nazývá dynamické volání - dynamic call. Protože program je objekt, musí operační systém při jeho volání vykonat mnoho akcí: nalézt objekt v knihovnách, ověřit oprávnění, zavést do paměti, inicializovat proměnnné apod. To vše značně zdržuje výpočet.

Podproceduru naopak voláme způsobem, kterému se říká statické volání - static call. Nazývá se tak proto, že při něm je už z doby kompilace a spojování známá adresa volané procedury a je zkontrolována správnost skladby parametrů (argumentů) i vracené hodnoty. Pochopitelně se nemusí kontrolovat ani oprávnění. Tím se významně zrychluje výpočet ve srovnání s dynamickým voláním programu.

**Využití funkcí z jazyků C a C++**
Funkce dostupné v jazyku C a C++ lze využít také v ostatních ILE jazycích. K tomu musíme zjistit, jaké potřebují parametry a jaké vracejí hodnoty, tj. jaké je rozhraní funkce. V programové dokumentaci jazyků C a C++ jsou tyto věci popsány velmi podrobně a k tomu jsou uvedeny tabulky k převodu datových typů do jiných jazyků. V dokumentaci volání API pro unixové systémy (Unix-type APIs), se rozhraní funkcí popisuje v jazyku C. Kdo se orientuje v jazyku C, má při převodu výhodu.

Aby bylo možné funkce použít v jazyku RPG, musí se do něj převést prototypy i parametry volání. Pro jazyk RPG lze na internetu nalézt dostatek prototypů pro různé funkce.

I když v jazycích Cobol a CL se prototypy nepoužívají, přesto je potřeba znát skladbu parametrů a typ vracené hodnoty. Zde již tolik informací na internetu není, takže zbývá jen “ruční” převod z jazyka C.

# Servisní program
Význam servisních programů spočívá v tom, že v nich můžeme soustředit vícenásobně použitelné procedury. Servisní program se pak může připojit k několika hlavním programům, což umožňuje napsat každou proceduru jen jednou a využít ji v různých programech.

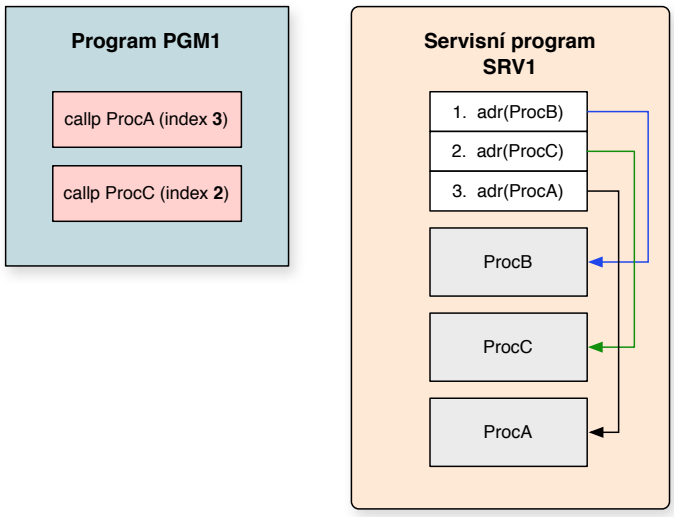
**Volba servisních programů**
Složení servisních programů lze volit libovolně, ale je vhodné zavést nějaký systém. Například můžeme do jednoho servisního programu umístit nejjednodušší a nejvíce používané procedury, např. konverze data mezi různými formáty, operace s datem, zjišťování údajů ze systému apod. Do druhého servisního programu umístíme procedury, které provádějí některé jednodušší aplikační operace a běžně používají procedury z prvního servisního programu. Například výpočet penále z nezaplacené faktury apod. Do třetího servisního programu zařadíme složitější aplikační procedury, které jsou však stále

vícenásobně používané a volají procedury z předchozích servisních programů. Takto lze postupovat dále. Servisní programy se pak připojí k programům jednotlivě nebo prostřednictvím spojovacího seznamu - binding directory.

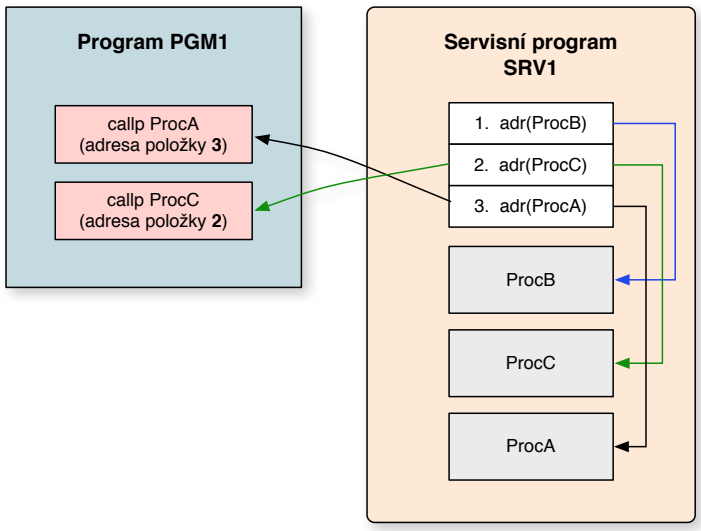
**Vazba referencí**

Servisní program SRV1 na obrázku je vytvořen příkazem CrtSrvPgm z jednoho modulu obsahujícího tři podprocedury poskytnuté k použití “veřejnosti”. V něm je pak k dispozici tabulka, kde jsou uvedeny relativní adresy jednotlivých procedur. Tato tabulka představuje reference čili odkazy na procedury.

Když tento servisní program spojujeme příkazem CrtPgm s programem PGM1, který volá některé z těchto procedur, spojovací program si zjistí ke každému volání index, který odpovídá referenci na danou proceduru. Podle následujícího obrázku odpovídá volání procedury ProcC tabulkový index 2, volání procedury ProcA index 3. V objektu programu PGM1 nejsou uvedena jména, ani adresy procedur, ale jen tabulkové indexy.




Při prvním spuštění programu PGM1 v rámci úlohy proběhne tzv. aktivace, při níž se do paměti zavede objekt servisního programu SRV1, do jehož tabulky se dosadí absolutní adresy procedur a do programu se místo indexů dosadí adresy tabulkových položek (ne však adresy začátků procedur). Tím je servisní program připoután k programu a od této chvíle jsou podprocedury volány staticky. Následující obrázek ukazuje program PGM1 po aktivaci.



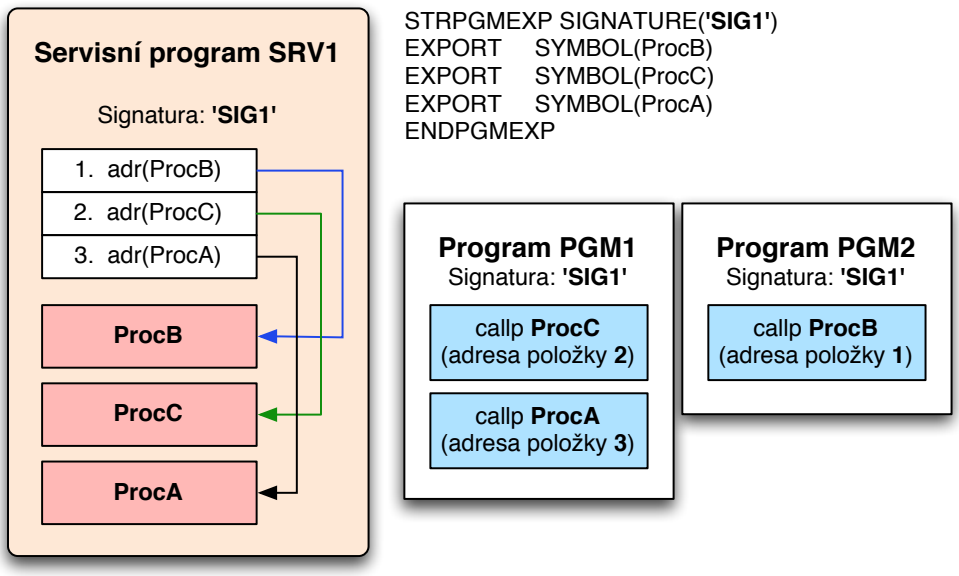

**Signatura a spojovací zdrojový text**
Servisní programy při vytváření získávají tzv. signaturu. Signatura je 16bajtový údaj vyjadřující určitou verzi servisního programu. Je to něco podobného jako úroveň (level) u databázových souborů. Signatura představuje prostředek, jak kontrolovat přístup k servisnímu programu z programů (nebo jiných servisních programů). Důvodem zavedení signatury je skutečnost, že servisní program se vytváří nezávisle a jeho skladba se může časem změnit, což se týká zejména jeho exportů (tedy zejména skladby podprocedur).

Signatura se vytvoří buď automaticky, nebo ji můžeme určit sami tak, že zadáme tzv. spojovací zdrojový text - binder source. Jde o jednoduchý specifikační jazyk, kterým určíme, zhruba řečeno, které procedury servisního programu může spojený program používat. Tento text slouží zejména při údržbě programového vybavení a jeho verzí. Zejména je užitečný při doplňování servisních programů o nové procedury, kdy při správném postupu odstraňuje nutnost nového spojování všech aplikačních programů, které daný servisní program používají. Servisní program může obsahovat více signatur odpovídajících různým verzím.

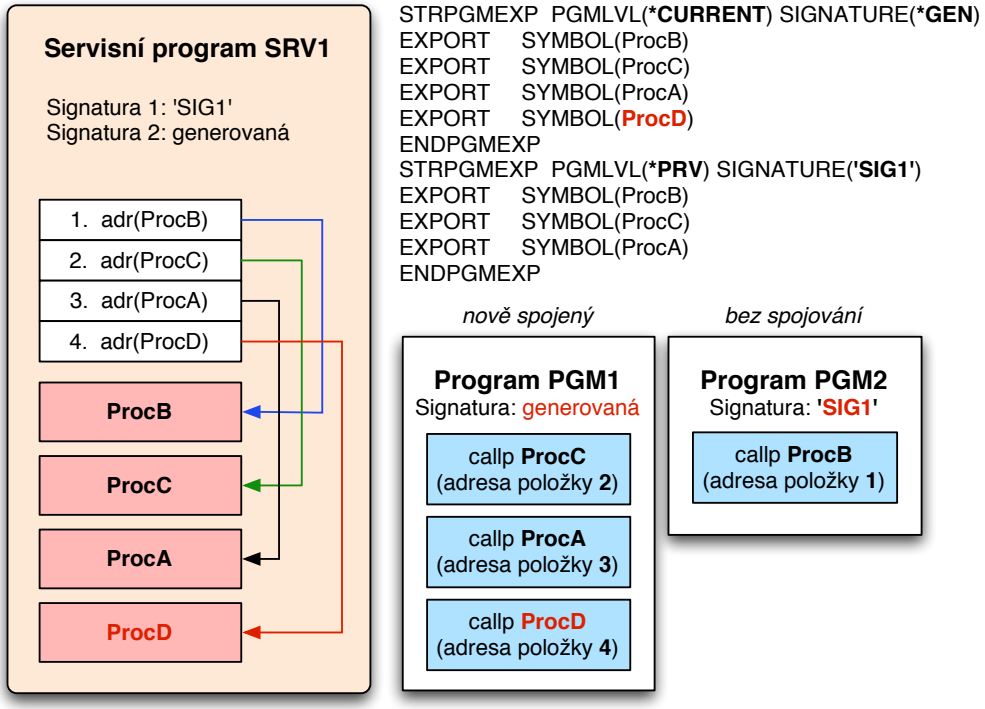
Při spojování programu příkazem CrtPgm se prohlíží každý připojovaný servisní program. Jeho poslední signatura zároveň s jeho jménem a jménem knihovny se uloží do programu. Později, při aktivaci programu, systém zkoumá, zda signatura obsažená v programu se shoduje alespoň s jednou ze signatur obsažených v připojovaném servisním programu. Jestliže ano, program se spustí. V opačném případě skončí chybou.

**Údržba servisního programu pomocí spojovacího textu**
Pomocí spojovacího textu lze udržovat servisní program, aniž je zapotřebí provádět jeho nové připojování k programům, které jej používají.

Při prvním vytváření servisního programu SRV1 vznikne na základě spojovacího textu první signatura 'SIG1', kterou si také při spojování poznamenají všechny programy, které tento servisní program používají. Program PGM1 používá procedury ProcA a ProcC. Program PGM2 používá proceduru ProcB. V obrázku je uvedena první verze spojovacího zdrojového textu (připomíná jazyk CL, ale nekompiluje se).




V další fázi údržby je chceme doplnit do servisního programu další proceduru - ProcD, kterou potřebuje program PGM1. Spojovací text upravíme tak, že dosavadní text zkopírujeme, kopii umístíme na konec a označíme *PRV (předchozí). Původní text doplníme o proceduru ProcD a blok označíme *CURRENT. Doplníme ještě specifikaci signatury *GEN, což způsobí, že při novém vytváření servisního programu vygeneruje novou signaturu systém a přidá ji k dosavadní signatuře 'SIG1'. Druhá verze servisního programu bude tedy obsahovat dvě signatury.




Program PGM1 musíme nově přeložit a spojit se servisním programem, protože používá novou proceduru. Při spojování se do programu PGM1 uloží nová, generovaná signatura

servisního programu. Program PGM2 však můžeme nechat beze změny, protože nic nového ze servisního programu nepotřebuje.

Je důležité si uvědomit, že servisní program je v operační paměti uložen jen jednou, i když jej používá více programů.

# Poznámky k údržbě programů

Při tvorbě aplikací se přirozeně snažíme psát vícenásobně použitelné funkce jako samostatné jednotky a připojovat je nějakým vhodným způsobem k programům, které je využívají. Lze to činit různě, zde si všimneme některých způsobů obvyklých v Systému i.

**Vícenásobně použitelné programy volané dynamicky**

Obecně použitelné funkce můžeme realizovat tradičně jako samostatné programy a volat je dynamicky. Změny v takových programech nevyžadují zásahy do programů, které je volají (za předpokladu, že se nemění skladba parametrů volání). Údržba je tedy snadná. Dynamické volání programů však spotřebuje mnohem více času a systémových prostředků než statické volání procedur.

**Subrutiny v RPG**

V jazyku RPG je běžné realizovat obecně použitelné funkce jako subrutiny, které se zapíší do zvláštního zdrojového členu a při kompilaci se vkopírují do zbytku programu na příslušné místo pomocí povelu /INCLUDE. V subrutinách ovšem nelze použít parametry volání, ani vracenou hodnotu a při každé změně je nutné všechny dotčené programy znovu zkompilovat. Údržba je tedy náročná na evidenci programů používajících takové subrutiny.

**Přímo začleněné moduly v ILE programech**

Podobná potíž nastává, když vytvoříme spojený ILE program prostým začleňováním modulů. Při změně funkčnosti podprocedury v určitém modulu bychom museli zjistit, ke kterým programům se dotyčný modul připojoval, a museli bychom provést nové spojení u všech těchto programů. Kdybychom to neprovedli u všech, hrozilo by, že některé programy budou volat novou verzi procedury, zatímco jiné by volaly starou verzi. Rovněž v tomto případě je údržba náročná na evidenci programů s takto připojenými moduly.

**Servisní programy s ILE programy**

Servisní programy jsou při údržbě programového vybavení nejvýhodnější. Když je například potřeba změnit funkčnost některé procedury, stačí pak jen zkompilovat modul, kde je procedura zapsána, a vytvořit znovu servisní program.

Můžeme dokonce přidávat nové procedury. Vhodným doplněním odpovídajícího zdrojového spojovacího textu (binder source) při vytvoření servisního programu docílíme toho, že stačí spojit doplněný servisní program jen s těmi programy, které nové procedury skutečně volají. Dosavadní procedury může používat řada jiných programů, kterých se tato změna vůbec nedotkne.

# Aktivační skupiny

Aktivační skupina (activation group) je podstruktura úlohy (job substructure). V dokumentaci se dočteme, že zajišťuje oddělený provoz programů v rámci jedné úlohy tak, aby se vzájemně neovlivňovaly.

Poznámka: Aktivační skupiny jsou také použity při řízení vláken - threads. Vlákna jsou potřebná k podpoře jazyků C, C++, Java a mechanismů IFS (Integrated File System), čímž se zajišťuje provoz převzatých unixových aplikací i některých nových aplikací vyžadujících paralelní zpracování podřízených procesů, tzv. multithreading. Tradiční aplikace žádný multithreading nevyužívají.

Aktivační skupina je pevně spojena s objektem ILE programu nebo servisního programu. Definuje se tudíž v příkazech CrtPgm a CrtSrvPgm. Každý ILE program se aktivuje a běží uvnitř nějaké aktivační skupiny.

Aktivační skupina obsahuje všechny prostředky nutné k provozu programu. Z těchto prostředků jmenujme paměť pro proměnné a otevřené přístupové cesty k databázovým souborům (ODP - open data path). Otevřená přístupová cesta je v podstatě buffer otevřeného souboru s ukazatelem na záznam (v SQL se nazývá cursor).

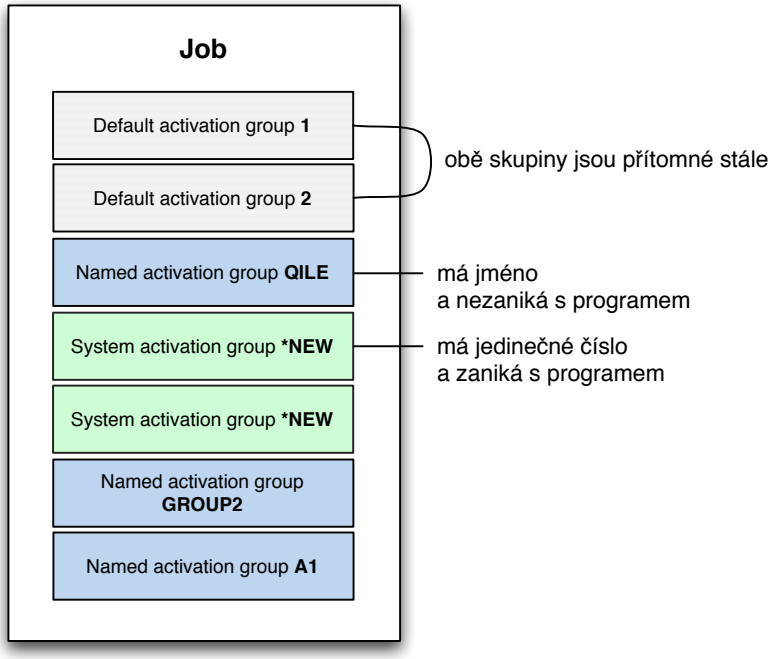
**Druhy aktivačních skupin**
Podle toho, jak se používají, jsou aktivační skupiny tří druhů: předvolená (default), pojmenovaná (named) a systémová (*NEW). V rámci úlohy (jobu) může existovat mnoho aktivačních skupin různých druhů, podle toho, jak se programy postupně volají. Aktivační skupiny lze sledovat, zobrazíme-li zásobník volání (call stack) aktivní úlohy.

Předvolená aktivační skupina (default activation group) je určena k provozu systémových programů a těch uživatelských programů, které nevyužívají vlastností ILE, zejména podprocedur. Vyvolaný program běží stále v této aktivační skupině, která nikdy nezaniká, dokud existuje úloha (job). Ve skutečnosti existují dvě předvolené aktivační skupiny. Skupina s číslem 1 hostí systémové programy, např. program QCMD, kdežto aplikační programy běží ve skupině s číslem 2.

Pojmenovaná aktivační skupina určená jménem v parametru ACTGRP(jméno) při vytváření programu příkazem CrtPgm, popř. servisního programu příkazem CrtSrvPgm. Při vyvolání (aktivaci) programu se tato aktivační skupina vytvoří a zůstává v úloze, i když všechny programy skončí svou činnost.

Systémová aktivační skupina určená parametrem ACTGRP(*NEW) při vytváření programu příkazem CrtPgm. Systém jí určí interní jméno tak, aby se neshodovalo s jiným. Při každém vyvolání programu se vytvoří nová aktivační skupina, která s ukončením programu zanikne. Této skutečnosti lze využít k rekursivnímu volání programu (program volá sám sebe) nebo k snadnějšímu ladění programů.

Na obrázku je několik aktivačních skupin, které odpovídají postupně volaným programům tak, že v každé skupině kromě *NEW může být zahrnuto několik programů.

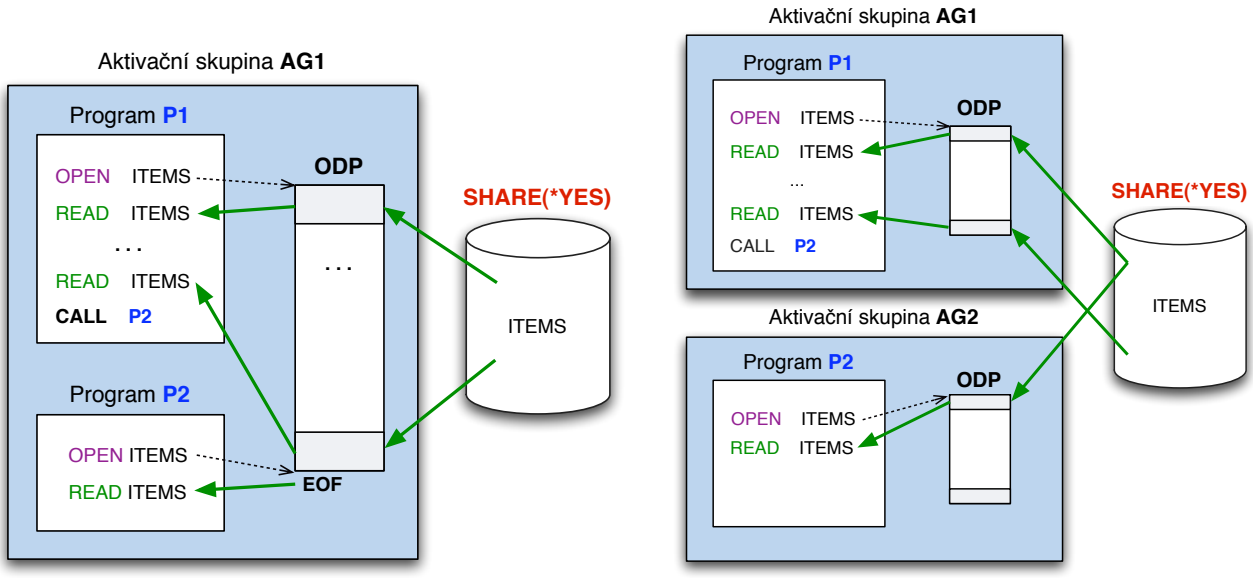



## Účinek aktivačních skupin

V následujících příkladech ukážeme účinky, jaké mají aktivační skupiny na provoz programů v rámci úlohy (jobu). Dvojice programů, z nichž jeden volá druhý, čte a tiskne záznamy ze stejného databázového souboru, který je definován pro sdílení parametrem SHARE(*YES). Příklady ukazují, jak se programy rozdílně chovají, běží-li ve stejné aktivační skupině nebo ve dvou různých aktivačních skupinách.

## Programy ve stejné aktivační skupině

Na obrázku vlevo jsou dva programy, P1 a P2, které mají společnou aktivační skupinu pojmenovaou AG1. Oba programy čtou sekvenčně databázový soubor ITEMS, jehož přístupová cesta (ODP - open data path) se otevírá jako sdílená, tedy s parametrem SHARE (*YES). Program P1 přečte celý soubor a vyvolá program P2, který tentýž soubor čte znovu. Program P2 ale nepřečte žádný záznam, protože program P1 nechal soubor nastavený na konci. Je to tím, že oba programy sdílejí stejnou otevřenou cestu k souboru a stejnou aktivační skupinu. Představíme-li si, že program P2 pochází od jiného dodavatele než program P1, mohli bychom nabýt dojmu, že je chybný. K nápravě by v tomto případě stačilo určit oběma programům různé aktivační skupiny.




**Programy v rozdílných aktivačních skupinách**
Na obrázku vpravo má program P1 opět aktivační skupinu AG1, ale program P2 má nyní aktivační skupinu AG2. Oba programy zase čtou soubor ITEMS od začátku do konce. Otevřená cesta k souboru ITEMS je také sdílená. Tentokrát má však každý program k dispozici svou vlastní otevřenou přístupovou cestu (ODP), takže po otevření přečte každý program první záznam souboru ITEMS.

# Závěr

V článku byly principy konceptu ILE jen naznačeny beze snahy objasnit podrobnosti. Tematika ILE je značně rozsáhlá, dokumentace IBM se jí věnuje velmi podrobně. Nejdůležitější příručka se nazývá ILE Concepts a má označení SC41-5606-07. Je koncipována jako společná pro všechny programovací jazyky. V příručkách pro jednotlivé jazyky jsou uvedeny další podrobnosti a příklady.

Dobrým příkladem použití ILE je softwarový balík CGIDEV2 psaný v jazyku RPG, který je zdarma k dispozici na stránce www.easy400.net. Je určen k tvorbě dynamických webových aplikací pomocí HTML stránek na straně prohlížeče a CGI programů v jazyku RPG nebo Cobol na straně serveru.

Probrali jsme pojmy jako modul, servisní program, ILE program, procedura, prototyp, statické volání, spojovací seznam (binding directory), spojovací zdrojový text (binder source), aktivační skupina (activation group). Naznačili jsme, jakými způsoby lze spojovat moduly do programů a k čemu jsou dobré servisní programy při údržbě aplikací. Vyhnuli jsme se však popisu příkazů a ukázkám programů, aby článek nebyl nadměrně dlouhý.

Doufejme, že článek představuje přehledný úvod pro programátory a systémové inženýry, kteří potřebují s prostředím ILE pracovat, ale ještě o něm nic nevědí.