Zpracování souběžně pracujících úloh programovacím jazykem Modula - 2

Radek Garzina

VŠB-TU Ostrava, Katedra měřicí a řídicí techniky, FEI VŠB TU, 17. listopadu, 708 33 Ostrava

Abstract
The pseudoparallelism and the quasiparallelism represent two available methods of concurrent programming on monoprocessor computer´s architecture. The paper describes both methods of parallelism and their realization with programming language Modula-2.

1. ÚVOD

V následujícím textu se budeme zabývat problematikou zpracování souběžně pracujících úloh. Slovo souběžně je použito jako překlad anglického „ concurrent" (Concurrent programming, concurrent processing ...). Concurrent programming, neboli technika programování souběžně pracujících úloh, je technikou tvorby programů, které umožňují provádět několik činností jakoby v témže časovém okamžiku. S takovými systémy se setkáváme denně. Jsou to např. víceuživatelské systémy mezi něž patří systémy pracující pod OS UNIX, které umožňují plnohodnotnou práci na několika terminálech současně, dále pak transakční systémy zabývající se určitým typem úlohy, k níž má přístup několik uživatelů - zákazníků (bankovní systémy, rezervační systémy) a v neposlední řadě systémy řízení technologických pochodů. Posledním druhem systému se nyní zabývejme podrobněji.

2. PARALELISMUS

Jednoprocesorové počítače pracují sekvenčně a většina programovacích jazyků dovoluje formulovat výpočetní postupy jako sekvenčně prováděné posloupnosti operací. Souběžně pracující programy však mají části, kterým říkáme PROCESY, které mohou pracovat souběžně. Procesy však sami od sebe jsou sekvenční. Souběžné provedení procesů pak může být skutečně paralelní nebo pouze quasiparalelní. Definujme nyní přesněji pojem proces.

2.1 Proces

Proces je část programu nebo systému (pak procesem může být také program), která je prováděna přísně sekvenčně, ale která může být provedena souběžně, paralelně s ostatními procesy programu.

2.2 Rozdělení počítačových architektur z hlediska paralelismu

Souběžně pracující programy můžeme ve své podstatě provozovat na dvou základních počítačových architekturách:

- monoprocesorové architektury - paralelně na nich provozované systémy jsou označovány jako systémy pracující ve sdílení času (time sharing) nebo multiprogramové systémy. Vlastní procesy jsou pak nazývány slovem PROCES, TASK nebo THREAD.
- víceprocesorové architektury - v těch jsou procesy prováděny na oddělených procesorech.

Podle možnosti přístupu ke společné paměti je pak rozdělujeme na další dvě skupiny zpracování - Multiprocessing (procesy mohou přistupovat ke společné sdílené paměti a sami mohou mít vlastní lokální paměť) a Distribuované zpracování - distributed processing (procesy jsou prováděny na oddělených procesorech bez možnosti přístupu ke společně sdílené paměti. Paměť, pouze lokální, mají jednotlivé procesory a výměnu dat je nutno provádět pomocí zasílání zpráv po komunikačních kanálech. Typickým představitelem takovýchto systémů jsou systémy založené na transputerech).

3. PARALELISMUS NA JEDNOPROCESOROVÝCH ARCHITEKTURÁCH

Zaměřme se na problematiku návrhu souběžně pracujících úloh na jednoprocesorových architekturách. Tyto systémy zpracovávají souběžně pracující úlohy metodou sdílení času procesoru. Procesům, taskům nebo vláknům - „threadům" je procesor přidělen na stanovený časový interval. Podle způsobu přidělení procesoru jednotlivým procesům lze paralelní zpracování na jednoprocesorových architekturách rozdělit na dva základní druhy:
- pseudoparalelismus
- quasiparalelismus

3.1 Pseudoparalelismus

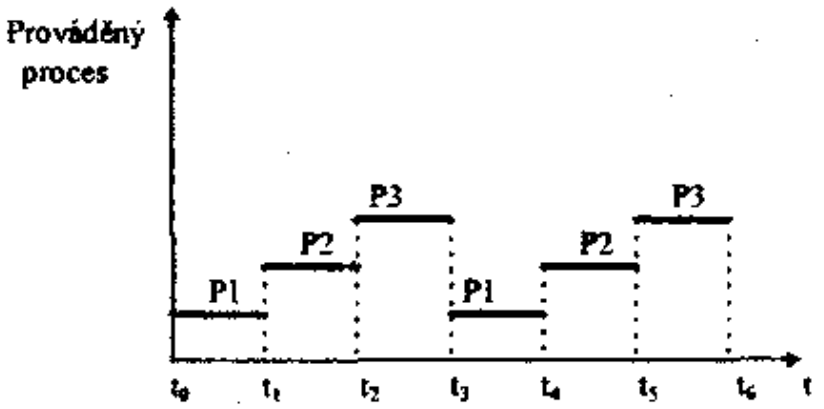
Tento druh paralelního zpracování je charakteristický dvěmi základními vlastnostmi:
- v daném okamžiku je prováděn pouze jeden proces
- přidělování času procesoru jednotlivým procesům je řízeno z vnějšku - hodinovými impulsy časovače

Proces je tedy zpracováván po částech, vždy po stanovený časový interval. Po uplynutí tohoto intervalu je procesor předán dalšímu procesu. Správnou činnost přepínání jednotlivých procesů řídí speciální procedura, kterou nazýváme schedulerem, kernelem. Tato procedura je dalším procesem navrhovaného systému. K zajištění bezchybného chodu celého systému tento proces uchovává pro každý proces tyto informace:
- proměnné náležející danému procesu
- prioritu a status daného procesu
- obsah registrů procesoru v případě, kdy proces není zpracováván procesorem

Všechny tyto informace souhrnně nazýváme kontextem procesu. Kontext každého procesu je ukládán při nečinnosti schedulerem do zásobníkové paměti příslušného procesu. Přechod od zpracování jednoho procesu ke druhému pak nazýváme přepnutí kontextu.

Povely scheduleru k přepnutí kontextu vydává časovač, který v přesných časových intervalech generuje přerušení vedoucí k požadovanému předání procesoru jinému procesu. Scheduler musí udržovat „banku" všech kontextů přerušených procesů. Po výběru vhodného procesu pro další zpracování tak zajistí přidělení a obnovení správného kontextu. Procedura scheduleru je součástí operačního systému nebo speciálních knihoven v případě, nepracujeme-li pod operačním systémem reálného času.

Grafické vyjádření přepínání procesů znázorňuje Obr.1:



Obr.1 Pseudoparalelní zpracování tří procesů

3.1.1 Synchronizace pseudoparalelního procesu

Pojem pseudoparalelismus je úzce spjat s problémem časové závislosti. Při zpracovávání paralelních procesů se mohou vyskytovat dvě základní situace: buď jsou procesy vzájemně nezávislé, tzn. nepodílejí se o společná data, zařízení apod. nebo je tomu naopak - existuje určitý společný objekt, jehož společným používáním se mohou akce procesů vzájemně ovlivňovat. Tuto negativní situaci je nutno velmi pečlivě řešit. Problém řešíme pomocí tzv. *SYNCHRONIZACE* procesů, která vede k odstranění časově závislých chyb. Úkolem synchronizace je tedy určitá koordinace postupu jednotlivých procesů. Docílíme ji doplněním procesů o synchronizační operace, kterých je několik druhů. Vysvětlíme si podstatu alespoň těch nejzákladnějších synchronizačních nástrojů. Metodami synchronizace jsou:

- *metoda čekání* - představující nejjednodušší synchronizační nástroj. Jde o synchronizaci v čase absolutní, která způsobí pozastavení procesu na daný počet časových jednotek. Použití této instrukce je však problematické, neboť požaduje znalost toho, na jak velký časový okamžik je nutno proces odložit
- *metoda čekání realizována procedurou TestAndSet* - každý proces prostřednictvím této procedury využívá společnou proměnnou, která signalizuje stav obsazení společného zdroje. Tato proměnná je logického typu.
- *metoda maskování přerušení* - je založena na vydání zákazu přerušení vůbec. Tím je znemožněno také přerušení hodinovým signálem časovače. Nevýhodou tohoto způsobu synchronizace je maskování všech přerušovacích signálů.
- *metoda uzamčení procesu* - na rozdíl od předešlého způsobu zajistí fungující přerušovací systém. Zámek má celkem dva možné stavy, buď je zámek otevřen nebo uzamčen. Začne-li proces využívat společný zdroj, „uzamkne" si procesor

pro svou vlastní potřebu, čímž zamezí zpracování dalšího procesu procesorem. Tento odložený proces musí čekat tak dlouho, dokud aktivní proces „neodemkne" procesor ostatním procesům. Nevýhodou této metody je odložení všech procesů, tzn. i procesů, kteří nevyužívají společného zdroje.

- *semafor* - metoda založena na použití dvouhodnotové proměnné, jejímž úkolem je udržovat informaci o stavu ji přiřazeném společném zdroji, tzn. informaci o tom, zda je zdroj využíván nebo ne.
- *signál* - metoda řešící problém, kdy jeden z procesů musí čekat než jiný proces provede akci, na jejímž základě může teprve pokračovat v činnosti. Tato metoda je založena na třech procedurách vysílající , čekající a inicializující daný signál.

3.2 Quasiparalelismus

Quasiparalelismus je další možný způsob paralelismu na jednoprocesorových architekturách. Je charakterizován těmito vlastnostmi:

- v daném okamžiku je prováděn pouze jeden proces
- přepnutí procesu zajišťuje aktivní proces provedením speciální instrukce

Schopnost simulace paralelního zpracování úloh je v tomto případě založena na využití korutin.

3.2.1 Korutina

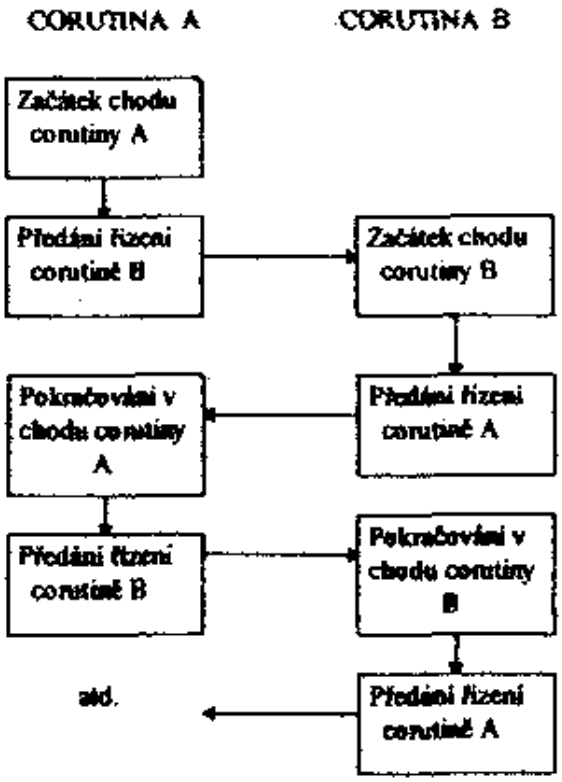
Představují stavební prvek při tvorbě quasiparalelních programů. Korutiny umožňují předávání řízení mezi sebou buď přímo prostřednictvím speciální instrukce, nebo pomocí rozvrhovacího mechanismu realizovaného schedulerem.

Korutiny jsou v některých rysech podobné podprogramům. Stejně jako ony obsahují lokální data a kód (příkazy). Podprogramy jsou aktivovány voláním z jiných podprogramů nebo z hlavního programu. Po vykonání své funkce se navracejí na místo, odkud byly zavolány, přičemž se ztrácejí hodnoty všech lokálních proměnných. Při dalším volání podprogramu začíná tento svoji činnost prvním příkazem ve svém těle.

Korutiny jsou formálně zapisovány jako podprogramy bez parametrů, tedy stejně jako procesy. Aktivování korutiny z jiné oblasti programu se provádí mechanizmem přenosu řízení nikoliv voláním podprogramu. Proto při dočasném přerušení běhu korutiny jsou uchovány hodnoty lokálních proměnných dosažené během její činnosti. Je-li v dalším chodu programu řízení opět předáno korutině, pokračuje tato v dalším zpracování v místě, kde naposled předala řízení jiné části programu, přičemž hodnoty lokálních údajů, kterých nabyly v okamžiku předchozího přerušení činnosti, zůstávají zachovány.

Je si třeba uvědomit, že k přerušení běhu korutiny nedochází vynucené na základě popudu scheduleru, jenž je pravidelně inicializován hodinovými impulsy, ale explicitně voláním speciální instrukce. Jde tedy o záměrné přerušení činnosti korutiny dané potřebou řešeného algoritmu. Tato skutečnost zmenšuje problémy se synchronizací, avšak má-li korutina kritickou oblast společnou s nějakým procesem řízeným schedulerem, je třeba dodržovat pravidla bezpečné synchronizace tak, jako by šlo o proces řízený schedulerem.

Mechanismus předávání řízení mezi korutinami je patrný z Obr.2:



Obr.2 Quasiparalelní zpracování dvou korutin

4. REALIZACE SOUBĚŽNĚ PRACUJÍCÍCH ÚLOH PROGRAMOVACÍM JAZYKEM MODULA -2

Programovací jazyk Modula-2 byl v letech 1978-80 navržen prof. Wirthem, který je také duchovním otcem programovacího jazyka Pascal a OBERON. Programovací jazyk Pascal umožňoval strukturované programování. Definoval základní typy, pole, záznamy a množiny. Velkým přínosem bylo zavedení ukazatelů a dynamických proměnných. To umožnilo vytvářet software pro širokou oblast aplikací vyžadujících dynamických datových struktur. Na sklonku 80. Let vznikají nové požadavky na tvorbu RT-aplikací vedoucích prof.Wirtha k návrhu nového programovacího jazyka. Tímto se stala Modula-2. Základní inovací byla modulární koncepce s možností samostatné kompilace jednotlivých modulů. Explicitní definice rozhraní a zajištění ochrany typů jistým způsobem usnadnila tvorbu rozsáhlých aplikací a tím umožnila snadnější týmovou práci nad vývojem softwaru. Důležitým přínosem tohoto programovacího jazyka byla možnost programování na strojové úrovni tzv. low level programming a pro nás velmi důležitá implementace prostředků pro tvorbu souběžně pracujících úloh.

4.1 Vývojové prostředí TopSpeed Modula - 2

Velmi snadné programování v jazyce Modula-2 nám umožňuje vývojové prostředí TopSpeed Modula -2. To bylo vytvořeno s ohledem na požadavek jednoduchého, rychlého a vyhovujícího návrhu softwarových aplikací na architekturách osobních počítačů. Prostředí TopSpeed Moduly obsahuje mimo jiné celou řadu speciálních

procedur a funkcí vytvořených za účelem snadnějšího a produktivnějšího návrhu vašich aplikací. Všechny tyto procedury jsou roztříděny do skupin modulů řešících určitou oblast problému. Pro nás jsou podstatné dva z této množiny modulů. Jsou to knihovní moduly:

- Process
- SYSTEM

4.2 Knihovní modul Process

Tento knihovní modul obsahuje všechny důležité procedury a funkce nutné k vytváření pseudoparalelních úloh. Stavebním prvkem tohoto druhu paralelismu je proces. Ten je realizován programovacím jazykem Modula-2 jako procedura bez parametrů. Struktura procesu pomocí základních příkazů jazyka je následovná:

```
PROCEDURE Proces;
VAR     deklarace lokálních proměnných ;
BEGIN
        inicializační část proměnných ;
        LOOP

                tělo procesu

        END;
END Proces;
```

Tímto jsme definovali část programu, která by bez vyvolání speciální procedury byla celá zpracovávána sekvenčně bez přerušení. Již jsme dříve uvedli, že touto procedurou je *scheduler*, který udržuje seznam všech aktivních procesů, přičemž jeho úkolem je přidělovat čas procesoru jednotlivým procesům. Knihovní modul *Process* obsahuje proceduru zahajující činnost tohoto scheduleru:

```
PROCEDURE StartScheduler;
```

Opačnou procedurou k této je pak

```
PROCEDURE StopScheduler;
```

ukončující vlastní činnost scheduleru. Vyvoláním této procedury končí pseudoparalelní zpracování jednotlivých procesů.

Pojem proces nebo jiné ekvivalentní vyjádření nepředstavuje v jazyce Modula-2 žádné rezervované slovo. Vlastní proces je vytvořen voláním procedury *StartProcess* modulu Process. Tato procedura má tři parametry:

```
PROCEDURE StartProcess ( P:PROC; N:CARDINAL; Pr:CARDINAL);
```

,kde P...je název procedury bez parametrů představující kód procesu

N...je velikost pracovního prostoru zásobníkové paměti procesu sloužící k uložení kontextu, který zahrnuje obsahy všech registrů procesoru, stavy proměnných procesu, status a prioritu příslušného procesu

Pr...určuje prioritu procesu. Každý proces má tak definovanou svou prioritu, přičemž přednost ve zpracování mají procesy s vyšší prioritou

Opačnou procedurou k této je pak *StopProcess*.

Strukturu pseudoparalelního programu představuje následující kód programu:

```
MODULE Procesy;
IMPORT Process;
(*** definice těl jednotlivých procesů ***)

PROCEDURE Proces1;
BEGIN
        LOOP
          . . .
        END;
END Proces1;

PROCEDURE Proces2;
BEGIN
        LOOP
          . . .
        END;
END Proces2;

(*** hlavní program ***)

BEGIN
        Process.StartScheduler;
        Process.StartProcess ( Proces1, 1024, 1);
        Process.StartProcess ( Proces2, 1024, 1);
        LOOP
        END;
END Procesy.
```

Využívají-li procesy společný zdroj, je nutné do těla takovýchto procesů umístit příkazy realizující synchronizační nástroje. Knihovní modul Process obsahuje procedury představující tyto základní synchronizační nástroje:

- *čekání* -

```
PROCEDURE Delay ( t: CARDINAL) ;
        , která pozastaví proces na   t*1/18 časový okamžik
```

- *uzamčení* -

```
PROCEDURE Lock ;
        , uzamykající proces

PROCEDURE Unlock;
        , odemykající proces
```

♦ *signál* -

```
PROCEDURE Init ( VAR s: SIGNAL );
        , inicializující signál s

PROCEDURE SEND ( s: SIGNAL ) ;
        , zasílající signál

PROCEDURE WAIT ( s: SIGNAL );
        , čekající na daný signál
```

Následující programový modul ukazuje definici procesů se zajištěním synchronizace. Program obsahuje celkem tři procesy. Dva procesy vypisující na obrazovce (představující společný zdroj) textovou zprávu. Třetí proces čeká na signál, který vznikne na základě stisku libovolné klávesy a způsobí ukončení celého programu.

```
MODULE Demo;
IMPORT IO,Process;
VAR Konec: Process.SIGNAL;

PROCEDURE Proces1;
BEGIN
        LOOP
                Process.Lock,
                IO.WrStr ( ' Proces1 ');
                Process.Unlock;
                Process.Delay(1);
        END;
END Proces1;

PROCEDURE Proces2;
BEGIN
        LOOP
                Process.Lock;
                IO.WrStr ( '  Proces2 ');
                Process.Unlock;
                Process.Delay(1);
        END;
END Proces2;

PROCEDURE TestKonce;
VAR press: BOOLEAN;
        ch: CHAR;
BEGIN
        LOOP
                Process.Lock;
                press:=IO.KeyPressed();
                Process.Unlock;
                IF press THEN
                        IF Process.Awaited (Konec) THEN
                                Process.SEND (Konec);
```

```
                END;
            Process.Delay(1);
        END;
END TestKonce;

BEGIN
        Process.Init(Konec);
        Process.StartScheduler;
        Process.StartProcess ( Proces1, 1000, 1);
        Process.StartProcess ( Proces2, 1000, 1);
        Process.StartProcess ( TestKonce, 1000, 1);
        Process.WAIT (Konec);
        Process.StopScheduler;
END Demo.
```

4.3 Knihovní modul System

Knihovní modul SYSTEM obsahuje všechny důležité funkce a procedury nutné k vytváření quasiparalelně pracujících úloh. Inicializace vlastního procesu je provedena voláním procedury NEWPROCESS tohoto knihovního modulu. Tato procedura má čtyři parametry:

```
PROCEDURE  NEWPROCESS (  P: PROC;
A: ADDRESS;
n: CARDINAL;
VAR p1: ADDRESS );
```

, kde P... je název procedury bez parametrů představující vlastní kód procesu
A a n ... definují pracovní oblast procesu
A ... je adresa zásobníku procesu
n ... je velikost tohoto zásobníku (pracovní oblast musí být alokována před voláním procedury NEWPROCESS)
p1... představuje referenci na vytvořený proces

Procedura NEWPROCESS proces sice vytvoří, ale neaktivizuje ho. To je provedeno vyvoláním procedury TRANSFER knihovního modulu SYSTEM.

```
PROCEDURE  TRANSFER ( VAR p1,p2 : PROC );
```

, kde p1 a p2 jsou referencemi na příslušné procesy. Tato procedura způsobí odložení procesu p1 a aktivizuje proces p2. Po provedení této procedury bude p1 identifikován jako odložený proces a jeho opětovné zahájení činnosti je umožněno dalším provedením příkazu TRANSFER tentokráte uvnitř těla procesu p2.
Struktura programu realizující quasiparalelní zpracování úloh může být například:

```
MODULA  Procesy;
IMPORT SYSTEM,Storage;

PROCEDURE Proces1;
```

```
        BEGIN
            LOOP
                ...
                SYSTEM.TRANSFER (p1,p2);
                ...
            END;
        END Proces1;

PROCEDURE Proces2;
        BEGIN
            LOOP
                ...
                SYSTEM.TRANSFER (p2,p1);
                ...
            END;
        END Proces2;

        BEGIN
            Storage.ALLOCATE ( WorkSpace1,1024);
            Storage.ALLOCATE ( WorkSpace2,1024);
            SYSTEM.NEWPROCESS ( Proces1, WorkSpace1, 1024,p1);
            SYSTEM.NEWPROCESS ( Proces2, WorkSpace2, 1024,p2);
            SYSTEM.TRANSFER (main,p1);
        END Procesy.
```

5. ZÁVĚR

Oblast využití programovacího jazyka Modula-2 je velmi široká. Je použitelný k návrhu operačních systémů, grafických aplikací, vojenských systémů apod. Díky nástrojů potřebných k realizaci souběžně pracujících úloh je tohoto jazyka využíváno k návrhu vizualizací a řízení průmyslových procesů.

LITERATURA

[1] Schiper,A., Concurrent programming, North Oxford Academic Publishers, 1989

[2] Burns,A.,Wellings,A.: Real-Time Systems and Their Programming Languages, Addison -Wesley, 1994

[3] King, K.N., TopSpeed Modula-2 language tutorial, Clarion Software Corporation, 1992