o.s. ReTOS

OPERAČNÍ SYSTÉM REÁLNÉHO ČASU

Příručka uživatele a programátora

***SofCon*®spol. s r.o.**
Střešovická 49
162 00 Praha 6
tel/fax: +420 220 180 454
E-mail: sofcon@sofcon.cz
www: http://www.sofcon.cz

Verze dokumentu 1.11

Informace v tomto dokumentu byly pečlivě zkontrolovány a SofCon věří, že jsou spolehlivé, přesto SofCon nenese odpovědnost za případné nepřesnosti nebo nesprávnosti zde uvedených informací.
SofCon negarantuje bezchybnost tohoto dokumentu ani programového vybavení, které je v tomto dokumentu popsané. Uživatel přebírá informace z tohoto dokumentu a odpovídající programové vybavení ve stavu, jak byly vytvořeny a sám je povinen provést validaci bezchybnosti produktu, který s použitím zde popsaného programového vybavení vytvořil.
SofCon si vyhrazuje právo změny obsahu tohoto dokumentu bez předchozího oznámení a nenese žádnou odpovědnost za důsledky, které z toho mohou vyplynout pro uživatele.

Datum vydání: 06.08.2004
Datum posledního uložení dokumentu: 06.08.2004
(Datum vydání a posledního uložení dokumentu musí být stejné)

Upozornění:
V dokumentu použité názvy výrobků, firem apod. mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Obsah :

1. O dokumentu 9
1.1. Revize dokumentu 9
1.2. Účel dokumentu 9
1.3. Rozsah platnosti 9
1.4. Související dokumenty 9
2. Termíny a definice 9
3. Úvod 10
3.1. Interaktivní program 10
3.2. Řídící program 10
3.3. Plánování času 10
3.4. Paralelismus 10
3.5. Předávání zpráv 10
3.6. Realtime program 11
3.7. Přepínání procesů 11
3.8. Priority 11
3.9. Paralelní výpočty 11
3.10. Využití jádra ReTOS 12
4. Základní vlastnosti 12
4.1. Implementace ReTOSu 12
4.2. Procesy 12
4.3. Správa času 12
4.4. Schránky 12
4.5. Přerušení 12
4.6. Chráněná sekce 13
4.7. Výjimky 13
5. Procesy 13
5.1. Data procesu 13
5.2. Deklarace procesu 13
5.3. Startovací sekvence 13
5.4. Instance procesu 14
5.5. Parametry procesu 14
5.6. Zásobník procesu 14
5.7. Ukončení procesu 15
5.8. Zrušení procesu 15
5.9. Změna priorit 15
5.10. Přepínání kontextu 15
6. Fronty procesů 16
6.1. Fronta Ready 16
6.2. Fronta Delay 16
6.3. Inicializace front 16
7. Hlavní proces 17
7.1. Inicializace jádra 17
7.2. Ukončení práce jádra 17
7.3. Uživatelské ukončení práce jádra 17
8. Priorita procesů 17
8.1. Statická priorita 18
8.2. Dynamická priorita 18
8.3. Synchronizace pomocí priorit 18

8.4. Změna priorit 18
9. Komunikace procesů 18
9.1. Schránky 19
9.2. Zasílání a příjem zpráv 19
9.3. Zasílání prázdných zpráv 19
9.4. Funkce Arrived a Accept 19
9.5. Vícenásobný příjem zpráv 20
9.6. Přijímání prázdných zpráv 20
9.7. Funkce Choice a Receive 22
9.8. Synchronní a asynchronní předávání zpráv 23
9.9. Timeout 23
9.10. Proměnná délka zprávy 23
10. Schránky 24
10.1. Fronta procesů u schránky 24
10.2. Inicializace schránek 24
10.3. Vlastnost broadcasting 24
10.4. Druhy schránek 24
10.5. Parametry schránek 24
10.6. Typy a druhy schránek 25
10.7. Testování a nedestruktivní čtení schránek 25
10.8. Vyprázdnění schránky 26
11. Správa času 26
11.1. Nastavení časové jednotky 26
11.2. Zpracování časového přerušení 27
11.3. Plánování procesů 27
11.4. Přerušovací zásobník 27
11.5. Zpracování přetečení času 28
12. Vypršení prodlevy (timeout) 29
12.1. Zadání timeoutu 29
12.2. Zpracování timeoutu 29
13. Chráněná sekce 30
13.1. Zamykání jádra 30
14. Přerušení 31
14.1. Přerušovací schránka 31
14.2. Zpracování přerušení 31
14.3. Začátek zpracování přerušení 31
14.4. Omezení při zpracování přerušení 32
14.5. Ukončení zpracování přerušení 32
14.6. Rychlé ukončení přerušení 32
14.7. Inicializace zpracování přerušení 32
14.8. Stanovení velikosti přerušovací schránky 34
14.9. Inicializace přerušovacího zásobníku 34
15. Používání aritmetického koprocesoru a softwarového emulátoru 34
15.1. Ošetření aritmetického koprocesoru 34
15.2. Inicializace emulátoru koprocesoru 35
16. Ladění a hlášení chyb 35
16.1. Trasování programu 35
16.2. Chyby jádra 36
17. Výjimky 36

17.1. Handlery 37
17.2. Parametry handlerů 37
17.3. Proměnná-výjimka 37
17.4. Instalace handleru 37
17.5. Prázdný handler 37
17.6. Rušení handlerů 37
17.7. Sdílení handlerů 37
18. Inicializace a ukončení činnosti jádra 38
18.1. Základní inicializace 38
18.2. Inicializace schránek 39
18.3. Inicializace přerušovacího zásobníku 39
18.4. Inicializace časového přerušení 39
18.5. Nastavení časové jednotky 39
18.6. Ošetření přetečení času 39
18.7. Inicializace ostatních přerušení 39
18.8. Ukončení činnosti jádra 39
18.9. Uživatelské ukončení 40
18.10. Znovuspuštění jádra 40
18.11. Inicializace trasování 40
19. Omezení 40
20. Přehled procedur a funkcí 40
20.1. Procesy 40
20.2. Schránky 41
20.3. Ostatní 42
21. Popis konstant a typů 42
22. Popis proměnných 43
23. Popis procedur a funkcí 44
23.1. Abort procedure 44
23.2. Accept procedure 44
23.3. AcceptTimeOut function 45
23.4. AcceptVar procedure 45
23.5. AcceptVarTimeOut function 46
23.6. Arrived function 46
23.7. ArrivedEmpty function 47
23.8. BreakPoint procedure 47
23.9. Choice procedure 47
23.10. ChoiceEmpty procedure 48
23.11. DelayCount function 48
23.12. DeleteHandlers procedure 48
23.13. Di procedure 48
23.14. Ei procedure 49
23.15. Empty function 49
23.16. FastInterrupExit procedure 49
23.17. FirstMessage function 49
23.18. Full function 50
23.19. GetPriority procedure 50
23.20. Identification function 50
23.21. Init8087 procedure 50
23.22. InitCounterMailBox procedure 51

23.23. InitDynMailBox procedure 51
23.24. InitEmulátor procedure 51
23.25. InitInterruptStack procedure 52
23.26. InitSingleMailBox procedure 52
23.27. InitSingleOverMailBox procedure 52
23.28. InitStatMailBox procedure 53
23.29. InitSyncMailBox procedure 53
23.30. InitTraceFile procedure 53
23.31. InitVarMailBox procedure 53
23.32. InstallHandler function 54
23.33. InterrupEntry procedure 54
23.34. InterrupExit procedure 54
23.35. Lock procedure 55
23.36. LockKernel procedure 55
23.37. Match function 55
23.38. MessageCount function 55
23.39. MessageLength function 56
23.40. NextMessage function 56
23.41. Popf procedure 56
23.42. ProcCount function 56
23.43. ProcessCount function 57
23.44. Purge procedure 57
23.45. Pushf procedure 57
23.46. ReadyCount function 57
23.47. Receive function 57
23.48. ReceiveTimeOut function 58
23.49. ReceiveVar function 59
23.50. ReceiveVarTimeOut function 59
23.51. Restore8087 procedure 59
23.52. RunningProcess function 60
23.53. Save8087 procedure 60
23.54. Send procedure 60
23.55. SendEmpty procedure 61
23.56. SendTimeOut function 62
23.57. SendEmptyTimeOut function 62
23.58. SendInterrupt procedure 63
23.59. SendInterruptEmpty procedure 63
23.60. SetEntryProc procedure 64
23.61. SetExitProc procedure 64
23.62. SetInterruptHandler procedure 64
23.63. SetPriority procedure 65
23.64. SetPriorityIdent procedure 66
23.65. SimulationTime function 66
23.66. Start procedure 66
23.67. StartMain procedure 67
23.68. StartTimeSlicing procedure 68
23.69. TestLock function 68
23.70. TestLockKernel function 68
23.71. TestKernel function 68

23.72. Terminate procedure 68
23.73. TerminateMain procedure 69
23.74. UnLock procedure 69
23.75. UnLockKernel procedure 69
23.76. Wait procedure 70
23.77. WaitUntil procedure 70
23.78. _Full funkce 71

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
1.00	1.XX			První vydání
1.10	3.XX	Tu	16.05.2003	Úprava dokumentu dle ISO9000
1.11	3.XX	Wil	06.08.2004	Drobné jazykové úpravy dokumentu.

1.2. Účel dokumentu

Tento dokument slouží jako popis operačního systému reálného času ReTOS, který je implementován v knihovně Kernel.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

1.4. Související dokumenty

Pro čtení tohoto dokumentu není potřeba číst žádný další manuál.

Popis formátu verze knihovny a souvisejících funkcí je popsán v manuálu „LibVer“.

2. Termíny a definice

Používané termíny a definice jsou popsány v samostatném dokumentu „Termíny a definice“.

3. Úvod

3.1. Interaktivní program

Tradiční schéma programu, který zpracovává připravená data a produkuje data výstupní, není pro mnoho softwarových úloh adekvátní. Každý interaktivní program, který komunikuje s uživatelem, se od tohoto schématu odchyluje, protože vstupní data (příkazy uživatele) vznikají až při běhu programu. Schéma takového programu je lépe vyjádřeno představou, že program v určitých místech očekává od uživatele zprávu, po jejím přijetí provádí požadovanou činnost, uživatele informuje pomocí výstupních zpráv a nakonec opět očekává zprávu od uživatele.

3.2. Řídící program

Program však nemusí komunikovat pouze s uživatelem. Stejným schématem (tedy výměnou zpráv) lze popsat i spolupráci počítače s technickým zařízením. Výstupní zprávy programu jsou potom obvykle data předávána programem na nějaký výstupní port, kterým se provádí ovládání tohoto zařízení. Vstupní zprávy jsou data, která program čte ze vstupních portů (například vstup z AD převodníku) a dále přerušení, které zařízení generuje.

3.3. Plánování času

Výstupní zprávy můžeme chápat jako povely pro vnější zařízení. Je zřejmé, že vydávání těchto povelů (např. povel pro jeden krok krokového motoru) musí být správně naplánováno v čase. Při psaní takového řídícího programu musíme mít tedy prostředky, které umožňují přesně určit, kdy bude který povel vydán.

3.4. Paralelismus

Obvykle však program neřídí pouze jedno technické zařízení, ale celou jejich sadu. Tato zařízení mohou nebo nemusí být shodná, avšak v každém případě nemusí pracovat synchronně a mohou se v jednom okamžiku nacházet v různých stavech činnosti. V takovém případě je nezbytné, aby na jednom počítači běželo současně několik řídících programů a každý program řídil jedno zařízení. Pokud se jedná o více zařízení stejného typu, pak příslušné programy mohou být shodné, avšak musí pracovat paralelně a asynchronně, stejně jako jednotlivá zařízení, která obsluhují.

3.5. Předávání zpráv

Stejný požadavek (tj. běh více programů najednou) vzniká i tehdy, když počítač řídí pouze jedno zařízení, avšak současně komunikuje s operátorem tohoto zařízení. Nelze totiž obvykle přerušit obsluhu zařízení (tedy vydávání povelů) po dobu, po kterou operátor přemýšlí jaký má vydat příkaz. I v tomto případě je nezbytné, aby na počítači běžely dva programy: jeden pro obsluhu zařízení a druhý pro komunikaci s operátorem. Je ovšem zřejmé, že tyto dva programy musí spolupracovat.

Poté co program, který komunikuje s operátorem, přečte a dekóduje jeho příkaz, musí sdělit řídícímu programu, že operátor požaduje změnu ve způsobu řízení. Jinými slovy si tyto dva programy potřebují předat zprávu.

3.6. Realtime program

Programy, které spolu komunikují a předávají si zprávy, se nazývají procesy a souhrn takových komunikujících procesů tvoří tzv. paralelní program. Pokud jsou jednotlivé procesy paralelního programu řízeny také skutečným časem, nazýváme takový program realtime. Realtime program je tedy paralelní program, který se skládá z jednotlivých procesů a ve kterém je řešen i problém časového řízení úlohy.

3.7. Přepínání procesů

Ideální by zřejmě bylo, kdyby každý proces běžel na jednom procesoru. Takové řešení je však komplikované a tedy drahé. Přitom je zřejmé, že když proces očekává zprávu a tedy stojí, je procesor nevyužitý a mohl by v tomto okamžiku vykonávat jiný proces, který je připraven k provádění. Mechanismus, při kterém procesor střídavě vykonává podle potřeby různé procesy, nazýváme sdílení času procesoru. Toto sdílení je zajišťováno tzv. přepínáním procesů. K přepínání procesů může docházet tehdy, když proces nemůže pokračovat v činnosti (čeká na zprávu), nebo navíc ještě pravidelně po uplynutí nějakého časového úseku.

3.8. Priority

Při přepínání procesů je třeba rozhodovat, kterému novému procesu se procesor přidělí. Aby toto přidělování nebylo náhodné, zavádí se pojem priority procesů a procesor se přiděluje procesu s nejvyšší prioritou. Priorita může mít dvojí charakter. Buď nezávisí na čase, a pak ji nazýváme statickou, nebo automaticky klesá spolu s dobou, po kterou byl procesu přidělen procesor, a pak ji nazýváme dynamickou. Dynamická priorita umožňuje spravedlivé rozdělování času procesoru mezi více procesů.

3.9. Paralelní výpočty

Paralelní programy však nemusí sloužit pouze pro obsluhu technických zařízení. Stejně nezbytné se jeví paralelní programování i pro zefektivňování klasických programů. Přitom prakticky jedinou metodou pro zrychlování výpočtů je jejich provádění na multiprocesorových počítačích, kde na každém procesoru se provádí pouze část výpočtu. Aby však bylo možné využívat najednou více procesorů, je třeba pro danou úlohu vytvořit paralelní algoritmus, který definuje paralelní procesy a potřebné předávání zpráv mezi těmito procesy. Paralelní algoritmy je však třeba z praktických důvodů ověřovat na dostupných jednoprocesorových počítačích.

3.10. Využití jádra ReTOS

Jádro ReTOS je prostředek, který umožňuje vytvářet paralelní i realtimeové programy v nejstandardnějším programovacím jazyku Turbo Pascal a spouštět je na nejdostupnějších a nejlevnějších počítačích PC. Z jazyka Turbo Pascal se použitím jádra ReTOS stává prostředek kvalitativně vyšší úrovně, který poskytuje možnosti specializovaných paralelních a realtimeových jazyků. Využití jádra ReTOS je jak při výuce paralelního a realtimeového programování, tak při tvorbě profesionálních realtimeových systémů. Všem těmto požadavkům vyhovují jednoduché a přitom mocné a efektivní konstrukce, které jsou v jádře použité.

4. Základní vlastnosti

4.1. Implementace ReTOSu

ReTOS je prostředek pro psaní paralelních programů v jazyku Turbo Pascal. Je tvořen jednotkou Kernel napsanou v jazyku Turbo Pascal 5.5 (resp. Turbo Pascal 6.0), která se použije v uživatelském programu. Tato jednotka umožňuje uživateli přístup k operacím typickým pro realtimeové programovací jazyky.

4.2. Procesy

Paralelní program pod jádrem ReTOS je tvořen procesy, které během výpočtu běží pseudoparalelně. Každý proces je naprogramován jako procedura (proces může mít parametry). Procesy se spouští voláním této procedury. Je možno spustit libovolně mnoho instancí procesů z jedné definice procedury. Procesům je přidělena statická a dynamická priorita.

4.3. Správa času

Časová správa procesů se provádí pomocí absolutního času, který se měří od doby spuštění systému. Proces je možno pozdržet o zadanou dobu nebo pozdržet do zadané doby.

4.4. Schránky

Procesy spolu komunikují prostřednictvím schránek, na které se posílají zprávy, a ze kterých se zprávy vybírají. Různou inicializací je dosažena velká variabilita chování schránek.

4.5. Přerušení

Pro ošetření přerušení je vytvořen vzor, podle kterého je třeba psát ošetřující procedury. Ošetřující procedura může zaslat hodnotu na schránku, a tak předat zpracování přerušení procesu, který u schránky čeká na zprávu.

4.6. Chráněná sekce

Pro správu prostředků, které nemohou být sdíleny, jsou k dispozici procedury Lock a UnLock, jejichž použití vytváří chráněnou sekci.

4.7. Výjimky

Jádro umožňuje definici výjimek a instalaci handlerů pro jejich ošetření.

5. Procesy

Základní jednotkou paralelního programu je proces. Procesy při běhu paralelního programu vznikají, provádějí svou činnost a zanikají. V realtimeovém programu je proces obvykle tvořen věčným cyklem, který provádí stále stejnou činnost až do ukončení programu. Vlastní program se chápe také jako proces, avšak platí pro něj některé výjimky, takže je popsán v odstavci Hlavní proces. Každý proces se skládá ze tří částí: z tabulky popisu procesu, ze zásobníku a z vlastního kódu procesu.

5.1. Data procesu

Tabulka popisu procesu a jeho zásobník jsou automaticky alokovány na heapu a inicializovány při startu procesu. Tabulka popisu procesu obsahuje všechny informace, které jádro potřebuje pro práci s procesem. Je to zejména jméno procesu, odkaz na jeho zásobník, statická a dynamická priorita procesu atd. Význam některých položek je popsán dále.

5.2. Deklarace procesu

Kód procesu je tvořen procedurou Turbo Pascalu. Tato procedura musí být na nejvyšší úrovni programu, musí se volat vzdáleně (direktiva $F+) a na jejím počátku musí být zapsána tzv. startovací sekvence. Proces vznikne zavoláním této procedury a automaticky se zruší jejím dokončením. Tato procedura může obsahovat vnořené procedury i volat jiné procedury na nejvyšší úrovni. Je také možné, aby tato procedura (proces) volala jiné procedury (procesy) a tak startovala další procesy. Je dokonce možné, aby se tato procedura volala rekurzivně, a tak jedna instance procesu spustila jinou instanci téhož procesu.

5.3. Startovací sekvence

Startovací sekvence se skládá z volání procedury Start a z příkazu Exit. Pomocí parametrů procedury Start se zadává řetěz, který tvoří jméno procesu (předdefinovaného typu IdentType), délka zásobníku, který se pro proces vyhradí, a dále statická a dynamická priorita procesu. Jméno procesu slouží pro jeho identifikaci uvnitř jádra. Uživatel může vytvořit i více procesů se stejným jménem. Vlastnosti statické a dynamické priority jsou popsány v odstavci Priorita procesů.

5.4. Instance procesu

Každé zavolání procedury se startovací sekvencí způsobí vznik nového procesu s vlastním zásobníkem a s vlastní tabulkou popisu procesu. Vzhledem k tomu, že všechny procesy vzniklé voláním jedné procedury, sdílejí stejný kód, nazýváme je také někdy instance procesu.

5.5. Parametry procesu

Procedura, která určuje činnost procesu, může mít parametry a lokální proměnné. Pokud má parametry volané odkazem, je třeba dát pozor na to, aby skutečné parametry existovaly po celou dobu života procesu. Skutečné parametry zadané při volání této procedury a tedy při spuštění procesu se automaticky přesunou na nově vytvořený zásobník a jsou tedy při provádění procesu dostupné.

5.6. Zásobník procesu

Při přepínání procesů a tedy výměnách hodnot registrů SS a SP se automaticky mění i hodnota systémové proměnné StackLimit (viz. standardní jednotka System), takže procesy je možno provádět se zapnutou kontrolou vyčerpání zásobníku (direktiva $S+). Kontrola hlídá, aby volné místo na zásobníku nebylo menší než 512 bytů. Velikost zásobníku zadávaná jako parametr procedury Start musí být tedy přinejmenším 512 bytů plus délka aktivací všech procedur, které jsou při běhu daného procesu současně zavolány, plus cca 30 bytů pro zpracování jedné úrovně přerušení. Každý proces má svůj vlastní zásobník a proto hodnota SS registru jednoznačně určuje daný proces.

V následujícím příkladu je ukázána definice procesu a vytvoření dvou instancí tohoto procesu s odlišným jménem. Při spuštění je procesům předán textový parametr, který je využit jako jméno spuštěné instance procesu.

příklad

```
uses Kernel; { použití jednotky Kernel }
{$F+}
procedure Kamarad(Jmeno : IdentType);
{$F-}
begin
  { začátek startovací sekvence }
  Start(Jmeno, { jméno vzniklého procesu }
        2048,  { délka zásobníku }
        100,   { statická priroita }
        0);    { dynamická priorita }
  Exit; { ukončení startovací sekvence }
  ...   { tělo procesu }
end;

begin
  ...
  Kamarad('Honza'); { spuštění procesu Honza }
  Kamarad('Karel'); { spuštění procesu Karel }
  ...
end.
```

5.7. Ukončení procesu

Proces se automaticky zruší, když se dokončí procedura, která definuje jeho chování. Přitom se automaticky uvolní tabulka popisu procesu a zásobník, které byly při startu procesu alokovány na heapu. Proces může sám sebe explicitně ukončit, jestliže kdykoliv během svého provádění zavolá proceduru jádra Terminate. Tato procedura je bez parametrů.

5.8. Zrušení procesu

Proces je možno zrušit také z jiného procesu procedurou jádra Abort. Parametrem této procedury je jméno rušeného procesu. Ve jménu je dovoleno použít hvězdičkové konvence. Procedura Abort zruší všechny procesy, jejichž jméno odpovídá zadanému parametru. Procedurou Abort je možno zrušit procesy bez ohledu na to, v jakém se nacházejí stavu (mohou být připraveny ke spuštění, čekat na zprávu nebo čekat na zadaný čas).

5.9. Změna priorit

Priority procesu, které jsou nastaveny pomocí parametrů procedury Start, je možno měnit i za běhu procesu. K tomu slouží procedury jádra SetPriority resp. SetPriorityIdent. Procedura SetPriority modifikuje priority tomu procesu, který jí zavolal, zatímco procedura SetPriorityIdent modifikuje priority všem procesům se jménem odpovídajícím zadanému jménu (je možno použít hvězdičkovou konvenci).

5.10. Přepínání kontextu

Během činnosti paralelního programu je každému procesu přidělován a odebírán procesor. Přitom může existovat potřeba měnit spolu s přidělováním a odebíráním procesoru kontext, ve kterém proces pracuje. Součástí každého procesu mohou být dvě dlouze volané lokální procedury bez parametrů, které se po inicializaci budou automaticky volat při odebírání a přidělování procesoru. Tyto procedury se inicializují po startu procesu procedurami SetEntryProc a SetExitProc a jsou například využívány pro ošetření aritmetického koprocesoru (viz kapitola „15 Používání aritmetického koprocesoru a softwarového emulátoru“).

V následujícím příkladu jsou uvnitř procesu Process definovány dvě procedury ExitProc a EntryProc, které při přidělování a odebírání procesoru procesu Process vypisují určenou zprávu na obrazovku.

příklad

```
uses Kernel;
{$F+}
procedure Process(Ident: IdentType);

  { deklarace procedury, která se volá při odebrání procesoru }
  procedure ExitProc;
  begin
    Writeln('Procesu ',Ident,' byl odebrán procesor');
  end;
```

```
  { deklarace procedury, která se volá při přidělení procesoru }
  procedure EntryProc;
  begin
    Writeln('Procesu ',Ident,' byl přidělen procesor');
  end;
{$F-}

begin
  { starovací sekvence procesu Process }
  Start(Ident, 1024, 0, 0); Exit;
  { instalace procedur ExitProc a EntryProc }
  SetExitProc(@ExitProc);
  SetEntryProc(@EntryProc);
  ...
end;
```

6. Fronty procesů

6.1. Fronta Ready

Tabulka popisu procesu obsahuje linkovací položky, které umožňují ukládat procesy do front. Jádro si stále udržuje dvě fronty procesů: frontu Ready a frontu Delay. Ve frontě Ready jsou zařazeny procesy, které mohou být okamžitě spuštěny. Fronta Ready je setříděna podle statické priority (podrobně viz kapitola „8 Priorita procesů“) a první proces v této frontě, který má nejvyšší statickou prioritu, je tzv. běžící. To je proces, který je právě procesorem vykonáván.

6.2. Fronta Delay

Ve frontě Delay jsou uloženy všechny procesy, které z nějakého důvodu nemohou být prováděny. Tyto procesy buď čekají na nějakou zprávu nebo jsou pozdrženy dokud nenastane určený čas. Procesy provádějící operaci s timeoutem (operace Accept nebo Receive s timeoutem a operace synchronní Send s timeoutem) čekají ve frontě Delay současně na dokončení operace a na vypršení timeoutu.

Fronta Delay je setříděna podle absolutního času, na který příslušný proces čeká. Procesy, které čekají na dokončení operací Send, Accept nebo Receive bez timeoutu, a tedy nečekají na žádný určený čas, jsou uloženy na konci fronty Delay.

6.3. Inicializace front

Fronty Ready a Delay jsou inicializovány procedurou StartMain, kterou se inicializuje jednotka Kernel. Do fronty Ready je přitom uložen proces Dummy (zahaleč), který se stává běžícím tehdy, když jsou všechny ostatní procesy ve frontě Delay (podrobnosti viz kapitola „18 Inicializace a ukončení činnosti jádra“).

7. Hlavní proces

Hlavní proces je proces, který je tvořen tělem programu. Zásobník hlavního procesu je tvořen standardním zásobníkem Turbo Pascalského programu (velikost je možno nastavit direktivou $M).

7.1. Inicializace jádra

Před jakoukoliv operací jádra musí být jádro inicializováno procedurou StartMain. Tato procedura otvírá činnost jádra a má pro hlavní proces podobnou funkci jako procedura Start pro ostatní procesy. Parametry procedury StartMain jsou statická a dynamická priorita hlavního procesu. Procedura inicializuje jádro a zařadí hlavní proces do fronty Ready. Jméno hlavního procesu je implicitně 'Main'.

7.2. Ukončení práce jádra

Hlavní proces musí být explicitně ukončen procedurou jádra TerminateMain, která odpovídá proceduře Terminate pro ostatní procesy. Po provedení procedury TerminateMain je činnost jádra ukončena a program dále pokračuje jako normální neparalelní program. Dříve však než dojde k pokračování programu za voláním procedury TerminateMain, čeká jádro na dokončení resp. zrušení ostatních procesů, které byly během provádění jádra spuštěny. Implicitně se čeká na dokončení všech procesů, avšak je možné zadat, aby procedura TerminateMain na jistý počet procesů nečekala. Tento počet musí být uložen do proměnné jádra MinProcessNo (proměnná MinProcessNo má implicitně hodnotu 0). V případě, že hodnota proměnné MinProcessNo je nenulová, ukončí procedura TerminateMain ty procesy, na jejichž dokončení se nemá čekat.

7.3. Uživatelské ukončení práce jádra

Procedura TerminateMain zavolá také proceduru bez parametrů, jejíž adresa je v proměnné jádra UserTerminate. Do této proměnné může uživatel zapsat adresu vlastní vzdálené procedury ukončující činnost jádra.

Nakonec procedura TerminateMain obnoví původní hodnoty přerušovacích vektorů platných v okamžiku volání procedury StartMain.

Nehledě na uvedené zvláštnosti se hlavní proces chová jako ostatní procesy a může používat všechny operace jádra např. pro komunikaci s jinými procesy nebo pro časové plánování.

8. Priorita procesů

Přidělování procesoru procesům je řízeno statickou a dynamickou prioritou. Platí, že čím je hodnota priorit vyšší, tím více dostávají procesy přednost. Hodnoty statické a dynamické priority procesu jsou součástí tabulky popisu procesu.

8.1. Statická priorita

Uživatel zadává statickou prioritu jako jeden z parametrů procedury **Start** resp. **StartMain** a to v rozsahu 0..MaxStatPri ($3ff0). První ve frontě Ready je vždy proces s největší statickou prioritou. Proces Dummy (zahaleč) má implicitně prioritu -MaxInt a je tedy vždy na konci fronty Ready.

8.2. Dynamická priorita

Mezi procesy, které mají stejnou statickou prioritu, rozhoduje tzv. součet časových penalizací, který je určován jednak dynamickou prioritou a jednak časem, který byl procesu přidělen. Dynamická priorita je také parametrem procedury **Start** resp. **StartMain** a je typu Byte. Za každou časovou jednotku (viz kapitola „4.3 Správa času“) je proces penalizován hodnotou 255-dynamická priorita. Pokud nechceme dynamickou prioritu využívat, je nejvhodnější nastavit ji na hodnotu 255 a tím zrušit časovou penalizaci procesů. Za delší časové období je tedy podíl času procesoru přiděleného jednotlivým procesům se stejnou statickou prioritou rovný podílu hodnot 255-dynamická priorita.

8.3. Synchronizace pomocí priorit

Pomocí statických priorit je možno procesy synchronizovat. Potřebujeme-li spustit z jednoho procesu více jiných procesů najednou, potom spouštěcí proces musí mít přinejmenším stejnou statickou prioritu jako spouštěné procesy. V opačném případě se spustí pouze první proces a dokud nedokončí svou činnost, nepřidělí se procesor spouštějícímu procesu, takže ke spuštění dalších procesů nedojde.

8.4. Změna priorit

Nastavovat hodnoty priorit je možno i za běhu procesu procedurami **SetPriority** a **SetPriorityIdent**. Obě procedury mají za parametry statickou a dynamickou prioritu. Procedura SetPriority nastaví nové hodnoty priorit tomu procesu, který ji zavolal. Proces je přemístěn ve frontě Ready podle nově zadaných priorit. Procedura SetPriorityIdent nastaví nové shodné priority všem procesům se zadaným jménem (je možno použít hvězdičkovou konvenci). Jsou-li procesy ve frontě Ready, jsou přeřazeny podle nových priorit. Priorita může být změněna i hlavnímu procesu, avšak jádro nepřipustí změnu priority procesu Dummy (zahaleč).

9. Komunikace procesů

Paralelně běžící procesy si obvykle potřebují sdělovat nějaké informace. Toto sdělování se provádí zasíláním zpráv. Zprávy mohou být libovolného typu (např. i pole nebo záznam), ale mohou být i prázdné, pokud je potřeba procesy zasíláním zpráv pouze synchronizovat. Zpráva může být zaslána buď některým procesem nebo rutinou pro ošetření přerušení.

9.1. Schránky

Pro zasílání a příjem zpráv slouží tzv. schránky. Schránka je proměnná předdefinovaného typu MailBox resp. SyncMailBox resp. InterruptMailBox (viz dále), která obsahuje frontu zpráv a frontu procesů, které na zprávu čekají. S výjimkou schránek pro zpracování přerušení (InterruptMailBox), je vždy alespoň jedna z těchto front prázdná.

9.2. Zasílání a příjem zpráv

Pro zasílání zpráv slouží operace Send (SendEmpty, SendTimeOut, SendEmptyTimeOut) a pro příjem zpráv slouží dvojice operací Arrived a Accept resp. Choice a Receive (ArrivedEmpty, AcceptTimeOut, AcceptVar, AcceptVarTimeOut, ChoiceEmpty, ReceiveTimeOut, ReceiveVar, ReceiveVarTimeOut). Vysílající proces zašle zprávu operací Send na nějakou schránku a přijímající proces si potom operací Accept nebo Receive zprávu ze schránky vybere. Zprávy jsou vybírány ve stejném pořadí, v jakém byly do schránky zasílány. Fronta zpráv ve schránce je tedy typu FIFO. Přijímající proces může vykonat operaci Accept resp. Receive dříve, než je na schránku nějaká zpráva poslána. V takovém případě je přijímající proces u schránky pozdržen tak dlouho, dokud zpráva nepřijde nebo eventuelně dokud nevyprší zadaný timeout (viz kapitola „12 Vypršení prodlevy (timeout)“).

9.3. Zasílání prázdných zpráv

Pokud zpráva není prázdná, musí být nejprve uložena do vysílacího bufferu. Parametrem operace procedury Send je schránka, na kterou se má zpráva zaslat a dále adresa bufferu a délka předávané zprávy. Pro zasílání prázdných zpráv je definována procedura SendEmpty, jejímž parametrem je pouze cílová schránka.

9.4. Funkce Arrived a Accept

Nejprve popíšeme způsob přijímání zpráv pomocí operace Accept. Před tím, než proces zavolá vlastní proceduru Accept, určenou pro příjem zprávy, musí zavolat funkci Arrived (resp. ArrivedEmpty), která jako parametr zadá schránku, ze které chce zprávu číst, a dále vstupní buffer, do kterého má být zpráva přesunuta (pouze v případě, že není prázdná). Funkce Arrived (resp. ArrivedEmpty) je typu Boolean a musí být zavolána jako podmínka v příkazu if. Funkce vrací vždy hodnotu False a po jejím zavolání pokračuje proces else-částí příkazu if, ve které musí být zavolána procedura Accept. Procedura Accept prohlédne schránku, která byla zadána jako parametr ve volání funkce Arrived a pokud v ní není připravena zpráva, pozastaví proces ve frontě Delay. V okamžiku, kdy zpráva přijde, a nebo pokud je zpráva k dispozici již v okamžiku volání procedury Accept, je zpráva (neprázdná) přesunuta do bufferu, který byl zadán ve volání funkce Arrived. Současně je řízení předáno na místo příslušného volání funkce Arrived (resp. ArrivedEmpty) a je vrácena funkční hodnota True, takže program pokračuje then-částí příkazu if.

příklad

```
var MB: MailBox;
...
if ArrivedEmpty(MB) then
  Writeln('Přišla prázdná zpráva na schránku MB')
else Accept;
...
```

9.5. Vícenásobný příjem zpráv

Často vzniká situace, kdy proces potřebuje přijímat zprávy ze dvou nebo i více schránek současně. Tyto zprávy mohou být obecně různého typu. Proto je možné před zavoláním procedury Accept zavolat funkci Arrived vícekrát s různými parametry. Vícenásobné volání funkce Arrived se provede pomocí vnořených příkazů if:

příklad

```
var MB1, MB2, MB3: MailBox;
...

if Arrived(MB1, ...) then ...
else if Arrived(MB2, ...) then ...
else if Arrived(MB3, ...) then ...
...
else Accept;
...
```

Vzhledem k tomu, že funkce Arrived vrací vždy hodnotu False, dojde postupně k opakovanému volání funkce Arrived s různými parametry a potom k vyvolání procedury Accept. Procedura Accept prohlédne všechny schránky, které byly zadány jako parametry ve volání funkce Arrived a pokud v žádné z nich není připravena zpráva, pozastaví proces ve frontě Delay. V okamžiku, kdy přijde zpráva na některou zadanou schránku, a nebo je na některé schránce zpráva k dispozici již v okamžiku volání procedury Accept, je zpráva (neprázdná) přesunuta do bufferu, který byl zadán v příslušném volání funkce Arrived. Současně je řízení předáno na místo příslušného volání funkce Arrived (resp. ArrivedEmpty) a je vrácena funkční hodnota True, takže program pokračuje then-částí toho příkazu if, ve kterém se vyskytuje příslušné volání funkce Arrived.

9.6. Přijímání prázdných zpráv

Přijímající proces musí obsahovat buffer pro příjem neprázdné zprávy. Adresa a délka tohoto bufferu je parametrem procedury Arrived. Pro příjem prázdných zpráv slouží operace ArrivedEmpty, která tyto parametry nemá.

V následujícím příkladu zasílají procesy P a Q zprávy na dvě různé schránky a proces R tyto zprávy přijímá.

příklad

```
uses Kernel;
{ deklarace schránek pro zasílání zpráv }
var IntBox, FlagBox: MailBox;
```

```
{$F+ definice procesu zasílajícího zprávy na schránku IntBox }
procedure P;
{$F-}
var OutBuff : Integer; { výstupní buffer }
begin
  Start('P', 1024, 0, 255); Exit;
  { zaslání zprávy na schránku IntBox }
  ...
  OutBuff:=33;
  Send(IntBox, OutBuff, SizeOf(OutBuff));
  ...
end;

{$F+ definice procesu zasílajícího zprávy na schránku FlagBox }
procedure Q;
{$F-}
begin
  Start('Q', 1024, 0, 255); Exit;
  ...
  { zaslání prázdné zprávy na schránku FlagBox }
  SendEmpty(FlagBox);
  ...
end;

{$F+ proces přijímající zprávy ze schránek FlagBox a IntBox }
procedure R;
{$F-}
var InBuff : Integer;
begin
  Start('R', 1024, 0, 255); Exit;
  ...
  { příjem zprávy ze schránky FlagBox nebo IntBox }
  if ArrivedEmpty(FlagBox)) then
    Writeln('zpráva přišla na schránku FlagBox')
  else if Arrived(IntBox, InBuff, SizeOf(InBuff)) then
    Writeln('zpráva přišla na schránku IntBox a její hodnota je: ',
          InBuff)
  else Accept;
  ...
end;

begin
  { inicializace jádra }
  StartMain(10, 255);
  { inicializace schránek }
  InitCounterMailBox(FlagBox, 'FlagBox', Normal);
  InitDynMailBox(IntBox, 'IntBox', SizeOf(Integer), Normal);
  { spuštění procesů P, Q a R }
  P;
  Q;
  R;
  ...
  TerminateMain;
end.
```

9.7. Funkce Choice a Receive

V případě, že jsou ve více schránkách zprávy stejného typu, nás v některých případech nezajímá, z které konkrétní schránky zpráva přišla. V tom případě chceme uložit zprávu do jednoho bufferu a pokračovat jednou větví programu. V takovém případě použijeme pro příjem zprávy dvojici operací Choice (ChoiceEmpty) a Receive.

Kromě určení schránky a přijímacího bufferu mají procedury Choice a ChoiceEmpty jako parametr identifikační číslo, které slouží k určení schránky, ze které byla zpráva přijata. Funkce Receive pracuje obdobně, jako procedura Accept. Vrací se však vždy normálním způsobem a její výsledek odpovídá identifikačnímu číslu schránky, ze které byla zpráva přijata.

příklad

```
uses Kernel;
{ deklarace schránek pro zasílání zpráv }
var FlagBox, IntBox1, IntBox2 : MailBox;

{$F+ definice procesu zasílajícího zprávy na schránku IntBox }
procedure P(var IntBox : MailBox);
{$F-}
var OutBuff : Integer;
begin
  Start('P', 1024, 0, 255); Exit;
  { zaslání zprávy na schránku IntBox }
  ...
  OutBuff:=33;
  Send(IntBox, OutBuff, SizeOf(OutBuff));
  ...
end;

{$F+ definice procesu zasílajícího zprávy na schránku FlagBox }
procedure Q;
{$F-}
begin
  Start('Q', 1024, 0, 255); Exit;
  ...
  SendEmpty(FlagBox);
  ...
end;

{$F+ proces přijímající zprávy ze schránek FlagBox, Int1Box a Int2Box }
procedure R;
{$F-}
var InBuff : Integer;
begin
  Start('R', 1024, 0, 255); Exit;
  { příjem zprávy ze schránky FlagBox, Int1Box a
  Int2Box }
  ...
  ChoiceEmpty(1, FlagBox);
  Choice(2, IntBox1, InBuff, SizeOf(InBuff));
  Choice(2, IntBox2, InBuff, SizeOf(InBuff));
  case Receive of
    1: Writeln('zpráva přišla na schránku FlagBox');
    2: Writeln('zpráva přišla na schránku IntBox1 nebo '+ 'IntBox2 "+
```

```
            "a její hodnota je: ', InBuff);
  end;
  ...
end;

begin
  { inicializace jádra }
  StartMain(10, 255);
  { inicializace schránek }
  InitCounterMailBox(FlagBox, 'FlagBox', Normal);
  InitDynMailBox(IntBox1, 'IntBox1', SizeOf(Integer),Normal);
  InitDynMailBox(IntBox2, 'IntBox2', SizeOf(Integer),Normal);

  { spuštění procesů P, Q a R }
  P(IntBox1);
  P(IntBox2);
  Q;
  R;
  ...
  TerminateMain;
end.
```

9.8. Synchronní a asynchronní předávání zpráv

Podle chování procesů při zasílání zprávy rozlišujeme zasílání synchronní a asynchronní. Zpráva totiž může být okamžitě předána přijímacímu procesu pouze tehdy, když tento proces již vykonal operaci Accept nebo Receive a zprávu tedy očekává. Pokud tomu tak není, musí být zpráva buď ve schránce uschována a tento způsob se nazývá asynchronní předávání zpráv, nebo musí být vysílající proces pozdržen dokud nějaký přijímací proces o zprávu nepožádá a tento způsob se nazývá synchronní předávání zpráv. Synchronní a asynchronní předávání zpráv je určováno typem schránky (viz dále).

9.9. Timeout

Operace Accept a Receive na kteroukoliv schránku a operace Send na synchronní schránku může být vyvolána s timeoutem (viz kapitola „12 Vypršení prodlevy (timeout)“). V takovém případě jsou operace realizovány funkcemi SendTimeOut, SendEmptyTimeOut, AcceptTimeOut, AcceptVarTimeOut, ReceiveTimeOut, ReceiveVarTimeOut. Operace končí buď tím, že se normálně provedou nebo vyčerpáním zadaného timeoutu.

9.10. Proměnná délka zprávy

Je-li přijímána zpráva ze schránky s proměnnou délkou, je k tomu možno použít procedury AcceptVar, ReceiveVar nebo funkce AcceptVarTimeOut a ReceiveVarTimeOut, které vrátí skutečnou délku zprávy pomocí parametru volaného odkazem.

10. Schránky

Schránky jsou objekty, sloužící pro předávání zpráv mezi procesy a také pro předávání zpráv mezi procedurami ošetřujícími přerušení a procesy. Existují tři předdefinované typy schránek: MailBox pro asynchronní schránky, SyncMailBox pro synchronní schránky a InterruptMailBox pro schránky předávající zprávy z procedur ošetřujících přerušení.

10.1. Fronta procesů u schránky

U každé schránky může být pozdržen jeden nebo více procesů čekajících na zprávu. Každá schránka tedy obsahuje frontu procesů čekajících na zprávu. Nečeká-li žádný proces u dané schránky na zprávu, je tato fronta prázdná. Synchronní schránka (SyncMailBox) dále obsahuje frontu procesů, které chtějí zprávu předat, zatímco asynchronní schránka (MailBox) a přerušovací schránka (InterruptMailBox) obsahuje frontu zpráv, které čekají na to až budou převzaty nějakým procesem.

10.2. Inicializace schránek

Všechny schránky musí být explicitně inicializované. Při inicializaci je schránkám obdobně jako procesům přiřazeno jméno a jsou určeny jejich další vlastnosti.

10.3. Vlastnost broadcasting

Synchronní i asynchronní schránky mohou mít tzv. vlastnost broadcasting. Tato vlastnost způsobuje, že pokud na vysílanou zprávu čeká více procesů, je zpráva předána současně všem těmto procesům. V opačném případě je zpráva předána pouze procesu, který je první ve frontě a který tedy čeká na zprávu nejdelší dobu. Tuto vlastnost lze zadat jako poslední parametr inicializační procedury.

10.4. Druhy schránek

Podle použité fronty na zprávy se asynchronní a přerušovací schránky dále dělí na čítače, jednoduché schránky pouze na jednu zprávu, statické schránky na pevný počet zpráv, dynamické schránky na "libovolný" počet zpráv a dynamické schránky na zprávy proměnné délky. Chování fronty zpráv se určuje použitou inicializační procedurou.

10.5. Parametry schránek

Všechny inicializační procedury mají jako parametr identifikaci inicializované schránky, jméno schránky a dále nastavení příznaku broadcasting (Normal, Broadcast). Všechny asynchronní schránky s výjimkou čítače a dynamické schránky na zprávy proměnné délky musí být dále inicializovány délkou předávaných zpráv. Statická schránka musí být dále inicializována maximálním počtem zpráv, které v ní mohou být uschovány.

10.6. Typy a druhy schránek

Schránky tedy dělíme podle předdefinovaného typu použitého pro deklaraci schránky:

- MailBox - schránky pro asynchronní předávání zpráv

- SyncMailBox - schránky pro synchronní předávání zpráv

- InterruptMailBox - schránky pro předávání zpráv z procedur ošetřujících přerušení.

Schránky typu MailBox a InterruptMailBox dělíme dále podle druhu fronty pro úschovu zpráv ve schránce. Tato fronta se určuje použitou inicializační operací a hodnotami jejích parametrů a určuje, zda je schránka:

- dynamická - pro každou uschovanou zprávu je alokováno místo na heapu. Zpráv může být ve schránce tolik, kolik se jich do heapu vejde. Schránka se inicializuje procedurou InitDynMailBox.

- dynamická s proměnnou délkou - pro každou uschovanou zprávu je alokováno místo na heapu podle skutečné délky zprávy. Zpráv může být ve schránce tolik, kolik se jich do heapu vejde. Schránka se inicializuje procedurou InitVarMailBox.

- statická - fronta zpráv je tvořena kruhovou frontou na pevný počet zpráv. Schránka se inicializuje procedurou InitStatMailBox.

- jednoduchá - fronta může obsahovat pouze jednu zprávu. Schránka se inicializuje procedurou InitSingleMailBox.

- s přepisováním - fronta obsahuje pouze jedinou zprávu, avšak tato zpráva se přečtením z fronty nevyjme a je tedy možno číst ji opakovaně, dokud není přepsána jinou zprávou. Schránka se inicializuje procedurou InitOverMailBox.

- čítač - schránka na prázdné zprávy, u kterých je třeba evidovat pouze jejich počet. Schránka se inicializuje procedurou InitCounterMailBox.

Synchronní schránky se inicializují procedurou InitSyncMailBox.

10.7. Testování a nedestruktivní čtení schránek

Nad všemi schránkami jsou definovány operace, které zjišťují, resp. nastavují stav schránky. Funkce Empty resp. Full zjišťují, zda je schránka prázdná (neobsahuje žádnou zprávu), resp. plná (nová zpráva se již do ní nevejde). Funkce ProcCount udává počet procesů, které u schránky očekávají zprávu a funkce MessageCount udává počet zpráv, které jsou ve schránce uschovány resp. u synchronních schránek počet procesů, které chtějí zprávu předat. Funkce FirstMessage, NextMessage a MessageLength umožňují provádět nedestruktivní čtení schránek. Funkce FirstMessage vrací adresu první zprávy ve frontě. Funkce NextMessage vrací postupně adresu dalších zpráv. Poslední hodnotou dodanou funkcemi FirstMessage resp. NextMessage je hodnota nil. Funkce MessageLength vrací délku zprávy, jejíž adresu vrátila funkce FirstMessage nebo NextMessage.

Následující příklad popisuje vypsání všech zpráv z dynamické schránky s proměnnou délkou zpráv. Zprávy přitom zůstanou ve schránce uschovány. O zprávách se předpokládá, že to jsou řetězy s délkou 0 až 64KB.

příklad

```
uses Kernel;
var
  P: ^ array[0..$fff0] of char;          { pomocný ukazatel na zprávu }
  M: MailBox;                            { vypisovaná schránka };
  I: Word;

begin
  StartMain(0,0);
  { inicializace schránky M jako dynamické }
  InitVarMailBox(M, 'M', Normal);
  ...
  { zamčení, aby schránka během výpisu neměnila svůj stav }
  Lock;
  { obsazení ukazatele P adresou první zprávy nebo
    hodnotou nil, jestliže je schránka prázdná }
  P:=FirstMessage(M);
  while P<>nil do
  begin
    for I:=0 to MessageLength(M) do
    Write(P^[I]);
    Writeln;
    P:=NextMessage(M);
  end;
  UnLock;
  ...
```

10.8. Vyprázdnění schránky

Všechny zprávy ve schránce je možno vymazat operací Purge. Po provedení této operace je schránka prázdná. Jedná-li se o synchronní schránku, potom procesy, které čekaly až budou moci předat zprávu, pokračují v činnosti.

11. Správa času

Aby bylo možno provádět přesné časování procesů bez vlivu kumulovaných časových chyb, udržuje jádro hodnotu času v absolutních časových jednotkách od provedení operace StartTimeSlicing. Tato operace spouští správu času. Jejím parametrem je číslo přerušení, které se dále chápe jako přerušení od časovače. V systému MS-DOS je to číslo 8.

11.1. Nastavení časové jednotky

Přerušení přicházející od časovače jsou nejprve vydělena hodnotou proměnné jádra ClockDivider+1 a teprve pak jsou chápána jako uplynutí časové jednotky. Proměnná ClockDivider je typu Word a její implicitní hodnota je 0. Uživatel může tuto implicitní hodnotu změnit. Je-li použito standardní přerušení od systémového časovače (int 8) a je-li ponechána implicitní hodnota proměnné ClockDivider, je

velikost časové jednotky dána periodou přerušení systémového časovače (standardně 55 ms).

11.2. Zpracování časového přerušení

Každou časovou jednotku je inkrementována proměnná jádra RealTime typu TimeType, určující hodnotu absolutního času (typ TimeType je shodný s typem LongInt). Dále je běžící proces pokutován časovou penalizací (viz dynamická priorita). Z fronty Delay jsou vyjmuty všechny procesy, které čekaly na nastálou hodnotu času, a jsou přesunuty do fronty Ready. Do fronty Ready jsou procesy zařazovány podle svých priorit. Nakonec je procesor přidělen procesu s největší prioritou.

11.3. Plánování procesů

Fronta Delay je setříděna podle času, na který procesy čekají. Do fronty Delay jsou procesy zařazeny tehdy, jestliže vykonají operaci WaitUntil (čekání na zadaný čas) resp. Wait (pozdržení o zadaný časový interval). Parametrem těchto procedur je absolutní čas resp. požadované zpoždění. Parametry procedur Wait a WaitUntil jsou udány v časových jednotkách, které jsou shodné s časovými jednotkami proměnné RealTime (tedy po vydělení časových přerušení hodnotou ClockDivider+1). Kromě toho jsou ve frontě Delay uloženy procesy, které čekají na dokončení operací Send nebo Accept. V případě, že se jedná o operace s timeoutem (viz dále), jsou procesy umístěny ve frontě Delay podle toho, v jakém čase má timeout vypršet. Jestliže se jedná o operace bez timeoutu, jsou uloženy na konci fronty Delay, jako by čekaly na nekonečný čas.

11.4. Přerušovací zásobník

Před zavoláním procedury StartTimeSlicing je třeba inicializovat zásobník, na který jádro přepíná při vzniku přerušení. Inicializace tohoto zásobníku se provádí procedurou InitInterruptStack, jejímiž parametry je maximální počet přerušení, které se mohou vyskytnout současně (časového i ostatních), spolu s maximální délkou zásobníku potřebnou pro zpracování jednoho přerušení (viz kapitola „14 Přerušení“).

V následujícím příkladu je ukázáno použití proměnné RealTime a procedury WaitUntil pro plánování operací bez kumulace časových chyb. Proces vypíše zprávu každých cca 5 sekund bez ohledu na režijní dobu operace Write.

příklad

```
uses Kernel;
var Time: TimeType;

begin
  StartMain(0, 0);
  { inicializace přerušovacího zásobníku }
  InitInterruptStack(1, 60);
  { odstartování správy času }
  StartTimeSlicing(8);
  { nastavení děliče, aby časová jednotka byla cca 1 s }
```

```
  ClockDivider:=18;
  Time:=RealTime;
  repeat
    Time:=Time+5;
    WaitUntil(Time);
    Writeln('Uplynulo ', Time, ' sekund');
  until false;
end.
```

11.5. Zpracování přetečení času

Vzhledem k tomu, že si jádro udržuje hodnotu absolutního času v proměnné RealTime, došlo by po určité době k přetečení této proměnné a k nesprávné funkci správy času. Při použití standardního systémového časovače (55 ms) a při implicitní hodnotě proměnné ClockDivider se jedná o dobu cca 1362 dní. Aby však bylo možno provozovat jádro v extrémních podmínkách nepřetržitě, je deklarována proměnná jádra MaxRealTime, která umožňuje ošetření přetečení času.

Pokud by měla hodnota proměnné RealTime přesáhnout hodnotu proměnné MaxRealTime, je proměnná RealTime vynulována a ode všech absolutních časů, na které čekají procesy uložené ve frontě Delay, je odečtena hodnota proměnné MaxRealTime. Stejným způsobem musí uživatel v tomto okamžiku modifikovat své proměnné, pokud si v nich udržuje hodnotu absolutního času (proměnná Time v předcházejícím příkladě). To lze provést procesem s nejvyšší statickou prioritou, který se po dosažení hodnoty MaxRealTime spustí. Tento proces může vypadat např. takto:

příklad

```
uses Kernel;
{ maximální použitá hodnota zpoždění nebo timeoutu }
const MaxWait=1000;
{ uživatelské proměnné s hodnotou absolutního času }
var TimeA, ... , TimeZ: TimeType;

{$F+}
procedure MaxTime;
{$F-}
begin
  { nastavení maximální statické priority }
  Start('MaxTime', 1024, MaxStatPri, 0); Exit;
  repeat
    { čekání na čas MaxRealTime }
    WaitUntil(MaxRealTime);
    { úprava všech uživatelských proměnných obsahujících absolutní čas }
    Dec(TimeA, MaxRealTime);
    ...
    Dec(TimeZ, MaxRealTime);
  until false;
end;

begin
  StartMain(0, 0);
  InitInterruptStack(1, 60);
  StartTimeSlicing(8);
  MaxRealTime:=MaxLongInt-MaxWait;
  Wait(100);
```

...

Implicitní hodnota proměnné MaxRealTime je MaxLongInt. Při operaci Wait resp. při operacích s timeoutem provádí jádro implicitně výpočet RealTime+zadané zpoždění. Aby tyto výpočty nepřetekly, musí se pomocí proměnné MaxRealTime omezit hodnota RealTime tak, aby se všechny výše uvedené výpočty vešly do rozsahu LongInt.

12. Vypršení prodlevy (timeout)

Operace Accept a Receive na jakoukoliv schránku a operace Send na synchronní schránku může způsobit pozastavení procesu, který tuto operaci provedl. Aby bylo možno ošetřit případ, že zpráva nepřijde resp. není odebrána v konečném čase, je možno zavolat tyto operace s maximální prodlevou neboli timeoutem.

12.1. Zadání timeoutu

K tomu slouží operace SendTimeOut resp. SendEmptyTimeOut a AcceptTimeOut, AcceptVarTimeOut, ReceiveTimeOut, ReceiveVarTimeOut. Všechny tyto operace jsou implementovány jako funkce (na rozdíl od procedur Send, SendEmpty a Accept) a mají jako další vstupní parametr typu TimeType maximální dobu - timeout, po kterou mají čekat na zaslání resp. odebrání zprávy. Timeout je udáván v časových jednotkách (viz kapitola „11 Správa času“).

12.2. Zpracování timeoutu

Funkce SendTimeOut a SendEmptyTimeOut vrací booleovskou hodnotu True, pokud byla zpráva odebrána před vypršením prodlevy, jinak vrací hodnotu False. Funkce AcceptTimeOut a AcceptVarTimeOut se při vypršení prodlevy vrací normálním způsobem a vrací hodnotu True. Funkce ReceiveTimeOut a ReceiveVarTimeOut při vypršení prodlevy vrací hodnotu 0.

příklad

```
uses Kernel;
var
  S: SyncMailBox;
  D: MailBox;
  I: integer;

...
{ inicializace schránky S jako synchronní }
InitSyncMailBox(S, 'S', Normal);
{ inicializace schránky D jako dynamické }
InitDynMailBox(D, 'D', SizeOf(Integer), Normal);
...
{ synchronní zaslání zprávy s max. prodlevou 50 }
if SendTimeOut(S, I, SizeOf(I), 50) then
else Writeln('Došlo k vypršení timeoutu!');
...
{ příjem zprávy s max. prodlevou 20 }
if Arrived(D, J, SizeOf(J)) then
```

```
  Writeln('Přišla zpráva o hodnotě ', J);
else if AcceptTimeOut(20) then
  Writeln('Došlo k vypršení timeoutu')
. . .
```

13. Chráněná sekce

Jádro nabízí několik možností, jak zajistit výlučný přístup ke sdíleným prostředkům. Je např. možno svěřit tento prostředek pouze jednomu procesu, který vyřizuje požadavky ostatních procesů. Je také možno chránit prostředek pomocí synchronizace procesů realizované schránkou-čítačem použitou jako semafor. Nejjednodušší způsob je ovšem chránit výlučný přístup chráněnou sekcí, která je tvořena voláním procedur Lock a UnLock. Procesu, který zavolal proceduru Lock, je dočasně přiřazena statická priorita, která je větší, než maximální priorita, kterou může přiřadit uživatel (MaxStatPri). Voláním procedury UnLock je mu obnovena jeho původní priorita. Obdobně je zvýšena priorita procesu Dummy, takže v chráněné sekci je možno používat procedury Wait a WaitUntil, aniž by došlo ke spuštění jiného procesu. Použití procedur Receive nebo Accept v zamčené sekci k příjmu zprávy od jiného procesu způsobí deadlock.

příklad

```
...
Lock;
GotoXY(10, 10);
{ Pokud by v tomto místě došlo k přerušení a procesor
  byl přidělen jinému procesu, mohl by tento nový
  proces přesunou kurzor na jiné místo. }
Writeln('Tento text má být na pozici 10, 10');
UnLock;
...
```

Je-li procedura UnLock zavolána bez předchozího volání procedury Lock, vznikne chyba.

13.1. Zamykání jádra

Datové struktury jádra (fronta Ready, fronta Delay atd.) musí být také chráněny proti vícenásobnému přístupu. Procedury jádra, které modifikují vnitřní datové struktury jádra, jsou proto vytvořeny jako chráněné sekce. Na začátku takových procedur je zavolána procedura LockKernel a na konci procedura UnLockKernel. V interface jádra je u těchto procedur poznámka, že zamykají jádro.

Při provádění zamčených procedur jádra mohou přicházet přerušení. Reakce na přerušení je však poněkud jiná než jindy. Při časovém přerušení se pouze inkrementuje proměnná RealTime, avšak neprovádí se změna priority běžícího procesu a s tím spojená reorganizace fronty Ready, a také se neprovádí přesun procesů, které čekají na dosažený čas, z fronty Delay do fronty Ready. Tyto akce jsou provedeny až při ukončení procedury jádra procedurou UnLock. Obdobně jsou zpracována ostatní přerušení. Jsou prováděny procedury pro ošetření přerušení a tedy i zasílány zprávy na přerušovací schránky, avšak procesy, které na tyto zprávy od

přerušovacích procedur čekají, nejsou přeřazeny z fronty Delay do fronty Ready. Tento přesun se také provede až při ukončení procedury jádra procedurou UnLock.

14. Přerušení

V zásadě je třeba rozlišovat přerušení od časovače a ostatní přerušení. Ošetření přerušení od časovače je součástí jádra ReTOS. Toto ošetření může být volitelně navázáno na standardní časové přerušení pod systémem MS-DOS (int 8) nebo může být navázáno na libovolné jiné přerušení, které si uživatel zvolí. To umožňuje využít pro ovládání časového přerušení jiný časovač než ten, který je standardní součástí počítače. Navázání časového přerušení se provádí procedurou StartTimeSlicing, jejímž parametrem je číslo přerušení, na které chceme ošetření časového přerušení navázat. Pokud se tato procedura nezavolá, nefunguje dynamická priorita, timeouty, operace Wait a WaitUntil a procesor může být procesu odebrán pouze tehdy, jestliže proces zavolá nějakou operaci jádra. Před zavoláním procedury StartTimeSlicing je třeba provést inicializaci přerušovacího zásobníku procedurou InitInterruptStack (viz dále).

14.1. Přerušovací schránka

Jelikož v procedurách pro ošetření přerušení nelze používat systémové operace, je třeba složitější zpracování přerušení provádět v procesech, kterým se informace o přerušení předá. Předání se provede pomocí vyhražené přerušovací schránky (InterruptMailBox) a procedury SendInterrupt resp. SendInterruptEmpty. Zpráva z této schránky je pak přečtena procesem, který může provést složitější zpracování vzniklého přerušení.

14.2. Zpracování přerušení

Ošetřovací procedury je třeba psát podle dále uvedeného vzoru. Ošetřovací procedura, jejíž adresa se ukládá do přerušovacího vektoru, musí být pascalská procedura na nejvyšší úrovni bez parametrů a lokálních proměnných. Tato přerušovací procedura nesmí mít klasifikaci interrupt a při její deklaraci musí být vypnuta kontrola přetečení zásobníku (direktiva $S-). Tělo ošetřovací procedury se skládá z volání inicializační procedury přerušení InterruptEntry, vlastního zpracování přerušení a volání ukončovací procedury přerušení InterruptExit resp. FastInterruptExit. Vzhledem k tomu, že ošetřovací procedura nesmí obsahovat lokální proměnné, je nejlépe vlastní ošetření přerušení popsat jako samostatnou proceduru.

14.3. Začátek zpracování přerušení

Úkolem procedury InterruptEntry je především přepnutí na přerušovací zásobník a úschova registrů. Na běžném zásobníku nemusí být dostatečný prostor pro úschovu registrů, protože proces může právě vykonávat operaci systému, která přepíná na vlastní zásobník. Procedura InterruptEntry také povolí další přerušení.

14.4. Omezení při zpracování přerušení

Vlastní zpracování přerušení závisí na požadavku uživatele. Pro vlastní zpracování přerušení platí následující omezení:

- nesmí být použita žádná funkce operačního systému, která přepíná na vlastní zásobník. Pokud by totiž v tomto okamžiku vzniklo další přerušení, byla by porušena konzistence datových struktur jádra.

- nesmí být použita žádná funkce jádra s výjimkou SendInterrupt resp. SendInterruptEmpty. Tyto operace slouží k předání informace o přerušení, do vyhrazené přerušovací schránky.

Uvedené operace se od standardní operace Send liší tím, že zprávu uloží do schránky i v tom případě, že je nějakým procesem očekávána. V opačném případě by totiž musela tato operace předat zprávu čekajícímu procesu a zařadit ho do fronty Ready. To je však nepřípustné, neboť fronta Ready nemusí být v okamžiku přerušení konzistentní.

14.5. Ukončení zpracování přerušení

Poslední operací je dokončení přerušení procedurou InterruptExit resp. FastInterruptExit. Obě tyto procedury obnoví původní zásobník a původní hodnoty registrů. Procedura InterruptExit navíc předá zprávy z přerušovacích schránek čekajícím procesům a přeřadí je z fronty Delay do fronty Ready. To je však možné provést pouze tehdy, když přerušení nepřišlo během vykonávání funkce jádra (viz předchozí kapitola). Pokud přerušení přišlo během vykonávání funkce jádra, chová se procedura InterruptExit jako procedura FastInterruptExit a předání zpráv čekajícím procesům se provede po dokončení funkce jádra. Pokud má některý z čekajících procesů dostatečně velkou prioritu, je po ukončení přerušení okamžitě spuštěn a tím je zaručena bezprostřední reakce na přerušení.

14.6. Rychlé ukončení přerušení

Procedura FastInterruptExit tento přesun neprovádí a předání zpráv z přerušovacích schránek čekajícím procesům a jejich přesun do fronty Ready se provede až po ukončení časové jednotky nebo při nejbližším vyvolání procedury UnLock.

Při použití procedury InterruptExit je režie přerušení cca 800 us, zatímco při použití procedury FastInterruptExit cca 100 us. Uvedené údaje platí pro počítač AT286/12 MHz.

14.7. Inicializace zpracování přerušení

Procedura pro ošetření přerušení se naváže na určené přerušení procedurou SetInterruptHandler. Parametry této procedury jsou: číslo přerušení, adresa procedury pro ošetření přerušení a identifikace schránky, která je pro přerušení rezervovaná. Pouze tato schránka pak může sloužit pro předání hodnoty mezi ošetřující procedurou a procesem, který provede další zpracování přijaté hodnoty. Použitá schránka musí být

asynchronní schránka, která nesmí být dynamická nebo s přepisováním (je povolená schránka typu InterruptMailBox inicializovaná procedurou InitStatMailBox resp. InitCounterMailBox resp. InitSingleMailBox). Na přerušovací schránku může posílat zprávy pouze procedura pro ošetření přerušení, pro které je schránka rezervována. Na druhé straně může z této schránky číst zprávy více procesů.

příklad

```
uses Kernel;
const CisloPreruseni = $78;
var SchrankaPreruseni: InterruptMailBox;

procedure CtiPort; { vlastní ošetření přerušení }
var I : Integer;
begin
  I:=PortW[33]; { sejmutí hodnoty z portu }
  { odeslání hodnoty na schránku přerušení }
  SendInterrupt(SchrankaPreruseni, I, SizeOf(I));
end;

{ procedura pro ošetření přerušení nesmí mít parametry a lokální proměnné }
{$F+,S-}
procedure CtiPortInt;
begin
  InterruptEntry; { příprava přerušení }
  CtiPort; { vlastní ošetření přerušení }
  InterruptExit; { ukončení přerušení }
end;
{$F-,S+}

{$F+ proces pro zpracování přečtených hodnot }
procedure Zpracovani;
{$F-}
var I : Integer;
begin
  Start('Zprac', 1024, 1, 255); Exit;
  repeat
    if Arrived(SchrankaPreruseni, I, SizeOf(I)) then
    begin
      Lock;
      Writeln('Hodnota portu 33 při přerušení je: ', I);
      UnLock;
    end
    else Accept;
  until false;
end;

begin
  StartMain(0, 255);
  { inicializace statické schránky pro přerušení }
  InitStatMailBox(SchrankaPreruseni, '', SizeOf(Integer), 5, Normal);
  { inicializace přerušovacího zásobníku }
  InitInterruptStack(1, 60);
  Zpracovani; { odstartování procesu pro zpracování přerušení }
  { inicializace vektoru přerušení }
  SetInterruptHandler(CisloPreruseni, CtiPortInt, SchrankaPreruseni);
  ...
```

14.8. Stanovení velikosti přerušovací schránky

Velikost schránky pro přerušení musí být taková, aby se v žádné očekávané situaci nepřeplnila. Schránka pro přerušení se plní tehdy, jestliže přerušení přijde v době, kdy je prováděna některá funkce jádra nebo jestliže se proces nachází v zamčené sekci (Lock, UnLock). Po opuštění jádra resp. po provedení operace UnLock se schránka postupně vyprazdňuje, jak si proces pro zpracování jednotlivé hodnoty odebírá.

14.9. Inicializace přerušovacího zásobníku

Před instalací přerušovací procedury je třeba inicializovat zásobník, na který jádro přepíná při vzniku přerušení. Inicializace tohoto zásobníku se provádí procedurou InitInterruptStack, jejímiž parametry je maximální počet přerušení, které se mohou vyskytnout současně (včetně časového), spolu s maximální délkou zásobníku potřebnou pro zpracování jednoho přerušení. Procedura InitInterruptStack rezervuje na heapu místo na MaxInterruptLength * MaxInterruptCount + 512 bytů. Současný výskyt přerušení vznikne tehdy, když další přerušení přijde dříve, než je dokončeno zpracování prvního. Jádro uschovává na tento zásobník při vzniku přerušení cca 30 bytů. Kromě toho jsou na tomto zásobníku prováděny procedury, které uživatel zavolá ve vlastním ošetření přerušení. Pro paralelní programy, které využívají správu času, ale neošetřují žádné další přerušení, je počet současně vzniklých přerušení MaxInterruptCount=1 a délka zásobníku pro zpracování časového přerušení MaxInterruptLength=60.

!!!Pozor !!! Pro každé přerušení musí být definována samostatná schránka. Na tuto schránku určenou pro přerušení může zasílat zprávy pouze procedura pro ošetření příslušného přerušení a pouze pomocí procedury SendInterrupt a SendInterruptEmpty. Na druhé straně může i více procesů z této schránky hodnoty odebírat.

15. Používání aritmetického koprocesoru a softwarového emulátoru

15.1. Ošetření aritmetického koprocesoru

V případě používání aritmetického koprocesoru (direktiva $N+) je třeba při každém přepnutí procesu, ve kterém se provádějí aritmetické operace, uschovat resp. obnovit obsah aritmetického koprocesoru. K tomu slouží pomocné procedury volané při předávání a odebírání procesoru, které se nastavují procedurami SetEntryProc a SetExitProc (viz kapitola „5 Procesy“). Při každém opuštění procesu se musí zavolat procedura jádra Save8087, která uschová obsah koprocesoru do vyhrazeného bufferu, a při vstupu do procesu musí být obsah obnoven procedurou jádra Restore8087. Před prvním použitím koprocesoru v procesu je ho třeba inicializovat procedurou jádra Init8087.

15.2. Inicializace emulátoru koprocesoru

Při použití emulátoru koprocesoru (direktiva $E+) je potřeba na začátku každého procesu, ve kterém se provádějí aritmetické operace, provést po startovací sekvenci inicializaci emulátoru procedurou jádra InitEmulator. Je-li použita direktiva $E+, je normální program v Turbo Pascalu nezávislý na přítomnosti koprocesoru. Chceme-li tuto vlastnost zachovat při použití jádra, musíme program koncipovat podle následujícího příkladu.

příklad

```
{$N+,E+,F+}
uses Kernel;

procedure P;
var
  A: Double;
  Buff: Buff8087;

procedure EntryProc;
begin
  Restore8087(Buff);
end;

procedure ExitProc;
begin
  Save8087(Buff);
end;

begin
  Start('P', 1000, 1, 0); Exit;
  { zjištění přítomnosti koprocesoru }
  if Test8087=0 then
    InitEmulator
  else
  begin
    SetEntryProc(@EntryProc);
    SetExitProc(@ExitProc);
    Init8087;
  end;
  A := 3.14;
  ...
```

16. Ladění a hlášení chyb

16.1. Trasování programu

Provádění programu je možno trasovat. Při trasování se dále uvedené události zapisují do zvoleného textového souboru. Soubor se inicializuje procedurou InitTraceFile, jejímž parametrem je jméno trasovacího souboru. Vlastní trasování se spouští přiřazením zvolené množiny příznaků do proměnné TraceStatus. Jednotlivé příznaky řídí obsah trasování:

TraceStart - výpis operací Start
TraceTerminate - výpis operací Terminate
TraceSend - výpis operací Send
TraceReceive - výpis operací Receive a Accept
TraceWait - výpis operací Wait
TraceAbort - výpis operací Abort
TraceTime - výpis času operace
TraceName - výpis jména procesu, který událost vyvolal
TraceMessage - hexadecimální výpis zprávy v operacích Send a Accept

Přiřazením jména do proměnné TraceIdent je možno zvolit trasování pouze pro zvolené procesy. Je možno použít hvězdičkovou konvenci.

16.2. Chyby jádra

Pokud je to možné, kontroluje jádro korektní provádění všech operací. Při vzniku chyby v jádře je program ukončen s runtimovou chybou. Jádro zavádí tyto další runtimové chyby:

50 - reálný čas je větší než požadovaný ve WaitUntil
51 - přerušovací schránka nesmí být dynamická nebo over
52 - počet a délka prvků statické fronty musí být nenulová
53 - zápis do plné fronty
54 - výběr z prázdné fronty
55 - ve startovací sekvenci chybí exit
56 - chybná instrukce návratu - chyba ReTOS
57 - délka zprávy je jiná než délka bufferu
58 - schránka je zaplněna
59 - jádro není inicializováno
60 - volání procedury UnLock bez předchozího volání procedury Lock
62 - není inicializován zásobník pro přerušení
63 - přišlo více uživatelských přerušení než je deklarováno
64 - výjimka není globální proměnná
65 - vyvolání neošetřené výjimky
66 - proces není vzdáleně volaná procedura

17. Výjimky

Jádro umožňuje deklaraci výjimek a jejich ošetření pomocí handlerů. Touto konstrukcí je kompenzována nepříjemná vlastnost jazyka Turbo Pascal, který nedovoluje opouštět procedury skokem. Výjimkou nazýváme nestandardní situaci, která může nastat v průběhu výpočtu a kterou si přejeme ošetřit zvláštní sekvencí tzv. handlerem. Po ošetření výjimky probíhá výpočet od místa, kde byl handler instalován. Výjimka tedy umožňuje opuštění rozpracovaných procedur a přenos řízení do libovolného místa v programu.

17.1. Handlery

V jádře je výjimka globální proměnná typu procedura. Handlery jsou procedury, které se spouštějí při tzv. vyvolání výjimky. Ke každé výjimce může existovat libovolně mnoho handlerů, které se musí explicitně instalovat a rušit. Vyvolání výjimky způsobí zavolání posledně nainstalovaného handleru. Před opuštěním procedury, ve které byl handler nainstalován, se musí jeho instalace zrušit a tím se pro příslušnou výjimku obnoví platnost předchozí instalace handleru. Pokud handler není zrušen, je další činnost nedefinovaná.

17.2. Parametry handlerů

Handlery mohou mít parametry. Parametrové části všech handlerů příslušných k jedné výjimce musí být stejné a odpovídat typu proměnné výjimka. Každý handler musí být dlouze volaná procedura na nejvyšší úrovni.

17.3. Proměnná-výjimka

Výjimka a tedy její posledně instalovaný handler se i s parametry vyvolává pomocí proměnné-výjimka, jako by byla adresa tohoto handleru uložena do této proměnné-výjimka.

17.4. Instalace handleru

Výjimky se instalují funkcí InstallHandler. Funkce InstallHandler instaluje handler pro určenou výjimku a vrátí hodnotu False. Handler se po svém ukončení vrátí do místa, ve kterém byl instalován, avšak vrátí hodnotu True.

17.5. Prázdný handler

Místo adresy handleru je možno použít hodnotu nil. V tom případě se při vzniku výjimky provede pouze návrat do místa, ve kterém byl tento prázdný handler instalován.

17.6. Rušení handlerů

Všechny instalace handlerů v proceduře se musí explicitně zrušit procedurou DeleteHandlers.

17.7. Sdílení handlerů

Proměnné výjimka a handlery mohou být sdílené více procesy. Instalace handlerů a vyvolání výjimky však platí pouze pro proces, který ji provedl. Handlery mohou být instalovány i v hlavním programu, avšak až po provedení procedury StartMain.

Vyvolání výjimky, pro kterou nebyl instalován žádný handler, způsobí zhroucení programu.

příklad

```
uses Kernel;
{ typ výjimky }
type ExcType = procedure(S: string);
var Exc : ExcType; { proměnná výjimka }

{$F+ handler pro výjimku Exc }
procedure HandlerExc(s: string);
{$F-}
begin
  Writeln('Byl vyvolán HandlerExc s parametrem: ',s)
end;

{$F+ proces, ve kterém je instalován handler }
procedure TestHandler;
begin
  Start('TestHandler', 1024, 0, 255); Exit;
  { instalace handleru pro výjimku Exc }
  if InstallHandler(@HandlerExc, Exc) then
    { místo, ve kterém se pokračuje po vyvolání výjimky }
    Terminate
  else
  begin
    { místo, kde se pokračuje po instalaci handleru }
    ...
    { vyvolání výjimky s parametrem }
    Exc('Vznik výjimky Exc');
  end;
  DeleteHandlers;
end;

begin
  StartMain(0, 0);
  TestHandler;
  TerminateMain;
end.
```

18. Inicializace a ukončení činnosti jádra

Tato kapitola shrnuje informace z předcházejícího textu, které jsou nezbytné pro správné spuštění a ukončení činnosti jádra.

Před vlastním během paralelního programu je třeba jádro inicializovat.

18.1. Základní inicializace

Základní inicializaci jádra včetně inicializace fronty Ready a fronty Delay provádí procedura StartMain. Procedura dále zařadí hlavní proces do fronty Ready, spustí proces Dummy a zařadí ho do fronty Ready. Potom je již možno startovat další procesy.

18.2. Inicializace schránek

Každou schránku je třeba před použitím inicializovat inicializačními procedurami. Inicializace schránek je možno provádět až po zavolání procedury StartMain.

18.3. Inicializace přerušovacího zásobníku

Je-li součástí programu ošetření přerušení, ať již přerušení od časovače nebo jiného, musí uživatel inicializovat zásobník, na který jádro přepíná při vzniku přerušení. Inicializace tohoto zásobníku se provádí procedurou InitInterruptStack. Procedura InitInterruptStack musí být zavolána před procedurami StartTimeSlicing nebo SetInterruptHandler.

18.4. Inicializace časového přerušení

Vyžaduje-li uživatel časovou správu procesů (t.j. fungování dynamických priorit, timeoutů a procedur Wait, WaitUntil), je třeba inicializovat časové přerušení procedurou StartTimeSlicing.

18.5. Nastavení časové jednotky

Podle frekvence použitého časovače a podle toho, jak velkou požadujeme základní časovou jednotku, je třeba nastavit proměnnou ClockDivider.

18.6. Ošetření přetečení času

Pokud požadujeme nepřetržitý chod systému, je třeba nastavit maximální hodnotu absolutního času MaxRealTime a spustit proces popsaný v odstavci Správa času, který čeká na dosažení tohoto času a který upraví hodnoty proměnných v uživatelském programu obsahující absolutní čas.

18.7. Inicializace ostatních přerušení

Pokud je součástí programu ošetření jiného přerušení než od časovače, musí být tato ošetření instalována procedurou SetInterruptHandler. Přerušovací schránka, která je parametrem procedury SetInterruptHandler, musí být před zavoláním této procedury již inicializovaná.

18.8. Ukončení činnosti jádra

Činnost jádra se ukončuje procedurou TerminateMain. Procedura může být zavolána pouze v hlavním procesu a pozastaví hlavní proces, dokud všechny spuštěné procesy nedoběhnou. Pokud nechceme čekat na dokončení všech procesů, je nutno počet procesů, na jejichž dokončení se nemá čekat přiřadit do proměnné

MinProcessNo. Procesy, na jejichž dokončení se nemá čekat jsou procedurou TerminateMain ukončeny.

18.9. Uživatelské ukončení

Procedura TerminateMain dále zavolá proceduru bez parametrů, jejíž adresa je uložena v proměnné UserTerminate. Do této proměnné může uživatel uložit adresu vlastní procedury pro ukončení činnosti jádra.

Nakonec procedura TerminateMain obnoví původní hodnoty přerušovacích vektorů platných v okamžiku volání procedury StartMain. Program dále pokračuje v hlavním programu za voláním procedury TerminateMain.

18.10. Znovuspuštění jádra

Činnost jádra může být v programu opakovaně zahajována a ukončována. Jádro automaticky uvolňuje paměť alokovanou na heapu s výjimkou vyrovnávacích pamětí vyhrazených při inicializaci schránek a neodebraných zpráv v dynamických schránkách. Pokud by tato okolnost byla na závadu, lze předřadit před operaci StartMain volání standardní procedury Mark a za operací TerminateMain zavolat standardní proceduru Release.

18.11. Inicializace trasování

Při požadavku na trasování je nutno soubor pro trasování inicializovat procedurou InitTraceFile. Vlastní trasování se spouští přiřazením množiny trasovacích příznaků do proměnné TraceStatus. Pokud se trasování má týkat pouze vybraných procesů, je možné jejich jméno (je možno použít hvězdičkovou konvenci) přiřadit do proměnné TraceIdent.

19. Omezení

V Turbo Pascalu 6.0 nelze použít direktivu $G+ pro generování instrukcí procesoru 80286.

20. Přehled procedur a funkcí

20.1. Procesy

Start	Odstartování nového procesu.
SetEntryProc	Určení procedury volané při přidělování procesoru.
SetExitProc	Určení procedury volané při odebrání procesoru.
Terminate	Ukončení procesu.
Abort	Ukončení všech procesů dané specifikace.
GetPriority	Zjištění statické a dynamické priority.
RunningProcess	Zjištění jména běžícího procesu.
SetPriority	Nastavení nové hodnoty statické a dynamické priority.

SetPriorityIdent	Nastavení nové hodnoty statické a dynamické priority specifikovaným procesům.
SimulationTime	Zjištění času procesoru přiděleného běžícímu procesu.
StartMain	Inicializace jádra a spuštění hlavního procesu.
TerminateMain	Ukončení hlavního procesu.
Wait	Zpoždění procesu o daný počet časových jednotek proces.
WaitUntil	Zpoždění procesu definované absolutní hodnotou času.
ProcessCount	Zjištění počtu všech procesů v systému.
ReadyCount	Zjištění počtu procesů ve frontě Ready.
DelayCount	Zjištění počtu procesů ve frontě Delay.

20.2. Schránky

Accept	Čekání na příchod zprávy.
AcceptTimeOut	Čekání na příchod zprávy s max. čekací dobou.
AcceptVar	Čekání na příchod zprávy proměnné délky.
AcceptVarTimeOut	Čekání na příchod zprávy proměnné délky s max. čekací dobou.
Receive	Čekání na příchod zprávy.
ReceiveTimeOut	Čekání na příchod zprávy s max. čekací dobou.
ReceiveVar	Čekání na příchod zprávy proměnné délky.
ReceiveVarTimeOut	Čekání na příchod zprávy proměnné délky s max. čekací dobou.
Send	Odeslání zprávy na schránku.
SendEmpty	Odeslání prázdné zprávy na schránku.
SendTimeOut	Odeslání zprávy na schránku s max. čekací dobou.
SendEmptyTimeOut	Odeslání prázdné zprávy na schránku s max. čekací dobou.
SendInterrupt	Odeslání zprávy na schránku příslušející danému přerušení.
SendInterruptEmpty	Odeslání prázdné zprávy na schránku příslušející danému přerušení.
Arrived	Označení schránky, ze které chce proces voláním Accept číst zprávu.
ArrivedEmpty	Označení schránky, ze které chce proces voláním Accept číst prázdnou zprávu.
Choice	Označení schránky, na které má funkce Receive očekávat zprávu.
ChoiceEmpty	Označení schránky, na které má funkce Receive očekávat prázdnou zprávu.
Purge	Vymazání všech zpráv z dané schránky.
InitCounterMailBox	Vytvoření asynchronní schránky tvořené čítačem prázdných zpráv.
InitSingleMailBox	Vytvoření schránky na jednu zprávu.
InitSingleOverMailBox	Vytvoření schránky uchovávající pouze poslední zprávu.
InitStatMailBox	Vytvoření asynchronní schránky na pevný počet zpráv.
InitDynMailBox	Vytvoření asynchronní dynamické schránky.
InitVarMailBox	Vytvoření asynchronní dynamické schránky s proměnnou délkou zpráv.
InitSyncMailBox	Vytvoření synchronní schránky.
ProcCount	Zjištění, kolik procesů čeká u schránky na zprávu.
MessageCount	Zjištění, kolik zpráv je ve schránce přítomno.
Empty	Zjištění zda schránka je prázdná.
Full	Zjištění zda schránka je plná.
FirstMessage	Poskytnutí adresy první zprávy ve schránce.
NextMessage	Poskytnutí adresy následující zprávy ve schránce.
MessageLength	Poskytnutí délky následující zprávy ve schránce.
_Full	Zjištění, zda se do schránky vejde správa o dané velikosti.

20.3. Ostatní

BreakPoint	Vloží přerušení breakpointem.
Match	Zjištění, zda zadané jméno odpovídá zadané masce.
POPF	Načtení příznaků procesoru.
PUSHF	Uložení příznaků procesoru.
DI	Zákaz přerušení.
EI	Povolení přerušení.
Lock	Počátek chráněné sekce.
UnLock	Konec chráněné sekce.
TestLock	Zjištění, zda je proces v chráněné sekci.
LockKernel	Počátek chráněné sekce.
UnLockKernel	Konec chráněné sekce.
TestLockKernel	Zjištění, zda je proces v chráněné sekci.
TestKernel	Zjištění, zda je instalováno jádro o.s. ReTOS.
SetInterruptHandler	Instalace obslužné procedury přerušení.
InterruptEntry	Zahájení přerušovací procedury.
InterruptExit	Ukončení přerušovací procedury.
FastInterruptExit	Ukončení přerušovací procedury.
StartTimeSlicing	Určení systémového časovače.
InitInterruptStack	Určení zásobníku pro ošetření přerušení.
InitTraceFile	Inicializace trasovacího souboru.
InstallHandler	Instalace handleru výjimky.
DeleteHandlers	Zrušení instalace handlerů.
InitEmulator	Inicializace programového emulátoru matematického koprocesoru.
Init8087	Inicializace matematického koprocesoru.
Save8087	Uložení stavu matematického koprocesoru.
Restore8087	Obnova stavu matematického koprocesoru.
Identification	Identifikace jádra pro registraci.

21. Popis konstant a typů

```
Const
  Version = 'ReTOS1703200301';
  cVerNo  = $0302; { BCD formát }
  cVer    = '03.02,17.03.2003';
```

Verze ReTOSu je popsaná v samostatném souboru „*Version.inc*“.

```
  IdentLength = 15;      { délka identifikačního stringu }
  MainIdent   = 'Main';  { jméno hlavního procesu }
  DummyIdent  = 'Dummy'; { jméno procesu zahaleč }
  MaxStatPri  = $3ff0;   { maximální statická priorita }
  KernelFl:Boolean = FALSE; ( příznak, že je jádro inicializováno }
  OriginTimeIntNo: Byte=0;  { standardní číslo přerušení od časovače}

type
  IdentType   = String[IdentLength]; { typ jména procesu }
  ModeType     = (Normal, Broadcast); { mód schránky }
  TimeType     = Longint;   { typ absolutního času }
  ProcTerminate= procedure; { typ ukončovací procedury }
  IntProc      = procedure; { typ (de)aktivační procedury}
  PriorityType = 0..MaxStatPri; { typ statické priority }

  TraceMode = (               { trasovací příznaky }
               TraceStart,    { výpis operací Start }
               TraceTerminate, { výpis operací Terminate }
               TraceSend,     { výpis operací Send }
```

```
                TraceReceive,   { výpis operací Receive a Accept }
                TraceWait,      { výpis operací Wait }
                TraceAbort,     { výpis operací Abort }
                TraceTime,      { výpis času operace }
                TraceName,      { výpis jména procesu }
                TraceMessage    { hexadecimální výpis zprávy }
               );
  TraceModes     = set of TraceMode;
```

22. Popis proměnných

RealTime : TimeType;

V proměnné RealTime je udržován systémový čas. Jeho hodnota odpovídá počtu pulsů systémového časovače vyděleného hodnotou ClockDivider, to celé modulo MaxRealTime. Systémový časovač je určen voláním procedury StartTimeSlicing. Tuto proměnnou je možné pouze číst a při jejím čtení musí být proces v chráněné sekci voláním procedur LockKernel a UnLockKernel nebo Di a Ei. RealTime nečastěji použijeme ve spojitosti s procedurou WaitUntil, při čekání na daný systémový čas.

MaxRealTime : TimeType;

MaxRealTime určuje maximální absolutní čas. Při jeho dosažení se nuluje proměnná RealTime a upravují se časy ve frontě Delay.

ClockDivider : Word;

Udává dělící poměr pro časové přerušení dané systémovým časovačem. Nastavením této proměnné lze snížit četnost přehodnocování front Ready a Delay.

UserTerminate : ProcTerminate;

Uživatelská procedura volaná při ukončení práce jádra operačního systému.

MinProcesNo : Word;

Udává počet procesů, na jejichž dokončení se v TerminateMain nečeká. Implicitní hodnota je 0.

TraceStatus : TraceModes;

Příznak pro trasování jádra.

TraceIdent : IdentType;

Nejednoznačné jméno trasovaných procesů.

DummyEntryProc : IntProc;

Aktivační procedura zahaleče.

DummyExitProc : IntProc;

Deaktivační procedura zahaleče.

LogFilePtr : ^Text;

Ukazatel na soubor pro výpis systémových zprav.

SaveSP : Word;

Pomocná proměnná pro uchovávání Stack Pointeru.

23. Popis procedur a funkcí

23.1. Abort procedure

Procedura zruší všechny procesy, které odpovídají zadané identifikaci. Procesy mohou být v libovolném stavu, ve frontě Ready nebo Delay.

deklarace

```
procedure Abort(Identification: IdentType);
```

parametry

Identification určuje, které procesy budou zrušeny (max. 15 znaků). Znak '?' zastupuje libovolný znak, '*' zastupuje libovolnou posloupnost znaků.

také

Terminate, TerminateMain, Start, StartMain

23.2. Accept procedure

Procedura Accept očekává příchod zprávy na některou ze schránek určených předcházejícím voláním funkcí Arrived nebo ArrivedEmpty. Po přijetí zprávy, nebo je-li zpráva k dispozici již při zavolání procedury Accept, přesune procedura Accept přijatou zprávu do bufferu, který je zadán jako parametr příslušného volání funkce Arrived, a vrátí řízení do místa vyvolání funkce Arrived. Přitom vrací funkční hodnotu true. Před použitím schránky je třeba ji nejprve inicializovat a definovat její vlastnosti.

deklarace

```
procedure Accept;
```

také

```
AcceptTimeOut, AcceptVar, AcceptVarTimeOut, Arrived,
ArrivedEmpty
```

příklad

Příklad ukazuje použití funkcí Arrived a procedury Accept. Funkcemi Arrived určíme schánky u kterých má následné volání Accept očekávat zprávu. Po přijetí zprávy program pokračuje příkazem bezprostředně následujícím za symbolem then funkce Arrived, na jejíž schránku byla zpráva přijata. Schránku je třeba nejprve inicializovat.

```
var
  MB1, MB2, MB3 : MailBox;

...
if Arrived(MB1,...) then ...
else if Arrived(MB2,...) then ...
else if Arrived(MB3,...) then ...
else Accept;
...
```

23.3. AcceptTimeOut function

Funkce AcceptTimeOut očekává příchod zprávy na některou ze schránek určených předcházejícím voláním funkcí Arrived nebo ArrivedEmpty. Po přijetí zprávy, nebo je-li zpráva k dispozici již při zavolání funkce AcceptTimeOut, přesune funkce AcceptTimeOut přijatou zprávu do bufferu, který je zadán jako parametr příslušného volání funkce Arrived, a vrátí řízení do místa vyvolání funkce Arrived. Přitom vrací funkční hodnotu true. Na rozdíl od procedury Accept, je proces znovu spuštěn nejpozději po uplynutí doby dané parametrem TimeOut. Jestliže uplyne doba daná parametrem TimeOut, vrátí funkce obvyklým způsobem hodnotu true. Před použitím schránky je třeba ji nejprve inicializovat a definovat její vlastnosti.

deklarace

```
function AcceptTimeOut(TimeOut: TimeType): Boolean;
```

parametr

TimeOut určuje maximální dobu, po kterou funkce čeká na zprávu.

také

```
Accept, AcceptVar, AcceptVarTimeOut, Arrived, ArrivedEmpty
```

příklad

Příklad ukazuje použití funkcí Arrived a AcceptTimeOut. Funkcemi Arrived určíme schránky u kterých má následné volání AcceptTimeOut očekávat zprávu. Je-li přijata zpráva v časovém limitu, program pokračuje příkazem bezprostředně následujícím za then funkce Arrived, na jejíž schránku byla zpráva přijata. Není-li přijata zpráva v časovém limitu, navrátí funkce AcceptTimeOut hodnotu true a program pokračuje příkazem uvedeným za symbolem then této funkce. Schránku je třeba nejprve inicializovat.

```
var
  MB1, MB2, MB3 : MailBox;

...
if Arrived(MB1,...) then ...
else if Arrived(MB2,...) then ...
else if Arrived(MB3,...) then ...
else if AcceptTimeOut(20) then { TimeOut }
...
```

23.4. AcceptVar procedure

Procedura AcceptVar očekává příchod zprávy na některou ze schránek určených předcházejícím voláním funkcí Arrived a ArrivedEmpty. Po přijetí zprávy, nebo je-li zpráva k dispozici již při zavolání procedury AcceptVar, přesune procedura AcceptVar přijatou zprávu do bufferu, který je zadán jako parametr příslušného volání funkce Arrived, a vrátí řízení do místa vyvolání funkce Arrived. Přitom vrací funkční hodnotu true. Do výstupního parametru MessageLen je uložena skutečná délka přijaté zprávy. Před použitím schránky je třeba ji nejprve inicializovat a definovat její vlastnosti.

deklarace

```
procedure AcceptVar(var MessageLen: Word);
```

parametr

V parametru **MessageLen** je navrácena skutečná délka zprávy.

také

```
Accept, AcceptTimeOut, AcceptVarTimeOut, Arrived, ArrivedEmpty
```

23.5. AcceptVarTimeOut function

Funkce AcceptVarTimeOut očekává příchod zprávy na některou ze schránek určených předcházejícím voláním funkcí Arrived nebo ArrivedEmpty. Po přijetí zprávy, nebo je-li zpráva k dispozici již při zavolání funkce AcceptVarTimeOut, přesune funkce AcceptVarTimeOut přijatou zprávu do bufferu, který je zadán jako parametr příslušného volání funkce Arrived a vrátí řízení do místa vyvolání funkce Arrived. Přitom vrací funkční hodnotu true. Do výstupního parametru MessageLen je uložena skutečná délka přijaté zprávy. Na rozdíl od procedury AcceptVar, je proces znovu spuštěn nejpozději po uplynutí doby dané parametrem TimeOut. Jestliže uplyne doba daná parametrem TimeOut, vrátí funkce obvyklým způsobem hodnotu true. Před použitím schránky je třeba ji nejprve inicializovat a definovat její vlastnosti.

deklarace

```
function AcceptVarTimeOut(TimeOut: TimeType;
                 var MessageLen: Word): Boolean;
```

parametry

TimeOut určuje maximální dobu, po kterou funkce čeká na zprávu. V parametru MessageLen je navrácena skutečná délka přijaté zprávy.

také

```
Accept, AcceptTimeOut, AcceptVar, Arrived, ArrivedEmpty
```

23.6. Arrived function

Funkce určuje schránku, ze které chce proces číst zprávu. Schránka musí být inicializovaná. Funkce vrací vždy hodnotu false. Funkce musí být volána v podmínce příkazu if, v jehož else části je buď další příkaz if s funkcí Arrived nebo volání procedury Accept. Přijatá zpráva je procedurou Accept přesunuta do zadaného bufferu a řízení je vráceno do místa volání funkce Arrived, v jejíž parametrech byla zadána ta schránka, na kterou zpráva přišla.

deklarace

```
function Arrived(var MailBox: AbstractMailBox;
      var MessageAddress; MessageLength: Word) : Boolean;
```

parametry

Parametr MailBox označuje schránku, ze které chce proces přijmout zprávu. MessageAddress označuje adresu bufferu pro zprávu. MessageLength určuje maximální očekávanou délku zprávy.

také

```
ArrivedEmpty, Accept, AcceptTimeout, AcceptVar,
AcceptVarTimeOut
```

23.7. ArrivedEmpty function

Funkce určuje schránku, ze které chce proces číst prázdnou zprávu. Schránka musí být inicializovaná a musí to být buď synchronní schránka nebo asynchronní schránka čítač, inicializovaná procedurou InitCounterMailBox. Funkce vrací vždy hodnotu false. Funkce musí být volána v podmínce příkazu if, v jehož else části je buď další příkaz if s funkcí Arrived ArrivedEmpty nebo volání procedury Accept. Po přijetí prázdné zprávy je řízení vráceno do místa volání funkce Arrived, v jejíž parametrech byla zadána ta schránka, na kterou zpráva přišla.

deklarace

```
function ArrivedEmpty(var MailBox: AbstractMailBox): Boolean;
```

parametry

Parametr MailBox označuje schránku, ze které chce proces přijmout zprávu.

také

```
Arrived, Accept, AcceptTimeout, AcceptVar, AcceptVarTimeOut.
```

23.8. BreakPoint procedure

Vloží přerušení breakpointem – INT 3.

deklarace

```
procedure BreakPoint;
```

23.9. Choice procedure

Procedura určuje schránku, na které má následující volání funkce Receive očekávat zprávu. Schránka musí být inicializovaná. Před voláním funkce Receive je možno proceduru Choice volat vícekrát, a tak očekávat zprávu na více schránkách najednou. Parametr ChoiceNo udává identifikační číslo, které vrací funkce Receive, jestliže zpráva přišla na schránku určenou parametrem MailBox. Při vícenásobném volání procedury Choice je možné mít hodnotu parametru ChoiceNo stejnou. Přijatá zpráva je funkcí Receive přesunuta do zadaného bufferu.

deklarace

```
procedure Choice(ChoiceNo: Word;
           var MailBox: AbstractMailBox;
           var MessageAddress; MessageLength: Word);
```

parametry

ChoiceNo určuje hodnotu navracenou funkcí Receive, při příjmu zprávy. Parametr MailBox označuje schránku, ze které chce proces přijmout zprávu. MessageAddress označuje adresu bufferu pro zprávu. MessageLength určuje maximální očekávanou délku zprávy.

také

```
ChoiceEmpty, Receive, ReceiveTimeOut
```

23.10. ChoiceEmpty procedure

Procedura určuje schránku, na které má následující volání funkce Receive očekávat zprávu. Schránka musí být inicializovaná a musí to být buď synchronní schránka SyncMailBox nebo to musí být asynchronní schránka čítač MailBox, inicializovaná procedurou InitCounterMailBox. Před voláním funkce Receive je možno volat víckrát proceduru ChoiceEmpty a tak očekávat zprávu na více schránkách najednou. Parametr ChoiceNo udává identifikační číslo, které vrací funkce Receive, jestliže zpráva přišla na schránku určenou parametrem MailBox. Při vícenásobném volání procedury ChoiceEmpty je možné mít hodnotu parametru ChoiceNo stejnou. Přijatá zpráva je funkcí Receive přesunuta do zadaného bufferu.

deklarace

```
procedure ChoiceEmpty(ChoiceNo: Word;
                      var MailBox: AbstractMailBox);
```

parametry

ChoiceNo určuje hodnotu navracenou funkcí Receive, při příjmu zprávy. Parametr MailBox označuje schránku, ze které chce proces přijmout zprávu.

také

```
Choice, Receive, ReceiveTimeOut
```

23.11. DelayCount function

Funkce navrací počet procesů ve frontě Delay.

deklarace

```
function DelayCount: Word;
```

také

```
ProcessCount, ReadyCount
```

23.12. DeleteHandlers procedure

Procedura pro zrušení instalace handlerů. Procedura musí být zavolána na konci procedury, ve které jsou nainstalovány nějaké handlery.

deklarace

```
procedure DeleteHandlers;
```

také

```
InstallHandler
```

23.13. Di procedure

Zákaz přerušení procesoru.

deklarace

```
procedure DI;
```

23.14. Ei procedure

Povolení přerušení procesoru.

deklarace

```
procedure EI;
```

23.15. Empty function

Funkce vrací hodnotu true jestliže je schránka prázdná. U synchronních schránek to znamená, že žádný proces nečeká, aby mohl předat zprávu.

deklarace

```
function Empty(var MailBox: AbstractMailBox): Boolean;
```

parametr

parametr MailBox označuje schránku.

23.16. FastInterrupExit procedure

Procedura pro ukončení přerušení. Volá se v obslužné proceduře přerušení. Procedura obnoví původní zásobník a obnoví všechny registry. Na rozdíl od procedury InterruptExit nepředá hodnoty v přerušovacích schránkách procesům, které na tyto hodnoty čekají. Předání hodnot a zařazení do fronty Ready provede nejbližší volání procedury UnLock (ať volané explicitně nebo některou operací, která zamyká a odemyká jádro) nebo časové přerušení. Použití procedury FastInterruptExit je možné u přerušení, které nepřicházejí v kratších intervalech než cca 80 mikrosekund (platí pro 16 MHz).

deklarace

```
procedure FastInterruptExit;
```

také

```
InterruptEntry, InterruptExit
```

23.17. FirstMessage function

Funkce vrací adresu první zprávy ve schránce nebo hodnotu nil, jestliže je schránka prázdná.

deklarace

```
function FirstMessage(var MailBox: MailBox): pointer;
```

parametr

parametr MailBox označuje schránku.

23.18. Full function

Funkce vrací true jestliže je schránka plná. U dynamických schránek to znamená zaplnění heapu. U synchronních schránek tato operace nemá smysl.

deklarace

```
function Full(var MailBox: MailBox): Boolean;
```

parametr

parametr MailBox označuje schránku.

23.19. GetPriority procedure

Procedura vrátí hodnoty statické a dynamické priority běžícího procesu.

deklarace

```
procedure GetPriority(var StaticPriority: PriorityType;
                      var DynamicPriority: Byte);
```

parametry

Do proměnné StaticPriority je uložena statická priorita a proměnné DynamicPriority je uložena dynamická priorita běžícího procesu.

také

```
SetPriority, SetPriorityIdent
```

23.20. Identification function

Identifikace jádra pro registraci.

deklarace

```
function Identification : IdentType;
```

23.21. Init8087 procedure

Inicializaci koprocesoru je nutno zavolat na začátku procesu, který koprocesor využívá.

deklarace

```
procedure Init8087;
```

také

```
Save8087, Restore8087
```

příklad

Příklad ukazuje proces Proc8087, který při své činnosti může využívat koprocesor. Procedurou SetEntryProc je určena procedura, která se volá pokaždé, je-li procesu přidělován procesor, procedurou SetExitProc je určena procedura, která je volaná při odebrání procesoru. V daném procesu je tedy při přidělení procesoru

obnoven obsah koprocesoru, naopak při odejmutí procesoru je obsah koprocesoru uložen.

```
{$F+}
procedure Proc8087;{ proces využívající koprocesor }
{$F-}
begin
  Start('Proc8087',1024,100,0);Exit;{ start procesu}
  Init8087; { inicializace koprocesoru }
  SetEntryProc(Restore8087);{ při přidělení proces.}
  SetExitProc(Save8087); { při odebrání procesoru }
  ...
end;
```

23.22. InitCounterMailBox procedure

Inicializace asynchronní schránky tvořené čítačem prázdných zpráv. Inicializace musí být provedena před jakoukoliv operací se schránkou. Maximální počet zpráv je 65535.

deklarace

```
procedure InitCounterMailBox(var MailBox: MailBox;
                             Ident: IdentType; Mode: ModeType);
```

parametry

MailBox označuje schránku, která má být inicializována. Ident je identifikace schránky. Mode určuje režim schránky, normal nebo broadcasting. Je-li nastaven režim brodcasting bude všem procesům, čekajícím u schránky, předána přijatá zpráva. V režimu normal bude zpráva předána jen jednomu procesu.

23.23. InitDynMailBox procedure

Inicializace asynchronní dynamické schránky. Zprávy jsou uschovávány v položkách alokovaných na heapu. Inicializace musí být provedena před jakoukoliv operací se schránkou. Maximální počet zpráv je dán velikostí heapu.

deklarace

```
procedure InitDynMailBox(var MailBox: MailBox;
      Ident : IdentType;
      MessageLength: Word;
      Mode: ModeType);
```

parametry

MailBox označuje schránku, která má být inicializována. Ident je identifikace schránky. MessageLength je maximální délka zpráv. Mode určuje režim schránky, normal nebo broadcasting. Je-li nastaven režim brodcasting bude všem procesům, čekajícím u schránky, předána přijatá zpráva. V režimu normal bude zpráva předána jen jednomu procesu.

23.24. InitEmulátor procedure

Inicializace programového emulátoru matematického koprocesoru.

deklarace

```
procedure InitEmulator;
```

23.25. InitInterruptStack procedure

Procedura alokuje na heapu zásobník pro ošetření přerušení. Musí být volána před procedurami SetInterruptHandler a StartTimeSlicing.

deklarace

```
procedure InitInterruptStack(MaxInterruptCount: Byte;
                        MaxInterruptLength: Word);
```

parametry

MaxInterruptCount určuje počet uživatelských přerušení včetně časovače. MaxInterruptLength definuje maximální velikost zásobníku potřebnou pro zpracování jednoho přerušení.

23.26. InitSingleMailBox procedure

Inicializace schránky na jednu zprávu. Inicializace musí být provedena před jakoukoliv operací se schránkou.

deklarace

```
procedure InitSingleMailBox(var MailBox: MailBox; Ident: IdentType;
                  MessageLength: Word; Mode: ModeType);
```

parametry

MailBox označuje schránku, která má být inicializována. Ident je identifikace schránky. MessageLength je maximální délka zprávy. Mode určuje režim schránky, normal nebo broadcasting. Je-li nastaven režim brodcasting bude všem procesům, čekajícím u schránky, předána přijatá zpráva. V režimu normal bude zpráva předána jen jednomu procesu.

23.27. InitSingleOverMailBox procedure

Inicializace schránky, která uchovává pouze poslední zprávu. Inicializace musí být provedena před jakoukoliv operací se schránkou.

dekarace

```
procedure InitSingleOverMailBox(var MailBox: MailBox;
Ident : IdentType; MessageLength: Word; Mode: ModeType);
```

parametry

MailBox označuje schránku, která má být inicializována. Ident je identifikace schránky. MessageLength je maximální délka zprávy. Mode určuje režim schránky, normal nebo broadcasting. Je-li nastaven režim brodcasting bude všem procesům, čekajícím u schránky, předána přijatá zpráva. V režimu normal bude zpráva předána jen jednomu procesu.

23.28. InitStatMailBox procedure

Inicializace asynchronní schránky na pevný počet zpráv. Schránka je realizována jako kruhová fronta. Inicializace musí být provedena před jakoukoliv operací se schránkou. Parametry MessageLength a Count musí být oba nenulové.

deklarace

```
procedure InitStatMailBox(var MailBox: MailBox;
       Ident: IdentType; MessageLength: Word;
       Count: Word; Mode: ModeType);
```

parametry

MailBox označuje schránku, která má být inicializována. Ident je identifikace schránky. MessageLength je maximální délka zpráv. Count určuje maximální počet zpráv. Mode určuje režim schránky, normal nebo broadcasting. Je-li nastaven režim brodcasting bude všem procesům, čekajícím u schránky, předána přijatá zpráva. V režimu normal bude zpráva předána jen jednomu procesu.

23.29. InitSyncMailBox procedure

Inicializace synchronní schránky. Schránka neobsahuje žádný buffer, protože zprávy se přesouvají přímo mezi vysílajícím a přijímacím procesem. Inicializace musí být provedena před jakoukoliv operací se schránkou.

deklarace

```
procedure InitSyncMailBox(var MailBox: SyncMailBox;
            Ident: IdentType; Mode: ModeType);
```

parametry

MailBox označuje schránku, která má být inicializována. Ident je identifikace schránky. Mode určuje režim schránky, normal nebo broadcasting. Je-li nastaven režim brodcasting bude všem procesům, čekajícím u schránky, předána přijatá zpráva. V režimu normal bude zpráva předána jen jednomu procesu.

23.30. InitTraceFile procedure

Procedura pro inicializci trasovacího souboru.

deklarace

```
procedure InitTraceFile(TraceFile : String);
```

parametr

parametr TraceFile definuje jméno trasovacího souboru.

23.31. InitVarMailBox procedure

Inicializace asynchronní dynamické schránky s proměnnou délkou zpráv. Zprávy jsou uschovávány v položkách alokovaných na heapu. Inicializace musí být

provedena před jakoukoliv operací se schránkou. Maximální počet zpráv je dán velikostí heapu.

deklarace

```
procedure InitVarMailBox(var MailBox: MailBox;
        Ident: IdentType; Mode: ModeType);
```

parametry

MailBox označuje schránku, která má být inicializována. Ident je identifikace schránky. Mode určuje režim schránky, normal nebo broadcasting. Je-li nastaven režim brodcasting bude všem procesům, čekajícím u schránky, předána přijatá zpráva. V režimu normal bude zpráva předána jen jednomu procesu.

23.32. InstallHandler function

Funkce pro instalaci handleru výjimky. Výjimka je globální proměnná typu procedura. Ke každé výjimce může existovat libovolně mnoho handlerů, které se instalují funkcí InstallHandler. Každý handler pro danou výjimku musí být dlouze volaná procedura na nejvyšší úrovni, jejíž parametrová část odpovídá typu příslušné proměnné výjimka. Funkce InstallHandler uloží do této proměnné odpovídající handler a vrátí hodnotu false. Posledně aktivovaný handler se zavolá i s parametry pomocí proměnné výjimka. Handler se po svém ukončení vrátí do místa, ve kterém byl instalován, avšak vrátí hodnotu true. Instalace handlerů se musí explicitně zrušit.

deklarace

```
function InstallHandler(Handler: Pointer;
                        var Exception): Boolean;
```

parametry

Handler je adresou handleru. Exception je výjimka.

také

`DeleteHandlers`

23.33. InterrupEntry procedure

Procedura pro zahájení přerušení. Volá se v obslužné proceduře přerušení. Procedura přepne na přerušovací zásobník, obnoví DS a uloží všechny registry.

deklarace

```
procedure InterruptEntry;
```

také

`InterruptExit, FastInterruptExit`

23.34. InterrupExit procedure

Procedura pro ukončení přerušení. Volá se v obslužné proceduře přerušení. Procedura obnoví původní zásobník a obnoví všechny registry. Dále předá hodnoty v přerušovacích schránkách procesům, které na tyto hodnoty čekají. Pokud na dané přerušení čeká proces, je okamžitě zařazen do fronty Ready a jestliže má nejvyšší

prioritu, je okamžitě spuštěn. Použití procedury InterruptExit je možné u přerušení, které nepřicházejí v kratších intervalech než cca 750 mikrosekund (platí pro 16 MHz) .

deklarace

```
procedure InterruptExit;
```

také

```
InterruptEntry, FastInterruptExit
```

23.35. Lock procedure

Voláním procedury Lock začíná chráněná sekce. V této sekci proces získá volající nejvyšší statickou prioritu a proces Dummy získá prioritu o 1 nižší. Chráněná sekce musí být ukončena voláním procedury UnLock.

deklarace

```
procedure Lock;
```

také

```
UnLock, TestLock
```

23.36. LockKernel procedure

Voláním procedury LockKernel začíná chráněná sekce. V této sekci se neprovádí žádná manipulace s frontami Ready, Delay a dále žádné operace nad schránkami s výjimkou přerušovacích schránek určených voláním procedury SetInterruptHandler. Chráněná sekce musí být ukončena voláním procedury UnLockKernel nebo voláním jiné procedury jádra, která volá proceduru UnLockKernel. Tyto procedury jádra jsou označeny, že zamykají jádro.

deklarace

```
procedure LockKernel;
```

také

```
UnLockKernel, TestLockKernel
```

23.37. Match function

Funkce testuje, zda jméno procesu Name odpovídá masce Mask. V masce je možno použít znak '?', který zastupuje libovolný znak nebo znak '*', který zastupuje libovolnou posloupnost znaků.

deklarace

```
function Match(var Name, Mask: IdentType): Boolean;
```

23.38. MessageCount function

Funkce vrací počet zpráv ve schránce. U synchronních schránek to je počet procesů, které čekají na předání zprávy.

deklarace

```
function MessageCount(var MailBox: AbstractMailBox): Word;
```

parametr

MailBox označuje schránku.

také

```
Purge
```

23.39. MessageLength function

Funkce vrací délku následující zprávy ve schránce.

deklarace

```
function MessageLength(var MailBox: MailBox): Word;
```

parametr

MailBox označuje schránku.

23.40. NextMessage function

Funkce vrací adresu další zprávy ze schránky nebo hodnotu nil, jestliže další zpráva neexistuje.

deklarace

```
function NextMessage(var MailBox: MailBox): pointer;
```

parametr

MailBox označuje schránku.

23.41. Popf procedure

Načtení stavového slova procesoru do zásobníku.

deklarace

procedure **POPF**;

23.42. ProcCount function

Funkce vrací počet procesů čekajících u schránky na zprávu.

deklarace

```
function ProcCount( var MailBox: AbstractMailBox): Word;
```

parametr

MailBox označuje schránku.

23.43. ProcessCount function

Funkce navrací počet všech procesů v systému. Tento počet je roven součtu procesů ve frontě Ready a Delay. Do počtu je zařazen i hlavní proces.

deklarace

```
function ProcessCount: Word;
```

také

```
ReadyCount, DelayCount
```

23.44. Purge procedure

Procedura vymaže všechny zprávy ze schránky. Pokud je schránka synchronní, jsou spuštěny všechny procesy, které čekají, aby mohly odevzdat zprávu. Funkce SendTimeOut a SendEmptyTimeOut vrací v tomto případě hodnotu false.

deklarace

```
procedure Purge(var MailBox: AbstractMailBox);
```

parametr

MailBox označuje schránku, u které se vymažou všechny zprávy.

také

```
SendTimeOut, SendEmptyTimeOut
```

23.45. Pushf procedure

Uložení stavového slova procesoru do zásobníku.

deklarace

```
procedure PUSHF;
```

23.46. ReadyCount function

Funkce navrací počet procesů ve frontě Ready.

deklarace

```
function ReadyCount: Word;
```

také

```
ProcessCount, DelayCount
```

23.47. Receive function

Funkce Receive vrací identifikaci schránky, ve které je k dispozici zpráva. Funkce předpokládá předcházející volání procedur Choice resp. ChoiceEmpty, které určí schránky, na kterých se má zpráva očekávat. Činnost funkce Receive závisí na tom, zda je na některé z určených schránek zpráva k dispozici. Pokud je zpráva k dispozici, vrací funkce Receive hodnotu parametru ChoiceNo procedury Choice nebo ChoiceEmpty, na jejíž schránku byla zpráva přijata. Parametr ChoiceNo může být u

několika schránek stejný. Pokud není zpráva k dispozici, je proces, který provedl Receive, pozdržen ve frontě Delay. Před použitím schránky je třeba ji nejprve inicializovat a definovat její vlastnosti.

deklarace

```
function Receive: Word;
```

také

```
ReceiveTimeOut, ReceiveVar, ReceiveVarTimeOut, Choice,
ChoiceEmpty
```

příklad

Příklad ukazuje použití procedur Choice, ChoiceEmpty a funkce Receive. Při volání funkce Receive je proces do příchodu zprávy na některou ze schránek, označených procedurami Choice nebo ChoiceEmpty, pozdržen. Po příchodu zprávy je čekání ve funkci Receive ukončeno. Funkce Receive navrátí hodnotu uvedenou v parametru ChoiceNo při volání procedury Choice nebo ChoiceEmpty, která označila schránku, na kterou přišla zpráva. Před použitím schránky je třeba ji nejprve inicializovat a definovat její vlastnosti.

```
var
  MB1, MB2, MB3 : MailBox;
...
Choice(1,MB1,...);
ChoiceEmpty(2,MB2,...);
Choice(3,MB3,...);
case Receive of
1 : ... { zpráva přišla na schránku MB1 }
2 : ... { zpráva přišla na schránku MB2 }
3 : ... { zpráva přišla na schránku MB3 }
end;
```

23.48. ReceiveTimeOut function

Funkce ReceiveTimeOut vrací identifikaci schránky, ve které je k dispozici zpráva. Funkce předpokládá předcházející volání procedur Choice resp. ChoiceEmpty, které určí schránky, na kterých se má zpráva očekávat. Činnost funkce ReceiveTimeOut závisí na tom, zda je na některé z určených schránek zpráva k dispozici. Pokud je zpráva k dispozici, vrací funkce ReceiveTimeOut hodnotu parametru ChoiceNo procedury Choice nebo ChoiceEmpty, na jejíž schránku byla zpráva přijata. Pokud není zpráva k dispozici, je proces, který provedl ReceiveTimeOut, pozdržen ve frontě Delay. Na rozdíl od funkce Receive, je proces znovu spuštěn nejpozději po uplynutí doby dané parametrem TimeOut. V tom případě vrací funkce ReceiveTimeOut hodnotu 0. Před použitím schránky je třeba ji nejprve inicializovat a definovat její vlastnosti.

deklarace

```
function ReceiveTimeOut(TimeOut: TimeType): Word;
```

parametr

TimeOut určuje maximální dobu, po kterou funkce čeká na zprávu.

také

```
Receive, ReceiveVar, ReceiveVarTimeOut, Choice, ChoiceEmpty
```

23.49. ReceiveVar function

Funkce ReceiveVar vrací identifikaci schránky, ve které je k dispozici zpráva a skutečnou délku zprávy. Funkce předpokládá předcházející volání procedur Choice resp. ChoiceEmpty, které určí schránky, na kterých se má zpráva očekávat. Činnost funkce Receive závisí na tom, zda je na některé z určených schránek zpráva k dispozici. Pokud je zpráva k dispozici, vrací funkce ReceiveVar hodnotu parametru ChoiceNo procedury Choice nebo ChoiceEmpty, na jejíž schránku byla zpráva přijata. Pokud ne, je proces, který provedl Receive, pozdržen ve frontě Delay. Před použitím schránky je třeba ji nejprve inicializovat a definovat její vlastnosti.

deklarace

```
function ReceiveVar(var MessageLen: Word): Word;
```

parametr

MessageLen určuje skutečnou délku zprávy.

také

```
ReceiveTimeOut, ReceiveVar, ReceiveVarTimeOut, Choice, ChoiceEmpty
```

23.50. ReceiveVarTimeOut function

Funkce ReceiveVarTimeOut vrací identifikaci schránky, ve které je k dispozici zpráva, a skutečnou délku zprávy. Funkce předpokládá předcházející volání procedur Choice resp. ChoiceEmpty, které určí schránky, na kterých se má zpráva očekávat. Činnost funkce ReceiveTimeOut závisí na tom, zda je na některé z určených schránek zpráva k dispozici. Pokud je zpráva k dispozici, vrací funkce ReceiveVarTimOut hodnotu parametru ChoiceNo procedury Choice nebo ChoiceEmpty, na jejíž schránku byla zpráva přijata. Pokud ne, je proces, který provedl ReceiveTimeOut, pozdržen ve frontě Delay. Na rozdíl od funkce Receive, je proces znovu spuštěn nejpozději po uplynutí doby dané parametrem TimeOut. V tom případě vrací funkce ReceiveTimeOut hodnotu 0. Před použitím schránky je třeba ji nejprve inicializovat a definovat její vlastnosti.

deklarace

```
function ReceiveVarTimeOut(TimeOut: TimeType;
                     var MessageLen: Word): Word;
```

parametry

TimeOut určuje maximální dobu, po kterou funkce čeká na zprávu. V parametru MessageLen je navrácena skutečná délka přijaté zprávy.

také

```
Receive, ReceiveTimeOut, ReceiveVar, Choice, ChoiceEmpty
```

23.51. Restore8087 procedure

Obnova stavu koprocesoru, nutno zavolat před přidělením procesoru procesu, který využívá koprocesor.

deklarace

```
procedure Restore8087(var Buff: Buff8087);
```

parametr

Buff je proměnná kam se uloží registry koprocesoru.

také

```
Save8087, Init8087
```

23.52. RunningProcess function

Funkce vrátí jméno právě spuštěného procesu. Není-li jádro spuštěno, vrátí hodnotu 'No Kernel'.

deklarace

```
function  RunningProcess: IdentType;
```

23.53. Save8087 procedure

Uložení stavu koprocesoru, nutno zavolat před odebráním procesoru.

deklarace

```
procedure Save8087(var Buff: Buff8087);
```

parametr

Buff je proměnná kam se uloží registry koprocesoru.

také

```
Restore8087, Init8087
```

23.54. Send procedure

Procedura pro odeslání zprávy na schránku. Zpráva musí být připravena v bufferu, který se uvede jako parametr procedury Send. Schránka musí být inicializovaná. Pokud je schránka asynchronní MailBox, pokračuje proces okamžitě dále v činnosti. Pokud je schránka synchronní SyncMailBox, závisí další činnost procedury Send na tom, očekává-li na uvedené schránce nějaký proces zprávu. Pokud ano, je zpráva předána a oba procesy pokračují okamžitě v činnosti. Pokud žádný proces na zprávu nečeká, je proces, který zavolal proceduru Send, zařazen do fronty Delay a čeká než je zpráva ze schránky odebrána.

deklarace

```
procedure Send(var MailBox: AbstractMailBox;
          var MessageAddress; MessageLength: Word);
```

parametry

Parametr MailBox označuje schránku, na kterou má být zpráva odeslána. MessageAddress označuje adresu bufferu se zprávou. MessageLength určuje skutečnou délku zprávy.

také

`SendEmpty, SendTimeOut, SendEmptyTimeOut`

příklad

Příklad ukazuje zaslání zprávy procesem PSend a její přijetí procesem PReceive.

```
var
  MB : MailBox; { schránka }

{$F+}
procedure PSend; { proces zasílající zprávu }
{$F-}
var
  Mess : Word;
begin
  Start('PSend',1024,100,0);Exit;
  ...
  Send(MB,Mess,Size(Mess)); { zaslání zprávy }
  ...
end;


{$F+}
procedure PReceive; { proces přijímající zprávu }
{$F-}
var
  Mess : Word;
begin
  Start('PReceive',1024,100,0);Exit;
  ...
  Choice(1,MB,Mess,Size(Mess)); { určení schránky }
  case Receive of               { příjem zprávy }
  1 : ...              { přišla zpráva ze schránky MB }
  end;
  ...
end;


begin                          { hlavní proces }
  StartMain(200,0);            { inicializace jádra o.s.}
  InitInterruptStack(1,4000);{ definování zásobníku}
  StartTimeSlicing(8);         { určení system.časovače }
  ...
                               { inicializace schránky }
  InitDynMailBox(MB,'MB',SizeOf(Word),Normal);
  ...
  PReceive;                    { spuštění procesu }
  PSend;                       { spuštění procesu }
  ...
  SetPriority(10,0;)           { snížení priority }
  TerminateMain;               { ukončení jádra o.s. }
end.
```

23.55. SendEmpty procedure

Procedura pro odeslání prázdné zprávy na schránku. Schránka musí být inicializovaná a musí to být buď synchronní schránka SyncMailBox nebo to musí být asynchronní schránka čítač MailBox, inicializovaná procedurou InitCounterMailBox. Pokud je schránka asynchronní, pokračuje proces okamžitě dále v činnosti. Pokud je schránka synchronní SyncMailBox, závisí další činnost procedury SendEmpty na tom,

očekává-li na uvedené schránce nějaký proces zprávu. Pokud ano, je zpráva předána a oba procesy pokračují okamžitě v činnosti. Pokud žádný proces na zprávu nečeká, je proces, který zavolal proceduru SendEmpty, zařazen do fronty Delay a čeká než je zpráva ze schránky odebrána.

deklarace

```
procedure SendEmpty(var MailBox: AbstractMailBox);
```

parametry

Parametr MailBox označuje schránku, na kterou má být zpráva odeslána.

také

```
Send, SendTimeOut, SendEmptyTimeOut
```

23.56. SendTimeOut function

Funkce pro odeslání zprávy na schránku. Zpráva musí být připravena v bufferu, který se uvede jako parametr funkce SendTimeOut. Schránka musí být inicializovaná. Další činnost funkce SendTimeOut závisí na tom, očekává-li na uvedené schránce nějaký proces zprávu. Pokud ano, je zpráva předána, oba procesy pokračují okamžitě v činnosti a funkce SendTimeOut vrací hodnotu true. Pokud žádný proces na zprávu nečeká, je proces, který provedl SendTimeOut, zařazen do fronty Delay. Na rozdíl od procedury Send, je proces znovu spuštěn nejpozději po uplynutí doby dané parametrem TimeOut. V tom případě vrací funkce SendTimeOut hodnotu false.

deklarace

```
function SendTimeOut(var MailBox: AbstractMailBox;
      var Message, MessageLength: Word;
      TimeOut: TimeType) : Boolean;
```

parametry

Parametr MailBox označuje schránku, na kterou má být zpráva odeslána. MessageAddress označuje adresu bufferu se zprávou. MessageLength určuje skutečnou délku zprávy. TimeOut určuje maximální dobu, po kterou funkce čeká na převzetí zprávy.

také

```
Send, SendEmpty, SendEmptyTimeOut
```

23.57. SendEmptyTimeOut function

Funkce pro odeslání prázdné zprávy na schránku. Schránka musí být inicializovaná a musí to být synchronní schránka SyncMailBox. Další činnost funkce SendEmptyTimeOut závisí na tom, očekává-li na uvedené schránce nějaký proces zprávu. Pokud ano, je zpráva předána, oba procesy pokračují okamžitě v činnosti a funkce SendEmptyTimeOut vrací hodnotu true. Pokud žádný proces na zprávu nečeká, je proces, který provedl SendEmptyTimeOut, zařazen do fronty Delay. Na rozdíl od procedury SendEmpty, je proces znovu spuštěn nejpozději po uplynutí doby dané parametrem TimeOut. V tom případě vrací funkce SendEmptyTimeOut hodnotu false.

deklarace

```
function SendEmptyTimeOut(var MailBox : AbstractMailBox;
       TimeOut : TimeType) : Boolean;
```

parametry

Parametr MailBox označuje schránku, na kterou má být zpráva odeslána. TimeOut určuje maximální dobu, po kterou funkce čeká na převzetí zprávy.

také

```
Send, SendEmpty, SendTimeOut
```

23.58. SendInterrupt procedure

Procedura pro odeslání zprávy na schránku příslušející danému přerušení. Zpráva musí být připravena v bufferu, který se uvede jako parametr. Procedura se smí použít jen v přerušovací proceduře. Schránka musí být asynchronní inicializovaná procedurami InitStatMailBox nebo InitSingleMailBox a musí to být schránka přiřazená příslušnému přerušení procedurou SetInterruptHandler. Procedura zprávu uloží do schránky bez ohledu na to, jestli na zprávu čeká u schránky nějaký proces. Na přerušovací schránku nesmí žádný proces zasílat zprávy. Naopak může více procesů na schránce zprávu očekávat.

deklarace

```
procedure SendInterrupt(var MailBox: InterruptMailBox;
        var Message; MessageLength: Word);
```

parametry

Parametr MailBox označuje schránku, na kterou má být zpráva odeslána. MessageAddress označuje adresu bufferu se zprávou. MessageLength určuje skutečnou délku zprávy.

také

```
SendInterruptEmpty, Receive, ReceiveTimeOut, Accept,
AcceptTimeOut
```

23.59. SendInterruptEmpty procedure

Procedura pro odeslání prázdné zprávy na schránku příslušející danému přerušení. Procedura se smí použít jen v přerušovací proceduře. Schránka musí být asynchronní inicializovaná procedurou InitCounterMailBox a musí to být schránka přiřazená příslušnému přerušení procedurou SetInterruptHandler. Procedura zprávu uloží do schránky bez ohledu na to, jestli na zprávu čeká u schránky nějaký proces. Na přerušovací schránku nesmí žádný proces zasílat zprávy. Naopak může více procesů na schránce zprávu očekávat.

deklarace

```
procedure SendInterruptEmpty(var MailBox: InterruptMailBox);
```

parametry

Parametr MailBox označuje schránku, na kterou má být zpráva odeslána.

také

```
SendInterrupt, Receive, ReceiveTimeOut
```

23.60. SetEntryProc procedure

Procedura SetEntryProc určí proceduru, která je volána pokaždé, když je procesu přidělován procesor. Tato procedura nesmí mít žádné parametry a musí být typu FAR (direktiva {$F+}).

deklarace

```
procedure SetEntryProc(EntryProc: Pointer);
```

parametry

EntryProc je ukazatel na proceduru, která má být volána, je-li procesu přidělován procesor.

také

```
SetExitProc
```

23.61. SetExitProc procedure

Procedura SetExitProc určí proceduru, která je volána pokaždé, když je procesu odebírán procesor. Tato procedura nesmí mít žádné parametry a musí být typu FAR (direktiva {$F+}).

deklarace

```
procedure SetExitProc(ExitProc: Pointer);
```

parametry

ExitProc je ukazatel na proceduru, která má být volána, je-li procesu odebírán procesor.

také

```
SetEntryProc
```

23.62. SetInterruptHandler procedure

Procedura pro inicializaci vektoru přerušení. Obslužná procedura smí využívat pouze tu schránku, která je zadána jako parametr. Procedura se smí volat až po proceduře InitInterruptStack.

deklarace

```
procedure SetInterruptHandler(InterruptNo: byte;
            InterruptHandler: IntProc;
            var InterruptMailBox: InterruptMailBox);
```

parametry

InterruptHandler je obslužná procedura. InterruptNo určuje číslo přerušení. InterruptMailBox určuje přerušovací schránku.

také

```
SendInterrupt, SendInterruptEmpty
```

23.63. SetPriority procedure

Procedura nastaví běžícímu procesu nové hodnoty statické a dynamické priority. Statická priorita musí být nezáporné číslo. Procesor je přidělován procesu ve frontě Ready s nejvyšší statickou prioritou. Dynamická priorita má význam pouze v okamžiku, kdy má více procesů stejnou statickou prioritu. Dynamická priorita umožňuje spravedlivé rozdělování času procesoru mezi jednotlivé procesy. Za delší časové období je podíl času procesoru přiděleného jednotlivým procesům se stejnou statickou prioritou rovný podílu hodnot 255-dynamická priorita. Pokud nechceme využívat dynamickou prioritu, nastavíme její hodnotu na 255. V tomto případě bude procesor přidělován pouze prvnímu procesu ve frontě Ready s nejvyšší statickou prioritou.

deklarace

```
procedure SetPriority(StaticPriority: PriorityType;
                      DynamicPriority: Byte);
```

parametry

StaticPriority určuje novou statickou prioritu. Její hodnota musí být z intervalu 1 až 16383. DynamicPriority určuje novou dynamickou prioritu. Může být z intervalu 0 až 255. Je-li hodnota 255, není dynamická priorita využívána.

také

```
SetPriorityIdent, GetPriority, Start, StartMain
```

příklad

Příklad ukazuje hlavní proces, který spustí dvě instance procesu ProcesA, sníží svoji prioritu a umožní provádění procesu ProcesA.

```
{$F+}
procedure ProcesA(S: String);
{$F-}
begin
  Start('ProcesA',4000,100,254);Exit;
  ...
end;

...
StartMain(200,0);
InitInterruptStack(1,4000);
StartTimeSlicing(8);
...
ProcesA('1');
ProcesA('2');
...
SetPriority(10,0);
...
TerminateMain;
...
```

23.64. SetPriorityIdent procedure

Procedura nastaví nové hodnoty statické a dynamické priority všem procesům, které odpovídají zadané identifikaci. Procesy mohou být v libovolném stavu, ve frontě Ready nebo Delay. Priorita může být nastavena i procesu, který proceduru zavolal. Prioritu nelze nastavit procesu 'Dummy'.

deklarace

```
procedure SetPriorityIdent(Identification: IdentType;
       StaticPriority: PriorityType; DynamicPriority: Byte);
```

parametry

Identification určuje, kterých procesů se nastavení priorit bude týkat. Znak '?' zastupuje libovolný znak, '*' zastupuje libovolnou posloupnost znaků. StaticPriority určuje novou statickou prioritu. Její hodnota musí být z intervalu 1 až 16383. DynamicPriority určuje novou dynamickou prioritu. Musí být z intervalu 0 až 255.

také

```
SetPriority, GetPriority, Start, StartMain
```

23.65. SimulationTime function

Funkce vrací hodnotu tzv. simulačního času, tj. skutečného času procesoru, který byl procesu přidělen.

deklarace

```
function SimulationTime: Word;
```

23.66. Start procedure

Procedura odstartuje nový proces. Volá se na začátku procedury, která definuje proces. Tato procedura musí být definována na nejvyšší úrovni a musí být typu FAR. Za voláním této procedury musí bezprostředně následovat volání standardní procedury Exit. Procedura Start alokuje na heapu tabulku popisu nového procesu a jeho zásobník. Na zásobník přesune novému procesu hodnoty parametrů, se kterými byla procedura definující proces zavolána.

deklarace

```
procedure Start(Identification: IdentType; StackLength: Word;
          StaticPriority: PriorityType; DynamicPriority: Byte);
```

parametry

Identification označuje jméno vytvářeného procesu (max. 15 znaků). Jméno procesu může být libovolné, více procesů může mít stejné jméno. StackLength určuje velikost zásobníku vytvářeného procesu v bytech. StaticPriority určuje statickou prioritu vznikajícího procesu. Její hodnota musí být z intervalu 1 až 16383. DynamicPriority určuje dynamickou prioritu vznikajícího procesu. Její hodnota může být z intervalu 0 až 255. Více informací o prioritách procesů lze nalézt v popisu procedury SetPriority.

také

```
StartMain, TerminateMain, Terminate, SetPriority
```

příklad

Příklad ukazuje proces ProcesA a vytvoření několika jeho instancí.

```
{$F+}
procedure ProcesA(S: String);
{$F-}
begin
  Start('ProcesA',4000,100,254);Exit;
  ...
  if S='První instance.' then ...
  ...
end;

...
{ spuštění instancí procesu ProcesA }
ProcesA('První instance.'),
ProcesA('Druhé instance.'),
...
```

23.67. StartMain procedure

Procedura inicializuje jádro a spouští hlavní proces. Zásobník hlavního procesu je dán zásobníkem Turbo Pascalu. Implicitní identifikace hlavního procesu je 'Main'. Hlavní proces je zařazen do fronty Ready a jsou mu nastaveny priority podle uvedených parametrů. Tato procedura musí být volána před spuštěním všech ostatních procesů.

deklarace

```
procedure StartMain(StaticPriority: PriorityType;
                    DynamicPriority: Byte);
```

parametr

StaticPriority určuje statickou prioritu hlavního procesu. Její hodnota musí být z intervalu 1 až 16383. DynamicPriority určuje dynamickou prioritu hlavního procesu. Musí být z intervalu 0 až 255.

také

```
TerminateMain, Start, TerminateMain, SetPriority,
SetPriorityIdent
```

příklad

Ukázka hlavního procesu. Po inicializaci jádra je procedurou InitInterruptStack definován počet a velikost zásobníků pro přerušení. StartTimeSlicing(8) označuje INT 8 jako systémový časovač. Nakonec je procedurou TerminateMain ukončena činnost jádra operačního systému.

```
...
StartMain(200,0); { inicializace jádra }
InitInterruptStack(1,4000); { definování zásobníku }
StartTimeSlicing(8); { určení syst.časovače }
...
TerminateMain; { ukončení jádra o.s. }
...
```

23.68. StartTimeSlicing procedure

Procedura nastaví zadaný vektor přerušení jako časovač. Musí být volána po proceduře InitInterruptStack.

deklarace

```
procedure StartTimeSlicing(TimeInterruptNo: byte);
```

také

```
SetInterruptHandler
```

23.69. TestLock function

Funkce testuje, zda se proces nachází v chráněné sekci.

deklarace

```
function TestLock : Boolean;
```

23.70. TestLockKernel function

Funkce testuje, zda se proces nachází v chráněné sekci.

deklarace

```
function TestLockKernel : Boolean;
```

23.71. TestKernel function

Funkce testuje, zda je v činnosti jádro operačního systému ReTOS.

deklarace

```
function TestKernel: Boolean;
```

23.72. Terminate procedure

Procedura zruší proces, který ji zavolá. Proces je vyjmut ze všech front a je uvolněna paměť, která mu byla přidělena. Rušení jiných procesů se provádí procedurou Abort. Procedura Terminate se volá automaticky na konci procedury, která definuje proces. Hlavní proces se ruší procedurou TerminateMain.

deklarace

```
procedure Terminate;
```

také

```
Start, StartMain, TerminateMain, Abort
```

příklad

```
{$F+}
procedure ProcesA(S: String);
{$F-}
begin
  Start('ProcesA',4000,100,254);Exit;
  repeat
```

```
    ...
    if S='První instance.' then
    Terminate;
  until false
end;
```

23.73. TerminateMain procedure

Procedura může být zavolána pouze v hlavním procesu. Procedura pozastaví hlavní proces dokud existuje více než MinProcessNo procesů (s výjimku hlavního procesu a zahaleče). Implicitní hodnota proměnné MinProcessNo je 0. Pokud tuto hodnotu uživatel nezmění, čeká hlavní proces na dokončení všech ostatních procesů. Potom je hlavní proces vyjmut z fronty Ready a je uvolněna paměť, která mu byla rezervováma na heapu. Jsou obnoveny původní obsahy přerušovacích vektorů a je zavolána uživatelská ukončovací procedura určená proměnnou UserTerminate.

deklarace

```
procedure TerminateMain;
```

také

```
StartMain, Start, Terminate
```

23.74. UnLock procedure

Ukončení chráněné sekce, která byla zahájena procedurou Lock. Procedura UnLock obnoví procesu prioritu, kterou měl před voláním procedury Lock a procesu Dummy je nastavena nejnišší priorita.

deklarace

```
procedure UnLock;
```

také

```
Lock, TestLock
```

23.75. UnLockKernel procedure

Ukončení chráněné sekce, která byla zahájena procedurou LockKernel. Procedura UnLockKernel provede zpracování všech přerušení, která přišla během provádění chráněné sekce. Procesům čekajícím na přerušovacích schránkách jsou předány zprávy uložené v těchto schránkách a jsou provedeny všechny úpravy front Ready a Delay.

deklarace

```
procedure UnLockKernel;
```

také

```
LockKernel, TestLockKernel
```

23.76. Wait procedure

Procedura zpozdí o DeltaTime časových jednotek proces, který jí zavolal. Před provedením procedury musí být inicializován přerušovací zásobník a časová správa procedurami InitInterruptStack a StartTimeSlicing. Proces je vyjmut z fronty Ready a zařazen do fronty Delay. Trvání časové jednotky je určeno periodou systémového časovače a hodnotou ClockDivider. (na počítači PC při standardním nastavením je doba rovna cca 55 ms)

deklarace

```
procedure Wait(DeltaTime: TimeType);
```

parametr

DeltaTime určuje velikost zpoždění v časových jednotkách, po kterém se proces opět spustí.

také

```
WaitUntil
```

příklad

Ukázka předvádí vytvoření impulsu na výstupním portu 99 šířky 5 časových jednotek. (na počítači PC při standardním nastavení je trvání pulsu = 5 x 55 ms)

```
...
Port[99]:=1;
Wait(5);
Port[99]:=0;
...
```

23.77. WaitUntil procedure

Procedura pozastaví do zadaného absolutního systémového času proces, který jí zavolal. Před provedením procedury musí být inicializován přerušovací zásobník a časová správa procedurami InitInterruptStack a StartTimeSlicing. Proces je vyjmut z fronty Ready a zařazen do fronty Delay. Proces se spustí znovu v čase Time.

deklarace

```
procedure WaitUntil(var Time: TimeType);
```

parametr

Proměnná Time obsahuje hodnotu absolutního času, ve kterém se proces opět spustí. Hodnota parametru musí být větší než aktuální hodnota RealTime.

také

```
Wait
```

příklad

Příklad ukazuje vykonávání činnosti s přesnou periodou opakování. Proces MaxTime opravuje všechny proměnné obsahující údaj o absolutním čase.

```
var
  TimeA: TimeTipe;
  i: Integer;
```

```
{$F+}
procedure MaxTime;
{$F-}
begin
  Start('MaxTime',1024,16383,0);Exit;
  repeat
    WaitUntil(MaxRealTime);
    Dec(TimeA,MaxRealTime);
    ...
  until false;
end;

begin { hlavní proces }
  StartMain(200,0); { inicializace jádra }
  InitInterruptStack(1,4000);{ definování zásobníku}
  StartTimeSlicing(8); { určení syst.časovače}

  LockKernel;
  TimeA:=RealTime; { získání absolutního času }
  UnLockKernel;
  i:=1;
  repeat
    LockKernel; { počátek chráněné sekce }
    TimeA:=TimeA+18; { hodnota abs. času za 1 s }
    UnLockKernel; { konec chráněné sekce }
    WaitUntil(TimeA); { pozastavení do daného času}
    WriteLn('Uplynulo ',i,' sekund.');
    Inc(i);
  until false;
  TerminateMain; { ukončení jádra o.s. }
end.
```

23.78. _Full funkce

Funkce vrací True jestliže je schránka plná, tj. nevejde se do ní zpráva o velikosti MessageLen. U dynamických schránek to znamená zaplněni heapu.

deklarace

```
function _Full(var MailBox: MailBox;
                           MessageLen: Word): Boolean;
```

Parametry

MailBox označuje schránku, MessageLen je velikost zprávy, která se má uložit do schránky.