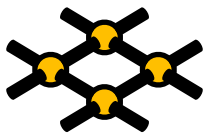
Vláknové programování
část XI

Lukáš Hejmánek, Petr Holub
{xhejtman,hopet}@ics.muni.cz



Laboratoř pokročilých síťových technologií

PV192
2010–05–06

Přehled přednášky

Ada

- Základy jazyka
- Paralelní programování

Erlang

Ada

"Regression testing?" What's that? If it compiles, it is good, if it boots up it is perfect.

– *Linus Torvalds*

Compilable but broken code is even worse than working code.

– *Alan Cox*
during a bright moment on the
linux-kernel list

Ada:
Ideally, when compiles fine it also runs fine.

Ada

- Ada 83/95/2005
- jazyk orientovaný na spolehlivé aplikace: vojáci, letectví, …
- WORM: write once read many
- silně typovaný jazyk
- podpora paralelismu
- enkapsulace, výjimky, objekty, správa paměti, atd.
- GNAT
  - volně dostupný překladač, frontend k GCC
- materiály na webu
  - (Annotated) Reference Manual
    `http://www.adaic.org/standards/ada05.html`
  - `http://stwww.weizmann.ac.il/g-cs/benari/books/index.html#ase`
  - `http://en.wikibooks.org/wiki/Programming:Ada`
  - `http://www.adahome.com/Tutorials/Lovelace/lovelace.htm`
  - `http://www.pegasoft.ca/resources/boblap/book.html`

Ada: Hello World!

```
1 with Ada.Text_IO;

3 procedure Hello is
  begin
5         Ada.Text_IO.Put_Line ("Hello, world!");
  end Hello;
```

- normální kompilace
  `gnatmake hello.adb`
- přidané kontroly
  `gnatmake -gnatya -gnatyb -gnatyc -gnatye -gnatyf -gnatyi -gnatyk -gnatyl -gnatyn -gnatyp -gnatyr -gnatyt -g -gnato -gnatf -fstack-check hello.adb`

Ada: typy

- Základní typy
  - celočíselné, plovoucí, decimální
  - výčty
  - pole
  - záznamy (struktury)
  - ukazatele (access)
  - úlohy (tasks)
  - rozhraní (interfaces)

```
Y : Boolean;
S : String;
F : Float;
Addr : System.Address;
type uhel is delta 1 / (60 * 60) range 0.0 .. 360.0;
type teplomer is delta 10.0**(-2) digits 10 range -273.15 .. 5000.0;
type pole is array (1 .. 10) of  Natural;
X : String (1 .. 10) := (others => ' ');
type Enum is (A, B, C);
E : Enum := A;
```

Ada: typy

- Anonymní vs. pojmenované typy

```
X : String (1 .. 10) := (others => ' ');
subtype My_String is String (1 .. 10);
Y : My_String := (others => ' ');
```

- Privátní versus limitované privátní typy

```
type typ_X is private; -- nevidime dovnitr
type typ_X is limited private; -- nelze priradit
```

Ada: typy

- Záznamy / Struktury

```
1  type Car is record
      Identity        : Long_Long_Integer;
3     Number_Wheels   : Positive range 1 .. 10;
      Paint           : Color;
5     Horse_Power_kW  : Float range 0.0 .. 2_000.0;
      Consumption     : Float range 0.0 .. 100.0;
7  end record;

9  BMW : Car :=
     (Identity        => 2007_752_83992434,
11    Number_Wheels   => 5,
      Horse_Power_kW  => 190.0,
13    Consumption     => 10.1,
      Paint           => Blue);
```

- Variantní záznamy, uniony

Ada: typy

- Objekty – tagované záznamy

```
1  type Person is tagged
      record
3        Name   : String (1 .. 10);
         Gender : Gender_Type;
5     end record;

7  type Programmer is new Person with
      record
9        Skilled_In : Language_List;
      end record;
11
   function Get_Skills (P : Programmer) return Language_List;
```

Ada: typy

- Typy vs. podtypy

```
type Integer_1 is range 1 .. 10;
subtype Integer_2 is Integer_1 range 7 .. 11;
A : Integer_1 := 8;
B : Integer_2 := A; -- OK

type Integer_1 is range 1 .. 10;
type Integer_2 is new Integer_1 range 2 .. 8;
A : Integer_1 := 8;
B : Integer_2 := A; -- nelze!
C : Integer_2 := Integer_2 (A); -- OK
```

Ada: typy

- Ukazatele

```
type Day_Of_Month is range 1 .. 31;
type Day_Of_Month_Access is access Day_Of_Month;
type Day_Of_Month_Access_All is access all Day_Of_Month;
for Day_Of_Month_Access'Storage_Pool use Pool_Name;
procedure Test (Call_Back: access procedure (Id: Integer; Text: String));

DoM : Day_Of_Month;
DoMA : Day_Of_Month_Access := Dom'Access; -- neee
DoMAA : Day_Of_Month_Access_All := Dom'Access; -- jo

DoMA : Day_Of_Month_Access := new Day_Of_Month;
procedure Free is new Ada.Unchecked_Deallocation
        (Object => Day_Of_Month
        Name   => Day_Of_Month_Access);
Free (DoMA);
```

Ada: typy

- Atributy

```
type barvy is (cervena, zelena, modra);
barvy'Pos(cervena) -- = 0
barvy'Val(0) -- = cervena
type pole is array (1 .. 10) of  Natural;
pole'First -- = 1
pole'Last -- = 10
pole'Range -- = 1 .. 10
type Byte is range -128 .. 127;
for Byte'Size use 8;
Byte'Min -- = -128
Byte'Max -- = 127
F : Float;
F'Ceiling;
F'Floor;
F'Rounding;
F'Image -- Float -> String
Float'Val -- String -> Float
```

Ada: řídící struktury

```
if condition then
        statement;
else
        other statement;
end if;

case X is
        when 1 =>
                Walk_The_Dog;
        when 5 =>
                Launch_Nuke;
        when 8 | 10 =>
                Sell_All_Stock;
        when others =>
                Self_Destruct;
end case;

Until_Loop :
loop
        X := Calculate_Something;
        exit Until_Loop when X > 5;
end loop Until_Loop;

for I in X'Range loop
        X (I) := Get_Next_Element;
end loop;
```

Ada: výjimky

```
package body Directory_Enquiries is

   procedure Insert (New_Name    : in Name;
                     New_Number  : in Number)
   is
   begin
      if New_Name = Old_Entry.A_Name then
         raise Name_Duplicated;
      end if;
      New_Entry :=  new Dir_Node'(New_Name, New_Number,);
   exception
      when Storage_Error => raise Directory_Full;
   end Insert;

   procedure Lookup (Given_Name   : in  Name;
                     Corr_Number  : out Number)
   is
   begin
      if not Found then
         raise Name_Absent;
      end if;
   end Lookup;

end Directory_Enquiries;
```

Ada: procedury, funkce

- procedury: **in**, **out** a **in out** parametry
- funkce: pouze **in** parametry, vrací hodnotu

```
1  procedure Procedura (A, B : in Integer := 0;
                        C : out Unbounded_String;
3                       D : in out Natural)
   is
5  begin
           C := To_Unbounded_String(A'Image & " " & B'Image);
7          D := D + 1;
   end Procedura;
9
   function Funkce (A, B: Integer) return String
11 is
           Retezec : Unbounded_String;
13         Prirozene_Cislo : Natural;
   begin
15         Procedura (A, B, Retezec, Prirozene_Cislo);
           Procedura (Retezec, Prirozene_Cislo);
17         Procedura (B => B, A => A, Retezec, Prirozene_Cislo);
           return To_String (Retezec);
19 end Funkce;

21 type Callback_Procedury is access procedure (Id  : Integer; T : String);
   type Callback_Function is access function (Id : Natural) return Natural;
```

Ada: balíky

- Princip zabalení
  - rozdělení funkcionality do balíků (packages)
  - hierarchie balíků: child packages (Ada95)
  - privátní část (balíků, chráněných typů)
- Princip oddělení specifikace a implementace
  - **`.ads`** popisuje rozhraní
    - lze kompilovat i jen proti specifikaci (bez implementace), ale nelze v takovém případě linkovat
  - **`.adb`** je implementace
- Doporučené pojmenování souborů podle jmen balíků
  - `s/\./-/g`
- Nejsou požadavky na adresářovou strukturu
- Možnost externalizace implementací balíků

Ada: balíky

balik.ads:

```
package Balik is

        type Muj_Typ is private;

        procedure Nastav_A (This : in out Muj_Typ;
                            An_A : in     Integer);

        function Dej_A (This : Muj_Typ) return Integer;

private

        type Muj_Typ is
        record
                A : Integer;
        end record ;

        pragma Inline (Dej_A);

end Balik;
```

Ada: balíky

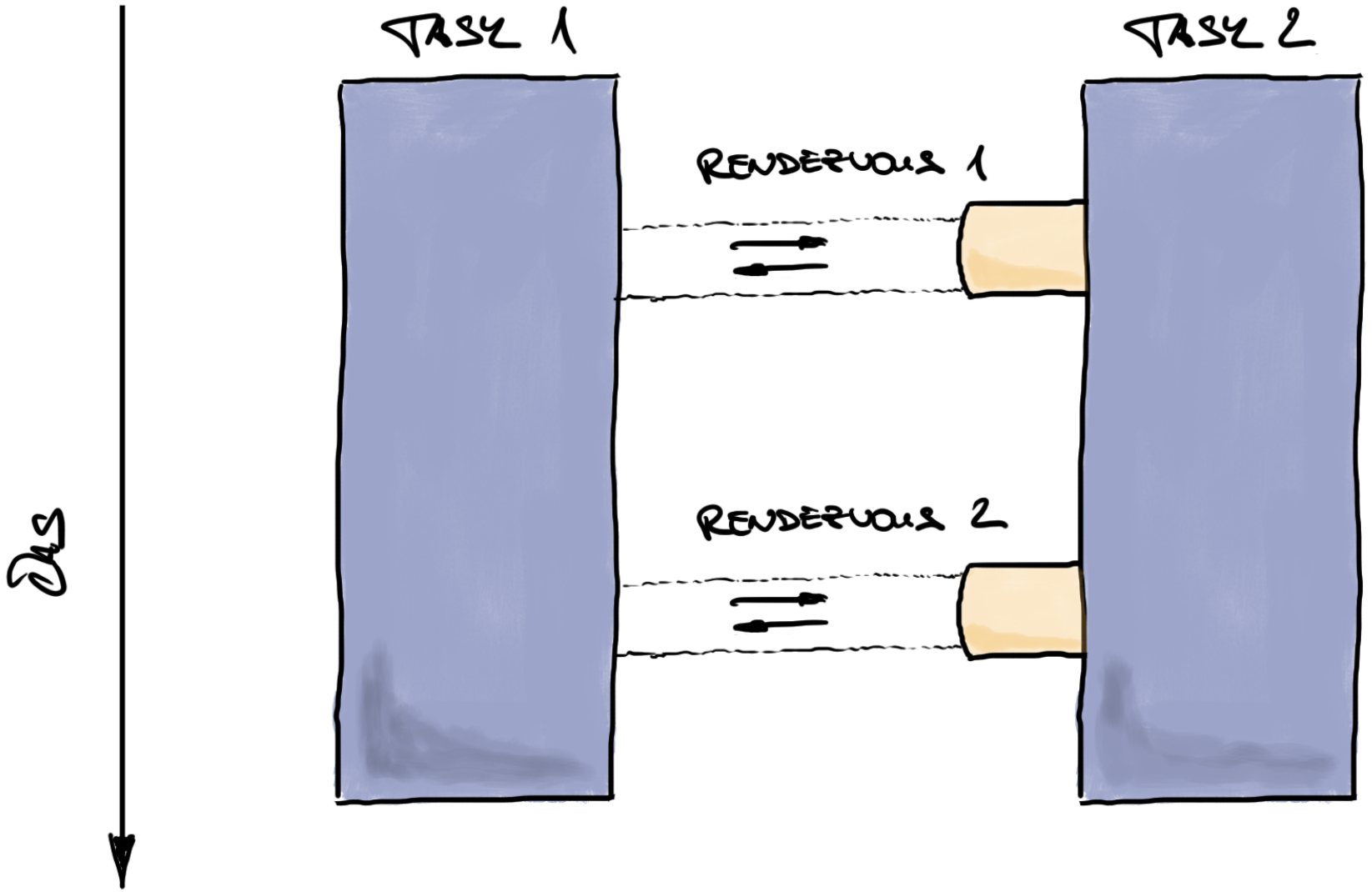
balik.adb:

```
1  package body Balik is
2
3     procedure Nastav_A (This : in out Muj_Typ;
4                         An_A : in     Integer)
5     is
6     begin
7        This.A := An_A;
8     end Set_A;
9
10    function Dej_A (This : Muj_Typ) return Integer is
11    begin
12       return This.A;
13    end Get_A;
14
15 end Balik;
```

Ada: Tasks, Rendezvous

- Koncept CSP: Communicating Sequential Processes
  - Hoare, 1978
  - paralelně běžící sekvenční procesy
  - komunikace: zasílání zpráv
  - synchronizace: synchronní zasílání zpráv
    - odesílatel se zablokuje, dokud příjemce není schopen přijmout zprávu
    - příjemce se zablokuje, dokud není schopen od odesílatele přijmout zprávu

Ada: Tasks, Rendezvous



Ada: Tasks

- **`task`**
  - lokálně definované
    - běží od začátku rozsahu, v němž jsou definované
  - dynamicky alokované
    - **`access`** typ
    - alokace pomocí **`new`**
    - běží až od alokace
  - pole tasků
- ukončování
  - spontánní
  - **`abort`**

Ada: Tasks

```
1  task T is
   end T;
3
   task body T is
5  begin
           makam;
7  end T;

9  task type T_Type is
   end T;
11
   task body T_Type is
13 begin
           loop
15                 makam;
           end loop;
17 end T_Type;

19 Pole_T : array (1..10) of T_Type;

21 type T_Type_Access is access T_Type;
   Dynamicky_T : T_Type_Access;
23 Dynamicky_T := new T_Type;
```

Ada: Rendezvous

- místa synchronizace – předávání dat
- **`entry`**
  - deklarace rendezvous bodu
  - **`in`**, **`out`**, **`in out`** parametry
- **`accept`**
  - implementace v těle tasku

Ada: Tasks, Rendezvous

```
1  procedure Task1 is
2
3          task Vlakno is
4                  entry ZadejX (X : in Integer);
5                  entry PrectiX (X : out Integer);
6          end Vlakno;
7
8          task body Vlakno is
9                  Hodnota : Integer;
10         begin
11                 accept ZadejX (X : in Integer) do
12                         Hodnota := X;
13                 end ZadejX;
14                 Hodnota := Hodnota + 1;
15                 accept PrectiX (X : out Integer) do
16                         X := Hodnota;
17                 end PrectiX;
18         end Vlakno;
19
20         Chci_Inkrementovat : Integer;
21
22 begin
23         Vlakno.ZadejX(Chci_Inkrementovat);
24         Vlakno.PrectiX(Chci_Inkrementovat);
25 end Task1;
```

Ada: Rendezvous

- **`select`**
  - výběr z více **`accept`** možností

```
1  task body T is
   begin
3          loop
                   select
5                          accept Rande1 do
                                   neco;
7                          end Rande1;
                   or
9                          accept Rande2 do
                                   neco;
11                         end Rande2;
                           neco;
13                         accept Rande3 do
                                   neco;
15                         end Rande3;
                   or
17                         terminate;
           end loop;
19 end T;
```

Ada: Rendezvous

- **`select`**
  - časovaný výběr

```
task body T is
begin
        loop
                select
                        accept Rande1 do
                                neco;
                        end Rande1;
                or
                        delay 10.0;
                        taky_neco;
                end select;
        end loop;
end T;
```

Ada: Rendezvous

- **`select`**
  - časovaný výběr

```
1  task body T is
   begin
3          loop
                   select
5                          accept Rande1 do
                                   neco;
7                          end Rande1;
                   else    --  ekvivalent "or delay 0.0"
9                          null;  --  busy waiting
                   end select;
11         end loop;
   end T;
```

Ada: pokročilé paralelní programování

- předávání parametrů při vzniku vlákna
  - parametrizace vlákna
- entry families
  - prioritizace a odlišení volajících vláken
- Real-Time and Distributed Systems Annex
  - (dynamické) priority vláken při volání entries
  - monotónní hodiny s vysokou přesností
  - restrikce tasků pro speciální případy (Ravenscar profile)
  - preemptivní **`abort`**
  - mnoho dalšího

Ada: atomické a volatilní proměnné

- **`pragma Atomic ();`**
  - zajišťuje atomické aktualizace proměnných

```
Prirozene_Cislo : Natural;
pragma Atomic (Prirozene_Cislo);
```

- **`pragma Atomic_Components ();`**
  - zajišťuje atomické aktualizace součástí složeného typu **`record`**

```
type Byte is range 0 .. 255;
for Byte'Size use 8;
type Moje_Struktura is
record
        B1 : Byte;
        B2 : Byte;
        B3 : Byte;
        B4 : Byte;
end record;
pragma Atomic_Components (Moje_Struktura);
```

Ada: atomické a volatilní proměnné

- **`pragma Volatile ();`**
  - upozornění pro kompilátor, že se hodnoty proměnných mohou neočekávaně měnit
  - zejména kompilátor musí zamezit optimalizacím, které by mohly interferovat

```
Buffer_Zarizeni : Integer;
pragma Volatile (Buffer_Zarizeni);
```

- **`pragma Volatile_Components ();`**
  - totéž pro komponenty složeného typu **`record`**

Ada: Protected Types – monitory

- Implementace monitorů
  - funkce – nemohou měnit data
    procedury – mohou měnit data
    entry – strážený vstup, mohou měnit data
  - efektivní paralelizace: podobne **`ReadWriteLock`**u v Javě
    - funkce mohou přistupovat paralelně
    - procedury a entries musí pracovat exkluzivně

```
protected type Muj_Typ is
        procedure Nastav_hodnotu (n : Integer);
        procedure Odnastav_hodnotu;
        function Zjisti_hodnotu return Integer;
        entry Pockej_na_nastaveni (n : Integer);
private
        Hodnota : Integer;
        Nastaveno : Boolean := False;
end Muj_Typ;
```

Ada: Protected Types – monitory

```
10 protected body Muj_Typ is
        procedure Nastav_hodnotu (n : Integer) is
12      begin
                Hodnota := n;
14              Nastaveno := True;
        end Nastav_hodnotu;
16
        procedure Odnastav_hodnotu is
18      begin
                Nastaveno := False;
20      end Odnastav_hodnotu;

22      function Zjisti_hodnotu return Integer is
        begin
24              return Hodnota;
        end Zjisti_hodnotu;
26
        entry Pockej_na_nastaveni
28      when Nastaveno is
        begin
30              null;
        end Pockej_na_nastaveni;
32 end Muj_Typ;
```

Ada: Guarded Entries

- chránění dle privátního stavu

```
protected type Chraneno_Stavem is
        entry Vstup;
private
        I : Integer;
end Chraneno_Stavem;

protected body Chraneno_Stavem is
        entry Vstup when I > 0 is
        begin
                null;
        end Vstup;
end Chraneno_Stavem;
```

  - používat pouze privátní proměnné
  - např. implementace mutexů a semaforů
  - **`requeue`**
    - zařazení volání privátní entry
    - volání sebe sama
    - vnitřní vs. vnější kruh entry

Ada: Guarded Entries

- chránění dle atributů

```
1  protected type Chraneno_Stavem is
           entry Vstup;
3  private
           I : Integer;
5  end Chraneno_Stavem;

7  protected body Chraneno_Stavem is
           entry Vstup when Vstup'Count > 4 is
9          begin
                   null;
11         end Vstup;
   end Chraneno_Stavem;
```

  - atribut **`E'Count`** vrátí počet zablokovaných vláken na vstupu do **`entry E`**
  - např. implementace bariér

Ada: Asynchronous Transfer of Control

- Časově omezený běh

```
select
        --  triggering_statement
        delay 5.0;
        --  post_trigger_part
        Put_Line ("Tudy cesta nevede!");
then abort
        --  abortable_part
        Prevelevelmidlouhe_Volani;
end select;
```

  - pokud **abortable_part** doběhne dříve než **triggering_statement**, pokusí se ukončit **triggering_statement**
  - pokud **triggering_alternative** doběhne dřív než **abortable_part**, je **abortable_part** ukončena a provede se část **post_trigger_part**
  - **triggering_statement** – v Ada 95 entry, v Ada 2005 i procedury

Erlang: distribuované programování

- Erlang
  - funkcionální programovací jazyk pro paralelní a distribuované programování
  - An Erlang Course
    http://www.erlang.org/course/course.html
  - **http://www.erlang.org/doc/reference_manual/processes.html**
  - skutečně použitelný: např. ejabberd
- vytvoření procesu

```
spawn(Modul, Exportovana_fce, Seznam_argumentu)
```

- předávání zpráv

```
PID_procesu ! zprava
receive zprava -> udelej_neco
```

- registrace procesů

```
register(nejaky_atom, PID)
whereis(registrovane_jmeno)
```

Erlang: distribuované programování

```
-module (priklad).
-compile(export_all).

klient(Pid) ->
        Pid ! {self(),pozadavek,ping},
        receive
                {Pid,odpoved,Odpoved} ->
                        io:format("dorazila odpoved ~p~n",[Odpoved]),
                        Pid ! exit
        end,
        io:format("klient skoncil~n").

server() ->
        receive
                {Od,pozadavek,Pozadavek} ->
                        io:format("obdrzel jsem pozadavek ~p od ~p~n",
                                [Pozadavek, Od]),
                        sleep(1000),
                        Od ! {self(),odpoved,pong},
                        server();
                _ ->
                        io:format("server skoncil~n")
        end.
```

Erlang: distribuované programování

```
sleep(Time) ->
        receive
        after Time -> void
  end.

spust() ->
        Pid = spawn(fun server/0),
        spawn(fun() -> klient(Pid) end),
        io:format("spusteni server i klient~n", []).
```

```
-bash-2.05b$ erl
Erlang (BEAM) emulator version 5.5.2 [source] [async-threads:0] [hipe]

Eshell V5.5.2  (abort with ^G)
1> c(priklad).
{ok,priklad}
2> priklad:spust().
spusteni server i klient
obdrzel jsem pozadavek ping od <0.37.0>
ok
dorazila odpoved pong
klient skoncil
server skoncil
```

Erlang: distribuované programování

- Svázené (linked) procesy

```
link(Pid) -> true
spawn_link(Modul, Exportovana_fce, Seznam_argumentu)
unlink(Id) -> true
```

  - pokud umře proces, pošle o tom zprávu všem, kteří jej mají „nalinkovaný“

```
{'EXIT', FromPid, Reason}
```

- Monitorování procesů

```
erlang:monitor(process, registrovane_jmeno)
```

  - alternativa k použití linků
  - při ukončení procesu zašle zprávu

```
{'DOWN', Ref, process, Pid2, Reason}
```

  - vícenásobné volání vytvoří více monitorů

- Zachytávání Exit signálů

```
process_flag(trap_exit,true)
```

  - při exitu vygeneruje zprávu jako u linků