Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

KATEDRA INFORMATYKI STOSOWANEJ

PRACA MAGISTERSKA

RAFAŁ STUDNICKI

PODSTAWOWA FUNKCJONALNOŚĆ ERLANGA DLA SYSTEMU FREERTOS

PROMOTOR:
dr inż. Piotr Matyasik

Kraków 2014

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE, I NIE KORZYSTAŁEM ZE ŹRÓDEŁ INNYCH NIŻ WYMIENIONE W PRACY.

. . . . . .

PODPIS

AGH

University of Science and Technology in Krakow

Faculty of Electrical Engineering, Automatics, Computer Science and Engineering in Biomedicine

DEPARTMENT OF APPLIED COMPUTER SCIENCE

MASTER OF SCIENCE THESIS

RAFAŁ STUDNICKI

IMPLEMENTATION OF BASIC FEATURES OF ERLANG FOR FREERTOS

SUPERVISOR:
Piotr Matyasik Ph.D

Krakow 2014

Spis treści

A. Lista operacji maszyny wirtualnej BEAM

Dodatek zawiera listę operacji maszyny wirtualnej BEAM, jakie może zawierać skompilowany kod pośredni przez nią wykonywany. Lista zawiera nazwę operacji, jej argumenty oraz opis jej działania.

Kod operacji oraz każdy argument zajmują zawsze 1 bajt w pliku skompilowanego kodu pośredniego. Kolejość bajtów w zapisie kodu pośredniego to *big endian*.

Kod operacji		**Nazwa operacji i jej argumenty**	**Opis operacji**
szesnastkowo	**dziesiętnie**		
01	1	label Lbl	Wprowadza lokalną dla danego modułu etykietę identyfikującą aktualne miejsce w kodzie.
02	2	func_info M F A	Definiuje funkcję F, w module M o arności A.
03	3	int_code_end	???

Tablica A.1: Lista operacji maszyny wirtualnej BEAM

B. Kompilacja kodu źródłowego

Podrozdział opisuje kolejne kroki, z jakich składa się proces otrzymywania skompilowanego kodu pośredniego maszyny wirtualnej BEAM z kodu źródłowego napisanego w języku Erlang.

B.1. Wprowadzenie

Jak zostało wspomniane w **??**, program napisany w języku Erlang wykonywany jest na dedykowanej do tego celu maszynie wirtualnej.

Narzędzia przeznaczone do operacji opisanych w niniejszym rozdziale zostały napisane w języku Erlang i dostępne są w aplikacji **compiler** dostarczanej wraz z maszyną wirtualną BEAM.

B.2. Kod źródłowy

```
1  -module(fac).
2
3  -export([fac/1]).
4  -define(ERROR, "Invalid argument").
5
6  -include("fac.hrl").
7
8  fac(#factorial{n=0, acc=Acc}) ->
9      Acc;
10 fac(#factorial{n=N, acc=Acc}) ->
11     fac(#factorial{n=N-1, acc=N*Acc});
12 fac(N) when is_integer(N) ->
13     fac(#factorial{n=N});
14 fac(N) when is_binary(N) ->
15     fac(binary_to_integer(N));
16 fac(_) ->
17     {error, ?ERROR}.
```

Listing B.1: Plik fac.erl

```
1  -record(factorial, {n, acc=1}).
```

Listing B.2: Plik fac.hrl

B.3. Preprocessing

```
1  -file("fac.erl", 1).
2
3  -module(fac).
4
5  -export([fac/1]).
6
7  -file("fac.hrl", 1).
8
9  -record(factorial,{n,acc = 1}).
10
11 -file("fac.erl", 7).
12
13 fac(#factorial{n = 0,acc = Acc}) ->
14     Acc;
15 fac(#factorial{n = N,acc = Acc}) ->
16     fac(#factorial{n = N - 1,acc = N * Acc});
17 fac(N) when is_integer(N) ->
18     fac(#factorial{n = N});
19 fac(N) when is_binary(N) ->
20     fac(binary_to_integer(N));
21 fac(_) ->
22     {error,"Invalid argument"}.
```

Listing B.3: Moduł fac po pierwszym przetworzeniu

```
1  -file("fac.erl", 1).
2
3  -file("fac.hrl", 1).
4
5  -file("fac.erl", 7).
6
7  fac({factorial,0,Acc}) ->
8      Acc;
9  fac({factorial,N,Acc}) ->
10     fac({factorial,N - 1,N * Acc});
11 fac(N) when is_integer(N) ->
12     fac({factorial,N,1});
13 fac(N) when is_binary(N) ->
14     fac(binary_to_integer(N));
15 fac(_) ->
16     {error,"Invalid argument"}.
17
18 module_info() ->
19     erlang:get_module_info(fac).
20
21 module_info(X) ->
```

```
22      erlang:get_module_info(fac, X).
```

Listing B.4: Moduł fac po drugim przetworzeniu

B.4. Transformacje drzewa syntaktycznego

```
1  [{attribute,1,file,{"fac.erl",1}},
2   {attribute,1,module,fac},
3   {attribute,5,export,[{fac,1}]},
4   {attribute,1,file,{"fac.hrl",1}},
5   {attribute,1,record,
6       {factorial,
7           [{record_field,1,{atom,1,n}},
8            {record_field,1,{atom,1,acc},{integer,1,1}}]}},
9   {attribute,9,file,{"fac.erl",9}},
10  {function,10,fac,1,
11      [{clause,10,
12           [{record,10,factorial,
13                [{record_field,10,{atom,10,n},{integer,10,0}},
14                 {record_field,10,{atom,10,acc},{var,10,'Acc'}}]}],
15           [],
16           [{var,11,'Acc'}]},
17       {clause,12,
18           [{record,12,factorial,
19                [{record_field,12,{atom,12,n},{var,12,'N'}},
20                 {record_field,12,{atom,12,acc},{var,12,'Acc'}}]}],
21           [],
22           [{call,13,
23                {atom,13,fac},
24                [{record,13,factorial,
25                     [{record_field,13,
26                          {atom,13,n},
27                          {op,13,'-',{var,13,'N'},{integer,13,1}}},
28                      {record_field,13,
29                          {atom,13,acc},
30                          {op,13,'*',{var,13,'N'},{var,13,'Acc'}}}]}]}]},
31       {clause,14,
32           [{var,14,'N'}],
33           [[{call,14,{atom,14,is_integer},[{var,14,'N'}]}]],
34           [{call,15,
35                {atom,15,fac},
36                [{record,15,factorial,
37                     [{record_field,15,{atom,15,n},{var,15,'N'}}]}]}]},
38       {clause,16,
39           [{var,16,'N'}],
40           [[{call,16,{atom,16,is_binary},[{var,16,'N'}]}]],
41           [{call,17,
42                {atom,17,fac},
```

```
43                [{call,17,{atom,17,binary_to_integer},[{var,17,'N'}]}]}]},
44       {clause,18,
45           [{var,18,'_'}],
46           [],
47           [{tuple,19,[{atom,19,error},{string,19,"Invalid argument"}]}]}]},
48  {eof,20}]
```

Listing B.5: Drzewo syntaktyczne modułu fac

B.5. Kod pośredni (*bytecode*)

```
1  {module, fac}.  %% version = 0
2
3  {exports, [{fac,1},{module_info,0},{module_info,1}]}.
4
5  {attributes, []}.
6
7  {labels, 11}.
8
9
10 {function, fac, 1, 2}.
11   {label,1}.
12     {line,[{location,"fac.erl",8}]}.
13     {func_info,{atom,fac},{atom,fac},1}.
14   {label,2}.
15     {test,is_tuple,{f,4},[{x,0}]}.
16     {test,test_arity,{f,4},[{x,0},3]}.
17     {get_tuple_element,{x,0},0,{x,1}}.
18     {get_tuple_element,{x,0},1,{x,2}}.
19     {get_tuple_element,{x,0},2,{x,3}}.
20     {test,is_eq_exact,{f,4},[{x,1},{atom,factorial}]}.
21     {test,is_eq_exact,{f,3},[{x,2},{integer,0}]}.
22     {move,{x,3},{x,0}}.
23     return.
24   {label,3}.
25     {line,[{location,"fac.erl",11}]}.
26     {gc_bif,'-',{f,0},4,[{x,2},{integer,1}],{x,0}}.
27     {line,[{location,"fac.erl",11}]}.
28     {gc_bif,'*',{f,0},4,[{x,2},{x,3}],{x,1}}.
29     {test_heap,4,4}.
30     {put_tuple,3,{x,2}}.
31     {put,{atom,factorial}}.
32     {put,{x,0}}.
33     {put,{x,1}}.
34     {move,{x,2},{x,0}}.
35     {call_only,1,{f,2}}.
36   {label,4}.
37     {test,is_integer,{f,5},[{x,0}]}.
```

```
38      {test_heap,4,1}.
39      {put_tuple,3,{x,1}}.
40      {put,{atom,factorial}}.
41      {put,{x,0}}.
42      {put,{integer,1}}.
43      {move,{x,1},{x,0}}.
44      {call_only,1,{f,2}}.
45    {label,5}.
46      {test,is_binary,{f,6},[{x,0}]}.
47      {allocate,0,1}.
48      {line,[{location,"fac.erl",15}]}.
49      {call_ext,1,{extfunc,erlang,binary_to_integer,1}}.
50      {call_last,1,{f,2},0}.
51    {label,6}.
52      {move,{literal,{error,"Invalid argument"}},{x,0}}.
53      return.
54
55
56  {function, module_info, 0, 8}.
57    {label,7}.
58      {line,[]}.
59      {func_info,{atom,fac},{atom,module_info},0}.
60    {label,8}.
61      {move,{atom,fac},{x,0}}.
62      {line,[]}.
63      {call_ext_only,1,{extfunc,erlang,get_module_info,1}}.
64
65
66  {function, module_info, 1, 10}.
67    {label,9}.
68      {line,[]}.
69      {func_info,{atom,fac},{atom,module_info},1}.
70    {label,10}.
71      {move,{x,0},{x,1}}.
72      {move,{atom,fac},{x,0}}.
73      {line,[]}.
74      {call_ext_only,2,{extfunc,erlang,get_module_info,2}}.
```

Listing B.6: *Bytecode* modułu fac

B.6. Plik binarny BEAM

Efektem przetworzenia kodu pośredniego, wyrażonego w postaci krotek, jest plik binarny w formacie IFF [6], w formacie zrozumiałym przez maszynę wirtualną BEAM. Maszyna ta wykorzystuje tego rodzaju pliki do ładowania kodu modułów do pamięci. Ich źródłem może być zarówno system plików na fizycznej maszynie, na której uruchomiony został BEAM, jak i inna maszyna wirtualna znajdująca się w tym samym klastrze *Distributed Erlang*, co docelowa.

W tabeli B.1 zaprezentowana została struktura pliku binarnego ze skompilowanym modułem.

	Oktet	**0**								**1**							
Oktet	**Bit**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	**0**	"FOR1"															
4	**32**	Rozmiar pliku bez pierwszych 8 bajtów															
8	**64**	"BEAM"															
12	**96**	Identyfikator fragmentu (*chunk*) 1															
16	**128**	Rozmiar fragmentu 1															
20	**160**	Dane fragmentu 1															
...	**...**	Identyfikator fragmentu (*chunk*) 2															
...	**...**	...															

Tablica B.1: Struktura pliku kodu pośredniego BEAM

Każdy plik binarny BEAM powinien zawierać przynajmniej 5 z następujących fragmentów (*chunków*). Obok opisu każdego fragmentu, w nawiasie podano ciąg znaków będący jego identyfikatorem w binarnym pliku modułu:

– tablica atomów wykorzystywanych przez moduł (`Atom`);

– bajtkod danego modułu (`Code`);

– tablica zewnętrznych funkcji używanych przez moduł (`ImpT`);

– tablica funkcji eksportowanych przez moduł (`ExpT`).

Ponadto, w pliku mogą znajdować się następujące, opcjonalne fragmenty:

– tablica funkcji lokalnych dla danego modułu (`LocT`);

– tablica lambd wykorzystwanych przed moduł (`FunT`);

– tablica stałych wykorzystywanych przed moduł (`LitT`);

– lista atrybutów modułu (`Attr`);

– lista dodatkowych informacji o kompilacji modułu (`CInf`);

– tablica linii kodu źródłowego modułu (używana przy debuggowaniu i generacji stosu wywołań - *stacktrace*) (`Line`);

– drzewo syntaktyczne pliku z kodem źródłowym (`Abst`).

W przypadku każdego rodzaju fragmentu, obszar jaki zajmuje on w pliku jest zawsze wielokrotnością 4 bajtów. Nawet jeżeli nagłówek fragmentu, zawierający jego rozmiar, nie jest podzielny przez 4, obszar zaraz za danym fragmentem dopełniany jest zerami do pełnych 4 bajtów.

Warto zaznaczyć również, że sposób implementacji maszyny wirtualnej BEAM nie definiuje kolejności w jakiej poszczególne fragmenty powinny występować w pliku binarnym.

B.6.1. Tablica atomów

Tablica atomów zawiera listę wszystkich atomów, które używane są przez dany moduł. W trakcie ładowania kodu modułu przez maszynę wirtualną, atomy, które nie wystepowały we wcześniej załadowanych modułach, zostają wstawione do globalnej tablicy atomów (w postaci tablicy z hashowaniem).

Fragment piku binarnego z tablicą atomów reprezentowany jest przez napis `Atom`. Struktura danych fragmentu zaprezentowana jest w tabeli B.2.

	Oktet	**0**								**1**							
Oktet	**Bit**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	**0**	Ilość atomów w tablicy atomów															
4	**32**	Dł. atomu 1								Nazwa atomu 1 w ASCII							
...	**...**	Dł. atomu 2								Nazwa atomu 2 w ASCII							
...	**...**	...															

Tablica B.2: Struktura tablicy atomów w pliku BEAM

B.6.2. Bajtkod

Sekcja z bajtkodem zawiera faktyczny kod wykonywalny modułu, który jest interpretowany przez maszynę wirtualną w trakcie uruchomienia systemu.

Fragment pliku z kodem identyfikowana jest przez napis `Code`. Struktura danych fragmentu zawarta została w tabeli B.3. Szczegółowy opis reprezentacji i znaczenia opkodów i ich argumentów zawarty został w dodatku A.

	Oktet	**0**								**1**							
Oktet	**Bit**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	**0**	0x000010															
4	**32**	Numer wersji formatu bajtkodu (w Erlangu R16 - 0x00000000)															
8	**64**	Maksymalny numer operacji (do sprawdzenia kompatybilności)															

12	**96**	Liczba etykiet w kodzie modułu	
16	**128**	Liczba funkcji eksportowanych z modułu	
20	**160**	Opkod 1	Argument 1
...	**...**	...	Argument N
...	**...**	Opkod 2	Argument 1
...	**...**	...	

Tablica B.3: Struktura bajtkodu w pliku BEAM

B.6.3. Tablica importowanych funkcji

Fragment pliku binarnego z tablicą importowanych funkcji zawiera informacje o funkcjach zaimplementowanych w innych modułach, które są wykorzystywane przez moduł.

Fragment pliku z kodem identyfikowana jest przez napis `ImpT`. Struktura danych fragmentu zawarta została w tabeli B.4.

	Oktet	**0**								**1**							
Oktet	**Bit**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	**0**	Liczba importowanych funkcji															
4	**32**	Indeks atomu z nazwą modułu 1															
8	**64**	Indeks atomu z nazwą funkcji 1															
12	**96**	Arność funkcji 1															
16	**128**	Indeks atomu z nazwą modułu 2															

...	**...**	...

Tablica B.4: Struktura tablicy importowanych funkcji w pliku BEAM

B.6.4. Tablica eksportowanych funkcji

Fragment pliku binarnego z tablicą eksportowanych funkcji zawiera informacje o funkcjach z modułu, które widoczne są z poziomu innych modułów.

Fragment pliku z kodem identyfikowana jest przez napis `ExpT`. Struktura danych fragmentu zawarta została w tabeli B.5.

	Oktet	**0**								**1**							
Oktet	**Bit**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	**0**	Liczba eksportowanych funkcji															
4	**32**	Indeks atomu z nazwą funkcji 1															
8	**64**	Arność funkcji 1															
12	**96**	Etykieta początku kodu funkcji 1															
16	**128**	Indeks atomu z nazwą funkcji 2															
...	**...**	...															

Tablica B.5: Struktura tablicy eksportowanych funkcji w pliku BEAM

B.7. Podsumowanie

Bibliografia

[1] Ericsson AB. Erlang Embedded Systems User's Guide, 1997.

[2] Richard Barry. *Using the FreeRTOS Real Time Kernel*. Real Time Engineers Ltd., 2011.

[3] Jim Gray. Why Do Computers Stop And What Can Be Done About It?, 1985.

[4] Maxim Kharchenko. Erlang on Xen, A quest to lower startup latency. *Erlang Factory SF Bay Area 2012, San Francisco*, 2012.

[5] Erlang Solutions Ltd. Erlang Embedded. `http://www.erlang-embedded.com`, 2013. [data dostępu: 17.03.2014].

[6] J. Morrison. EA IFF 85: Standard for interchange format files. *Amiga ROM Kernel Reference Manual: Devices (3rd edition), Addison-Wesley*, 1(99):1, 1985.

[7] Peer Stritzinger. Full Metal Erlang. *Erlang User Conference 2013, Stockholm*, 2013.