ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по “Изчислителна Техника и Автоматизация”
Катедра “Софтуерни и интернет технологии”

Курсов проект по дисциплина: “Учебна практика – 3 част”

Вариант №1

Изготвил: Тодор Иванов Йорданов, Ф.№: 20621600, група: 4а, курс: IV, спец: СИТ

Варна 2024г.

Съдържание

1. Задание

1. Създайте произволна структура от Erlang процеси по свой избор (нишка, пръстен, дърво или друго).

2. Създайте алгоритъм за обхождане на горната структура, при който всеки един от процесите извършва някаква полезна работа и агрегира резултата произведен от предишния процес.

3. Реализирайте процесите във вариант – Client-Server.

4. Създайте процес, който наблюдава състоянието на процесите от горната структура и при exit на даден процес, стартира процес от същия тип.

2. Анализ на програмата

Програмата съдържа 5 функции. Всяка функция изпълнява различна програмна логика, спрямо точките от заданието. Цялостната концепция по реализацията на курсовия проект е единствено и само по моя собствена идея и усмотрение. Програмата реализира структура от свързани erlang процеси, от тип “N-ary Tree” или т.нар. “Generic Tree”. Целта на изпълнението на програмата, е всяко едно от листата на дървото да генерира случайно цяло число между 1 и 49 като то е уникално спрямо останалите числа и да бъде изпратено към корена на дървото посредством различни съобщения. Дървото има само 1 корен и само 6 листа, от които всички са на едно ниво и са свързани с корена. Ако се случи корена на дървото да получи повтарящо се число от някое от листата си, той непременно изпраща съобщение обратно към листото, от което е получил дубликат и изисква ново число, като листото е длъжно непременно да генерира ново число и да го изпрати отново на корена. Този процес се повтаря докато всяко едно от листата не изпрати уникално число. Освен това корена изпълнява и ролята на мониторинг процес или т.нар. супервайзър. С други думи, ако някое от листата получи съобщение за прекратяване на своето съществуване, тоест процесът е на път да бъде убит, то корена на дървото е длъжен да стартира нов процес, който да замести стария, като самия корен е длъжен да премахне стария и добави новия процес както и всички останали преди това, към структурата си от данни. Корена на дървото съдържа в себе си “Map” като ключът на всяко “entry” отговаря на процесния идентификатор на дадения процес или т.нар. “pid”, а стойността отговаря на числото, което дадения процес(листо) е генерирало и изпратило на корена на дървото. При самото стартиране на програмата, потребителя е длъжен да подаде като аргументи 6 уникални цели числа между 1 и 49. Целта на програмата е посредством по-горе описаната дървовидна структура, да реализира принципа на действие на т.нар. лотарийни игри, или по-конкретно част от работата на “Български Спортен Тотализатор” – “Тото 6 от 49”. След като програмата се изпълни, тоест е генерирала 6 уникални цели числа между 1 и 49, тя трябва да информира потребителя за броя числа, които е успял да познае като сравни неговите числа и числата от дървовидната структура.

“**start**” е основната функция на програмата, защото тя извиква почти всички функции директно, а останалите индиректно. В нея извикваме функция “validate_numbers”, която валидира въведените от потребителя аргументи

към функция “start” задължително като списък от уникални цели числа. След това ако валидацията е успешна, се извиква функция “create_tree”, чрез която се инициализира дървовидната структура. По-късно се изпращат различни видове съобщения към “root” процеса(корена на дървото), за да се изпълни цялата програмна логика, заложена в кода, съобразена със заданието. Ако обаче валидацията не е успешна, тогава се връща грешката.

“**validate_numbers**” е функцията, която има за цел да валидира въведените от потребителя уникални цели числа между 1 и 49. Тази функция проверява дали броят на числата е точно 6. Също така проверява дали всички числа са уникални, тоест няма дубликати. Накрая се проверява дали числата са цели и между 1 и 49 като освен това дали изобщо аргументите са числа или не.

“**create_tree**” е една от най-важните функции в програмата, тъй като тя инициализира дървовидната структура от ерланг процеси. Първия процес, който се създава е именно мониторинг процеса или т.нар. супервайзър. Също така този процес е и корен на дървото. След него се създават и всички останали процеси, които са листа на дървото и биват свързани директно с корена.

“**root**” функцията агрегира, обработва и отговаря на всички съобщения изпратени към корена на дървото. Едно от най-важните свойства на “root” процеса е , че му се задава флагът – “trap_exit” да бъде “true”. Благодарение на този флаг, при ликвидиране на дъщерен процес, “root” процеса остава жив, защото възприема “exit” като обикновено съобщение, а не като сигнал за убиване на процес. Функцията сама по себе си е най-сложната в цялата програма, защото е програмирана да обработва най-голям брой съобщения с различен брой аргументи изпълняващи различна логика, но със свързан смисъл помежду си. А именно – обработка, проверка и съхранение на уникални числа, заявка за ново число/а, премахване на процес, добавяне на процес и показване на краен резултат.

“**leaf**” функцията задава конкретен стандарт за това по какъв начин дъщерните процеси(листата) да обработват съобщенията, предвидени за реализацията на програмната логика свързана с тях. А именно – генериране на цяло число между 1 и 49 и изпращането му към корена на дървото, както

и обработка съобщения за убиване на процес(exit). Всякакви други съобщения, които не са предвидени са заложени да попаднат в т.нар. “Any” клауза.

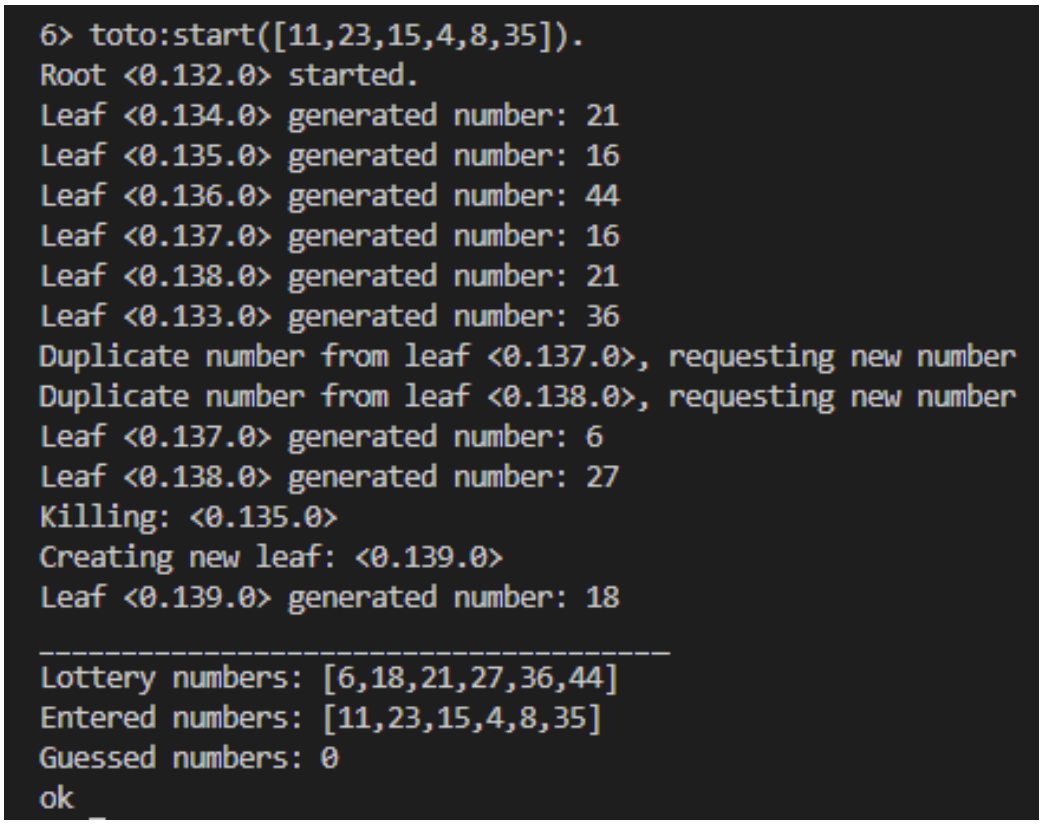
3. Упътване за изпълнение на програмата

За успешно стартиране и изпълнение на програмата е необходимо на машината, която изпълнява програмния код, да бъде инсталирана Ерланг виртуалната машина и компилатор. За предпочитание е версия 26, тъй като кода е писан, тестван и компилиран на тази версия на Ерланг.

1. Компилирайте файла **toto.erl** с командата : “**erlc toto.erl”**
2. Въведете команда “**erl toto.beam**”, за да стартирате генерирания файл от предишната команда в runtime средата на ерланг.
3. Въведете команда “**toto:start([1,2,3,4,5,6]).**” Като числата от 1 до 6 са примерни. Вие можете да въведете числа по ваше желание, стига да са между 1 и 49, да са 6 на брой и да са уникални.
4. Наблюдавайте изпълнението на програмата и проследете колко от числата, които сте въвели отговарят на случайно генерираните числа от дървовидната структура. Шансът за джакпот е 1 към 13 983 816. Успех!

4. Резултат от изпълнение на програмата

Като резултат от изпълнение на програмата се изписва на екрана работата на всички ерланг процеси от дървовидната структура, както и броя числа, които потребителя е успял да познае. Също така от *фиг.1.* може да се види, че при генериране на число, което вече е било генерирано от някой от другите процеси, то се изписва на екрана за конкретния процес и се генерира ново число.



Фиг.1. Резултат от изпълнение на програмата

5. Код на програмата

```
-module(toto).
-import(timer, [sleep/1]).
-export([create_tree/2, root/2, leaf/1, validate_numbers/1, start/1]).

-define(MAX_LEAVES, 6).
-define(DELETED_LEAF, 3).

Create_tree(Size, EnteredNumbers) ->
    Root = spawn(toto, root, [#{}, EnteredNumbers]),
    io:format("Root ~p started.~n", [Root]),
    lists:foreach(
        fun(_) ->
```

```
        Leaf = spawn_link(toto, leaf, [Root]),
        Root ! {setLeaf, Leaf}
    end,
    lists:seq(1, Size)
  ),
  Root.

Root(Map, EnteredNumbers) ->
  process_flag(trap_exit, true),
  receive
    work ->
      lists:foreach(
        fun(Leaf) ->
          Leaf ! work
        end,
        maps:keys(Map)
      ),
      root(Map, EnteredNumbers);
    {return, GeneratedNumber, Leaf} ->
      Values = maps:values(Map),
      case lists:member(GeneratedNumber, Values) of
        true ->
          % If the number exists, ask the leaf to generate a new number
          io:format("Duplicate number from leaf ~p, requesting new number ~n", [Leaf]),
          Leaf ! work,
          root(Map, EnteredNumbers);
        false ->
          % If the number does not exist, update the map with a new entry <Leaf,GeneratedNumber>
          NewMap = maps:put(Leaf, GeneratedNumber, Map),
```

```
            root(NewMap, EnteredNumbers)
        end;
    {setLeaf, Leaf} ->
        % Initialize the new leaf in the map with a placeholder number(0)
        NewMap = maps:put(Leaf, 0, Map),
        root(NewMap, EnteredNumbers);
    {exiting, Leaf} ->
        % Remove the Leaf's entry from the map
        NewMap = maps:remove(Leaf, Map),
        NewLeaf = spawn_link(toto, leaf, [self()]),
        io:format("Creating new leaf: ~p ~n", [NewLeaf]),
        NewLeaf ! work,
        root(NewMap, EnteredNumbers);
    {exit_test, Nth} ->
        case lists:nth(Nth, maps:keys(Map)) of
            Leaf when is_pid(Leaf) ->
                Leaf ! exit,
                root(Map, EnteredNumbers)
        end;
    print ->
        SortedValues = lists:sort(maps:values(Map)),
        io:format("_____~n"),
        io:format("Lottery numbers: ~p~n", [SortedValues]),
        io:format("Entered numbers: ~p~n", [EnteredNumbers]),
        GuessedNumbers = length([Num || Num <- EnteredNumbers, lists:member(Num,
SortedValues)]),
        io:format("Guessed numbers: ~p~n", [GuessedNumbers]),
        root(Map, EnteredNumbers);
    Any ->
        io:format("Unrecognized message: ~p~n", [Any]),
        root(Map, EnteredNumbers)
```

```
    end.

Leaf(Root) ->
    receive
        work ->
            sleep(100),
            GeneratedNumber = rand:uniform(49),
            io:format("Leaf ~p generated number: ~p ~n", [self(), GeneratedNumber]),
            Root ! {return, GeneratedNumber, self()},
            leaf(Root);
        exit ->
            io:format("Killing: ~p ~n", [self()]),
            Root ! {exiting, self()},
            exit(normal);
        Any ->
            io:format("Unrecognized message: '~p' ~n", [Any]),
            leaf(Root)
    end.

Validate_numbers(EnteredNumbers) ->
    if
        length(EnteredNumbers) =/= 6 ->
            {error, "Input must contain exactly 6 numbers."};
        true ->
            UniqueNumbers = lists:usort(EnteredNumbers),
            if
                length(UniqueNumbers) =/= 6 ->
                    {error, "Numbers must be unique."};
                true ->
                    case
```

```
                lists:all(
                    fun(N) -> is_integer(N) andalso N >= 1 andalso N =< 49 end,
                    UniqueNumbers
                )
            of
                true -> ok;
                false -> {error, “All numbers must be whole numbers between 1 and 49.”}
            end
        end
    end.

Start(EnteredNumbers) when is_list(EnteredNumbers) ->
    case validate_numbers(EnteredNumbers) of
        ok ->
            Root = create_tree(?MAX_LEAVES, EnteredNumbers),
            Root ! work,
            sleep(500),
            Root ! {exit_test, ?DELETED_LEAF},
            sleep(500),
            Root ! print,
            sleep(10),
            ok;
        {error, Reason} ->
            {error, Reason}
    end.
```

6. Източници

1. Erlang Reference Manual – User’s Guide
2. Programming Erlang – Joe Armstrong
3. Stack Overflow
4. GeeksforGeeks