Murilo Rosa (201900689) e Nazar Berbeka (201907148)

Programação Concorrente: Trabalho Prático 1

Introdução

Para esse trabalho prático foi desenvolvido um pequeno programa em Erlang, separado em 4 módulos. Cada um desses com uma funcionalidade específica, nomeadamente, cliente, servidor, polynomial, testes. Para além disso, foi usada como referência o livro Programming Erlang, Software for a Concurrent World.

Explicação do Funcionamento

Primeiramente, no módulo de cliente temos as funções para fazermos os pedidos de soma, subtração e multiplicação ao nosso servidor. Cada função pode ser chamada passando uma string Host representando uma *url*, ou com apenas os polinômios, sendo assim a Host definida para “localhost” por padrão. Para fazermos então os pedidos, colocamos qual a operação e os valores em tuplos, para então usarmos o módulo *gen_tcp* do erlang para abrir uma conexão e fazer um pedido com o servidor, e, logo após receber o resultado, fechar a conexão.

Já no módulo do servidor, começamos também por iniciar a função *listen/2* do módulo *gen_tcp*, que vai receber nossos pedidos TCPs, e um processo *calculate/0*, para o qual vamos passar os polinômios para normalizá-los e então fazermos os cálculos. Logo após essa inicialização, entramos numa função em que vamos receber os pedidos, processá-los, e chamarmos a função recursivamente. Nessa implementação o nosso servidor é sequencial, por isso os pedidos são recebidos e processados um por vez, ou colocados numa fila.

Para processarmos os processos, primeiramente normalizamos os polinômios e então extraímos qual a funcionalidade que está sendo pedida para chamarmos a função respectiva. Essas estão definidas no módulo polynomial.

Por fim, definimos também um módulo com funções de testes automatizados, para testarmos o funcionamento das nossas funções e especificamente para testar se os cálculos dos polinômios estavam corretos.

Ilustração de Funcionamento

Para usarmos o nosso programa vamos primeiramente compilar os módulos.

```
 1> c(server).
{ok,server}
 2> c(client).
{ok,client}
 3> c(poly).
{ok,poly}
 4> c(test).
{ok,test}
```

Em seguida, podemos usar os testes automatizados para confirmar que as nossas funcionalidades dão os resultados esperados.

```
 5> test:test().
Starting server...
Server started.
Using polynomials:
  5x y^2 z + 3y
  3x^2 + 3y + z x y^2
Requesting sumation, subtraction and multipliation...
Got answers:
  Sum: 6x y^2 z + 6y + 3x^2
  Difference: 4x y^2 z + -3x^2
  Product: 15x^3 y^2 z + 18x y^3 z + 5x^2 y^4 z^2 + 9y x^2 + 9y^2
Closing server...
Server successfully closed.
ok
```

Entretanto, se quisermos utilizar as nossas funções manualmente, iniciamos o nosso servidor em um processo usando a função de *spawn* da shell e podemos fazer os pedidos usando o módulo de cliente, passando os polinômios como argumentos (nesse caso não precisamos passar uma *URL*, pois o padrão é o “localhost”).

```
 6> spawn(server, start, []).
<0.103.0>
 7> client:request_sum( %% (2*x^2 + 3*y*z) + (x^2)
 7>     [
 7>         {2, [{x, 2}]},
 7>         {3, [{y, 1}, {z, 1}]}
 7>     ], [
 7>         {1, [{x, 2}]}
 7>     ]).
{success,[{3,[{x,2}]},{3,[{y,1},{z,1}]}]}
```

Podemos assim fazer o mesmo para cada uma das outras funcionalidades.

```
 8> client:request_subtract( [ {2, [{x, 2}]},{3, [{y, 1}, {z, 1}]} ], [ {1, [{x,
2}]} ]).
{success,[{3,[{x,2}]},{3,[{y,1},{z,1}]}]}
 9> client:request_multiply( [ {2, [{x, 2}]},{3, [{y, 1}, {z, 1}]} ], [ {1, [{x,
2}]} ]).
{success,[{3,[{x,2}]},{3,[{y,1},{z,1}]}]}
```