Sistemas Distribuídos

trabalho: Chamada Remota de Procedimento

março de 2021

Trabalho: RPC com Lua

```
lrpc = require “luarpc”
…
o1 = { foo = function(a, b)
         return a+b, "alo alo"
       end,
       boo = function (z)
         armazena(z)
       end,
       bar = function(a, b)
         return a-b, "tchau tchau"
       end,
     }
o2 = ( foo = function(m, n) …
     }
ip, p = lrpc.registerServant (idl, o1)
print ("sou 1, estou esperando reqs na porta " ..
p)
ip, p = lrpc.registerServant (idl, o2)
print ("sou 2, estou esperando reqs na porta " ..
p)
lrpc.waitIncoming()
```

```
lrpc = require “luarpc”
…
rep1 = lrpc.createProxy (idlserv, ip, porta)
rep2 = lrpc.createProxy(idlserv, ip, outraporta)
…
print (rep1:foo(4,5))
rep1:boo(1003)
print (rep2:foo(x,y))
```

- tanto cliente como servidor são single-threaded
- servidor deve poder receber pedidos para qualquer servente

Trabalho: RPC com Lua

```
lrpc = require “luarpc”
…
o1 = { foo = function(a, b)
         return a+b, "alo alo"
       end,
…
o2 = { foo = function(a, b)
          …
         f = function(m, n)
          …
      }
ip1, p1 = lrpc.registerServant (idl, o1)
ip2, p2 = lrpc.registerServant (idl, o2)
ip3, p3 = lrpc.registerServant (idl, o2)
…
lrpc.waitIncoming()
```

```
lrpc = require “luarpc”
…
rep1 = lrpc.createProxy (idl1, ip, porta)
rep2 = lrpc.createProxy (idl1, ip, outraporta)
…
print (rep1:foo(4,5))
rep1:boo(1003)
print (rep2:foo(x,y))
```

```
lrpc = require “luarpc”
…
p = lrpc.createProxy (idl2, ip, porta3)
…
print (p:f (x, y))
```

RPC com Lua — IDL

```
struct { name = "minhaStruct",
         fields = {{name = "nome",
                    type = "string"},
                   {name = "peso",
                    type = "double"}
                   {name = "idade",
                    type = "int"},
                   }
       }
interface { name = "minhaInt",
            methods = {
               foo = {
                 resulttype = "double",
                 args = {{direction = "in",
                          type = "double"},
                         {direction = "in",
                          type = "string"},
                         {direction = "in",
                          type = "minhaStruct"},
                         {direction = "out",
                          type = "int"}
                        }

               },
               boo = {
                 resulttype = "void",
                 args = {{direction = "in",
                          type = "double"},
                         {direction = "out",
                          type = "minhaStruct",
                        }
               }
             }
            }
```

Alguns pontos importantes

- Definir protocolo de comunicação.
- Um servidor deve ser capaz de abrigar vários objetos de serviço (*servants*), cada um em porta distinta.
- Inicialmente abrir e fechar conexão a cada requisição. Depois que estiver funcionando:
  - servidor deve manter um pool de 5 conexões abertas
  - cliente deve estar sempre preparado para conexão fechada
- Verificações: ao construir a chamada, verificar se está de acordo com IDL
  - conversões e adaptações de números de parâmetros são admissíveis para Lua
  - tipos e valores que não têm conversão razoável não devem ser admitidos

Protocolo Cliente-Servidor

- cliente de cada grupo deve ser capaz de falar com servidor de outro
- estabelecer *protocolo* de comunicação

protocolo

- mensagens são strings? (imagino que sim)
- como separar argumentos?
- como organizar structs?
- como indicar fim de requisição?

Criação dinâmica do stub cliente

- sistemas clássicos "compilam" IDL para gerar stubs
- aqui iremos gerar os stubs cliente e servidor dentro do próprio programa

- geração de strings no programa [NÃO]
- criação dinâmica de funções [SIM]

Criação dinâmica de funções e escopo léxico

```
local function f()
  local v = 0
  return function ()
    local val = v
    v = v+1
    return val
  end
end

cont1, cont2 = f(), f()
print(cont1())
print(cont1())
print(cont2())
print(cont1())
print(cont1())
print(cont2())
```

trecho extraído de trabalho anterior

```
function createrpcproxy(hostname, port, interface)
  local functions = {}
  local prototypes = parser(interface)
  for name,sig in pairs(prototypes) do
    functions[name] = function(...)  -- !!!
      -- validating params
      local params = {...}
      local values = {name}
      local types = sig.input
      for i=1,#types do
        if (#params >= i) then
        values[#values+1] = params[i]
        if (type(params[i])~="number") then
          values[#values] = "\"" .. values[#values] .. "\""
        end
        ...
      end
      -- creating request
      local request = pack(values)
      -- creating socket
      local client = socket.tcp()
      ...
      local conn = client:connect(hostname, port)
      ...
      local result = client:send(request .. '\n')
      ...
    end
  end
  return functions;
end
```