The Max Stern Academic
College of Emek Jezreel

Dept. of Information Systems

המכללה האקדמית עמק יזרעאל

ע"ש מקס שטרן

החוג למערכות מידע

Instruction
ERLang Conversation

Lecturer: Mr. Avigdor Rabinovitch

Programmers Team:

Arthur Zarakin

Inga Samkani

Product description

This function applies conversation between the 2 or more clients CMD,

Using a server that store all the client ports in list and calculate for them the function results.

Using UDP method connection.

Each sent message by one will received by other clients.

Each sent math instruction/function will received by other clients with the solution/results.

Executing instruction

For server execute

First run CMD for the server execute

Enter the Folder Address inside the command

```
erl
cd (" Folder Address ").
c(server).
server: start().
```

For example:

```
cd("C: /Users/lp/Desktop/Eclipse/homework/src").
c(server).
server: start().
```

For client execute

Run CMD for each client execute

Enter the Folder Address inside the command

Please enter the client PORT example 4001 4002 4003 4004

```
erl
cd (" Folder Address ").
c(client)
client: start(  PORT ).
```

For example:

```
erl
cd("C: /Users/lp/Desktop/Eclipse/homework/src").
c(client).
client: start(4001).
```

Repeat this command for each client in separate CMD window.

Client orders, with examples

Text message

```
client: send("message text here").
```

Line print

```
client: send("line","*", 5).
```

Rectangle print

```
client: send("rectangle","b", 4, 5).
```

Triangle print

```
client: send("triangle","a", 4).
```

Fibonacci result print

```
client: send("fib", 5).
```

Fact resulte print

```
client: send("fact", 5).
```

Multiplier resulte print

```
client: send("mult", 3, 4).
```

Example for results that will show on other client/s screen (for example for client that using port 4001):

```
client: send("message text here").
```

```
client4001:  message text here
```

```
client: send("line","*", 5).
```

```
client4001:  ("line","*", 5)

*****
```

```
client: send("rectangle","b", 4, 5).
```

```
client4001:  ("rectangle", "b", 4, 5).

bbbb

bbbb

bbbb

bbbb

bbbb
```

```
client: send("triangle", "a", 4).
```

```
client4001: ("triangle", "a", 4).
a
aa
aaa
aaaa
```

```
client: send("fib", 5).
```

```
client4001: ("fib",5) = 5.
```

```
client: send("fact", 5).
```

```
client4001: 5! = 120.
```

```
client: send("add", 1, 2).
```

```
client4001: 1 + 2 = 3
```

```
client: send("mult", 3, 4).
```

```
client4001: 3 X 4 = 12
```

Additional functions

Client exit function, will send all the client that this client port left the conversation and delete it from server port list.

```
client: exit().
```

For server exit function, will send all the client that this client that the server closed and will close the UDP server

```
server: exit().
```

Server actual port list (all the clients that registered to it)

```
server: portList().
```

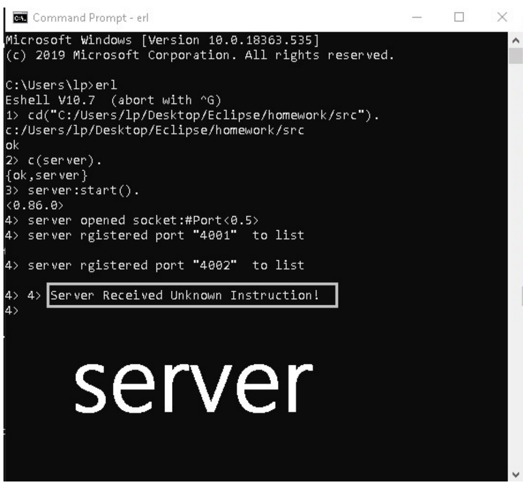
Demonstration of two learned concepts

We will demonstrate two concept that learned in the course:
1- Error handling.
2- Asyncronic prosses.

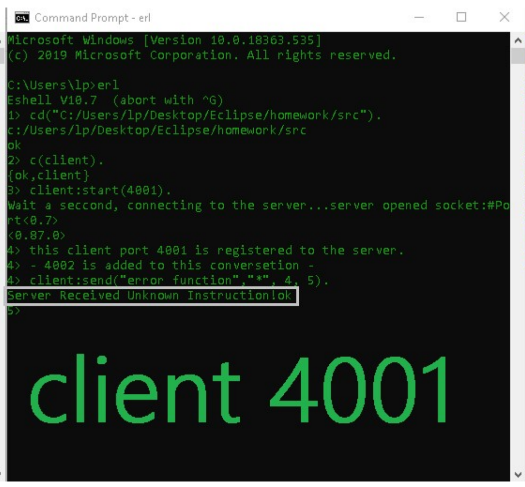
1- Error handling:
This example show sending how server will react on wrong received order/function.

```
client: send("error function", "*",4, 5).
```

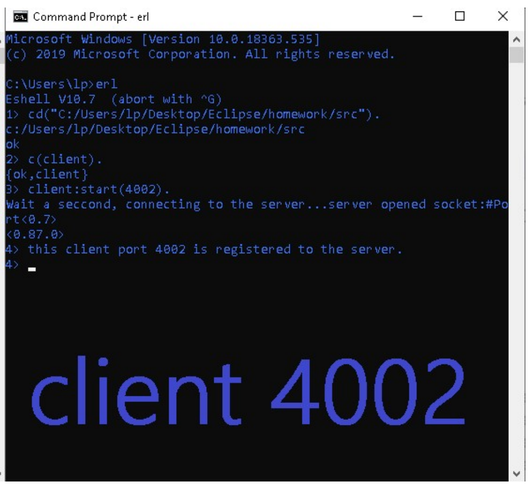
Client and server will receive message message that inform them that that this function is wrong.
The results:



```
Microsoft Windows [Version 10.0.18363.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\lp>erl
Eshell V10.7  (abort with ^G)
1> cd("C:/Users/lp/Desktop/Eclipse/homework/src").
c:/Users/lp/Desktop/Eclipse/homework/src
ok
2> c(server).
{ok,server}
3> server:start().
<0.86.0>
4> server opened socket:#Port<0.5>
4> server rgistered port "4001"  to list

4> server rgistered port "4002"  to list

4> 4> Server Received Unknown Instruction!
4>
```

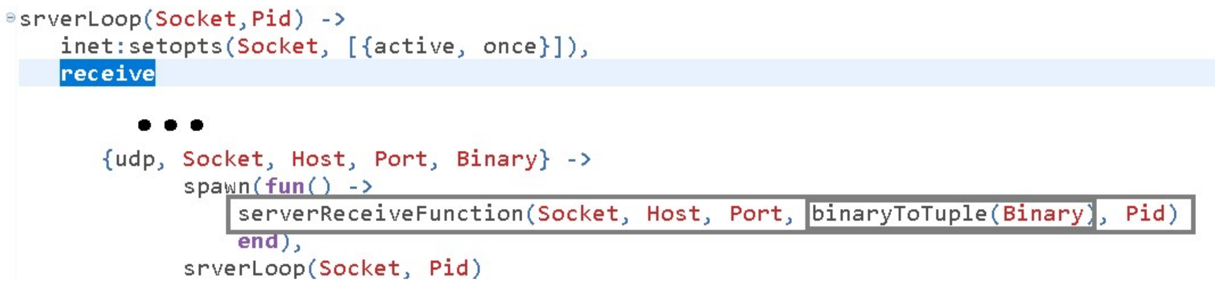


```
Microsoft Windows [Version 10.0.18363.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\lp>erl
Eshell V10.7  (abort with ^G)
1> cd("C:/Users/lp/Desktop/Eclipse/homework/src").
c:/Users/lp/Desktop/Eclipse/homework/src
ok
2> c(client).
{ok,client}
3> client:start(4001).
Wait a seccond, connecting to the server...server opened socket:#Po
rt<0.7>
<0.87.0>
4> this client port 4001 is registered to the server.
4> - 4002 is added to this conversetion -
4> client:send("error function","*", 4, 5).
Server Received Unknown Instruction!ok
5>
```



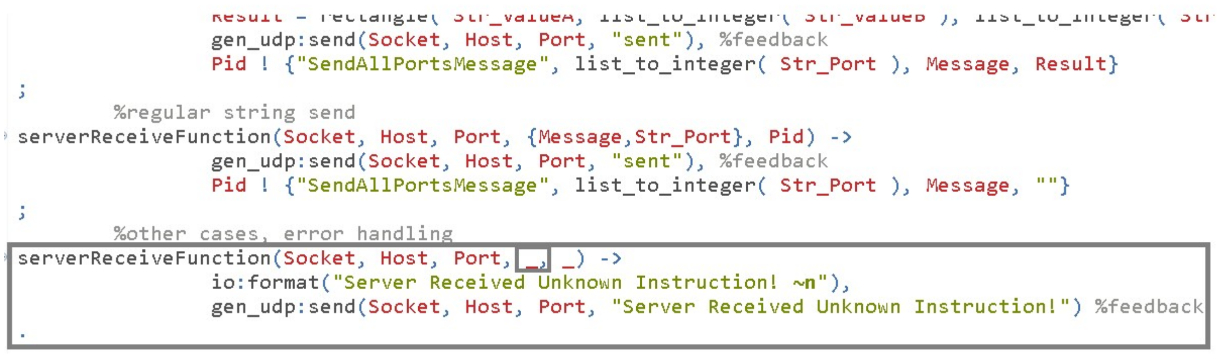
```
Microsoft Windows [Version 10.0.18363.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\lp>erl
Eshell V10.7  (abort with ^G)
1> cd("C:/Users/lp/Desktop/Eclipse/homework/src").
c:/Users/lp/Desktop/Eclipse/homework/src
ok
2> c(client).
{ok,client}
3> client:start(4002).
Wait a seccond, connecting to the server...server opened socket:#Po
rt<0.7>
<0.87.0>
4> this client port 4002 is registered to the server.
4>
```

Based on server.erl file code that :
This code using all other case of order/function that received and not mentioned in the server before.



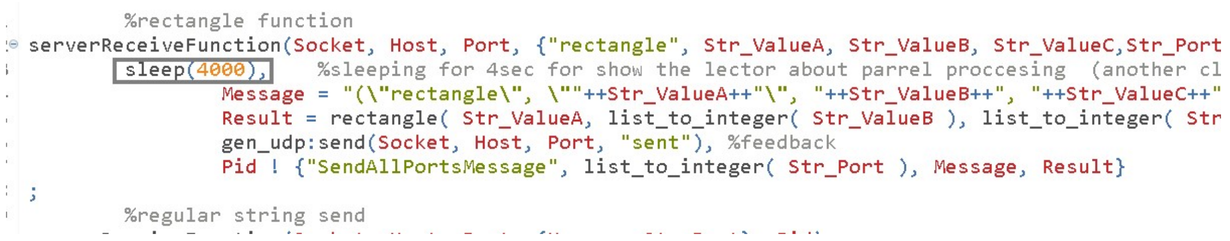
```
srverLoop(Socket,Pid) ->
    inet:setopts(Socket, [{active, once}]),
    receive

        • • •

        {udp, Socket, Host, Port, Binary} ->
            spawn(fun() ->
                serverReceiveFunction(Socket, Host, Port, binaryToTuple(Binary), Pid)
                end),
            srverLoop(Socket, Pid)
```

• • •



```
                Result = rectangle( Str_ValueA, list_to_integer( Str_ValueB ), list_to_integer( Str
                gen_udp:send(Socket, Host, Port, "sent"), %feedback
                Pid ! {"SendAllPortsMessage", list_to_integer( Str_Port ), Message, Result}
;
        %regular string send
serverReceiveFunction(Socket, Host, Port, {Message,Str_Port}, Pid) ->
                gen_udp:send(Socket, Host, Port, "sent"), %feedback
                Pid ! {"SendAllPortsMessage", list_to_integer( Str_Port ), Message, ""}
;
        %other cases, error handling
serverReceiveFunction(Socket, Host, Port, _, _) ->
                io:format("Server Received Unknown Instruction! ~n"),
                gen_udp:send(Socket, Host, Port, "Server Received Unknown Instruction!") %feedback
.
```

2- Asyncronic prosses:
This example will show that sending an order from several clients will show in parralel.
In this case "recatangle" receiving function changed with addition code that make it send to the clients after 4 seconds



```
        %rectangle function
serverReceiveFunction(Socket, Host, Port, {"rectangle", Str_ValueA, Str_ValueB, Str_ValueC,Str_Port
        sleep(4000),    %sleeping for 4sec for show the lector about parrel proccesing  (another cl
                Message = "(\"rectangle\", \""++Str_ValueA++"\", "++Str_ValueB++", "++Str_ValueC++"
                Result = rectangle( Str_ValueA, list_to_integer( Str_ValueB ), list_to_integer( Str
                gen_udp:send(Socket, Host, Port, "sent"), %feedback
                Pid ! {"SendAllPortsMessage", list_to_integer( Str_Port ), Message, Result}
;
        %regular string send
```

After sending in the same time the request from client 4001 to send a "rectangle", we immidiatly after that will request from client 4002 to send a "trinagle",
the result will show that client 4003 will receive first the Triangle print and after 4 seconds will receive the Rectangle print.

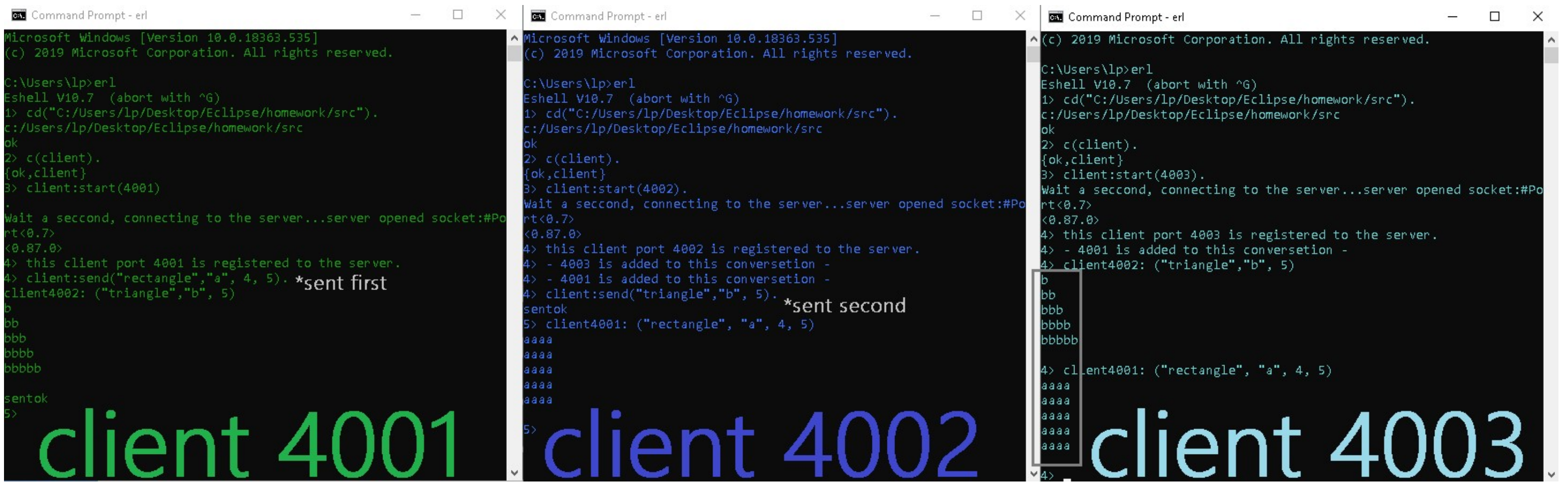
client4001

```
client: send("rectangle", "a",4, 5).
```

client4002

```
client: send("triangle", "b", 5).
```

The results:



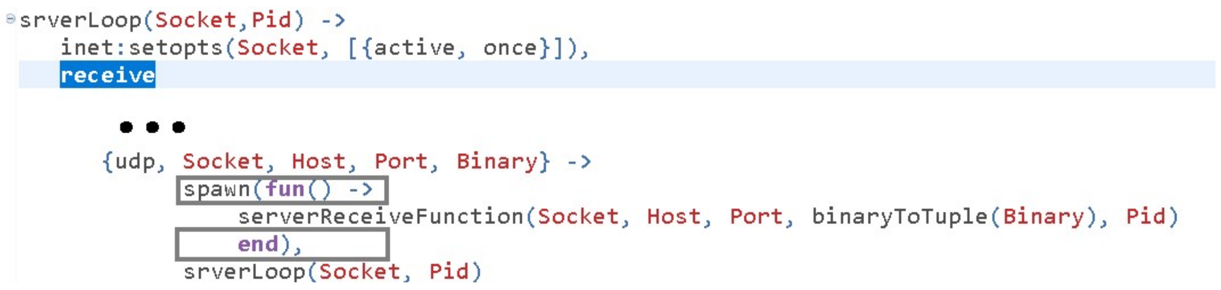
```
Microsoft Windows [Version 10.0.18363.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\lp>erl
Eshell V10.7  (abort with ^G)
1> cd("C:/Users/lp/Desktop/Eclipse/homework/src").
c:/Users/lp/Desktop/Eclipse/homework/src
ok
2> c(client).
{ok,client}
3> client:start(4001)
.
Wait a seccond, connecting to the server...server opened socket:#Po
rt<0.7>
<0.87.0>
4> this client port 4001 is registered to the server.
4> client:send("rectangle","a", 4, 5). *sent first
client4002: ("triangle","b", 5)
b
bb
bbb
bbbb
bbbbb

sentok
5>
```

```
Microsoft Windows [Version 10.0.18363.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\lp>erl
Eshell V10.7  (abort with ^G)
1> cd("C:/Users/lp/Desktop/Eclipse/homework/src").
c:/Users/lp/Desktop/Eclipse/homework/src
ok
2> c(client).
{ok,client}
3> client:start(4002).
Wait a seccond, connecting to the server...server opened socket:#Po
rt<0.7>
<0.87.0>
4> this client port 4002 is registered to the server.
4> - 4003 is added to this conversetion -
4> - 4001 is added to this conversetion -
4> client:send("triangle","b", 5).
sentok                                 *sent second
5> client4001: ("rectangle", "a", 4, 5)
aaaa
aaaa
aaaa
aaaa
aaaa

5>
```

```
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\lp>erl
Eshell V10.7  (abort with ^G)
1> cd("C:/Users/lp/Desktop/Eclipse/homework/src").
c:/Users/lp/Desktop/Eclipse/homework/src
ok
2> c(client).
{ok,client}
3> client:start(4003).
Wait a seccond, connecting to the server...server opened socket:#Po
rt<0.7>
<0.87.0>
4> this client port 4003 is registered to the server.
4> - 4001 is added to this conversetion -
4> client4002: ("triangle","b", 5)
b
bb
bbb
bbbb
bbbbb

4> client4001: ("rectangle", "a", 4, 5)
aaaa
aaaa
aaaa
aaaa
aaaa

4>
```

Based on server.erl file code that : runs all the received orders/functions in parralel, because of this lines (spawn(fun()-> …)

For running the server syncronicly remove this lines, so it will print the after 4 seconds the rectangle and only after that the triangle.



```
srverLoop(Socket,Pid) ->
    inet:setopts(Socket, [{active, once}]),
    receive

        • • •

      {udp, Socket, Host, Port, Binary} ->
            spawn(fun() ->
                serverReceiveFunction(Socket, Host, Port, binaryToTuple(Binary), Pid)
                end),
            srverLoop(Socket, Pid)
```

Server file - how it works diagrams

Server:

```
exit() -> sendSelfOrder("exit").
```

Server (4000) will send message to himself with orders {"Exit"}

```
portList() -> sendSelfOrder("portList").
```

Server (4000) will send message to himself with orders {"Exit"}

```
sendSelfOrder(Order)
    {ok, Socket} = gen_udp:open(0, [binary])
    ok = gen_udp:send(Socket, "localhost", 4000, "{\""++Order++"\"}"),
    ...
    gen_udp:close(Socket)
```

Server (4000) will send message to himself with orders like {"Print"} {"Exit"} {"Delete", Port}

*This functions for internal use

```
allPortsSend(List ,ThisPort, Message)
```

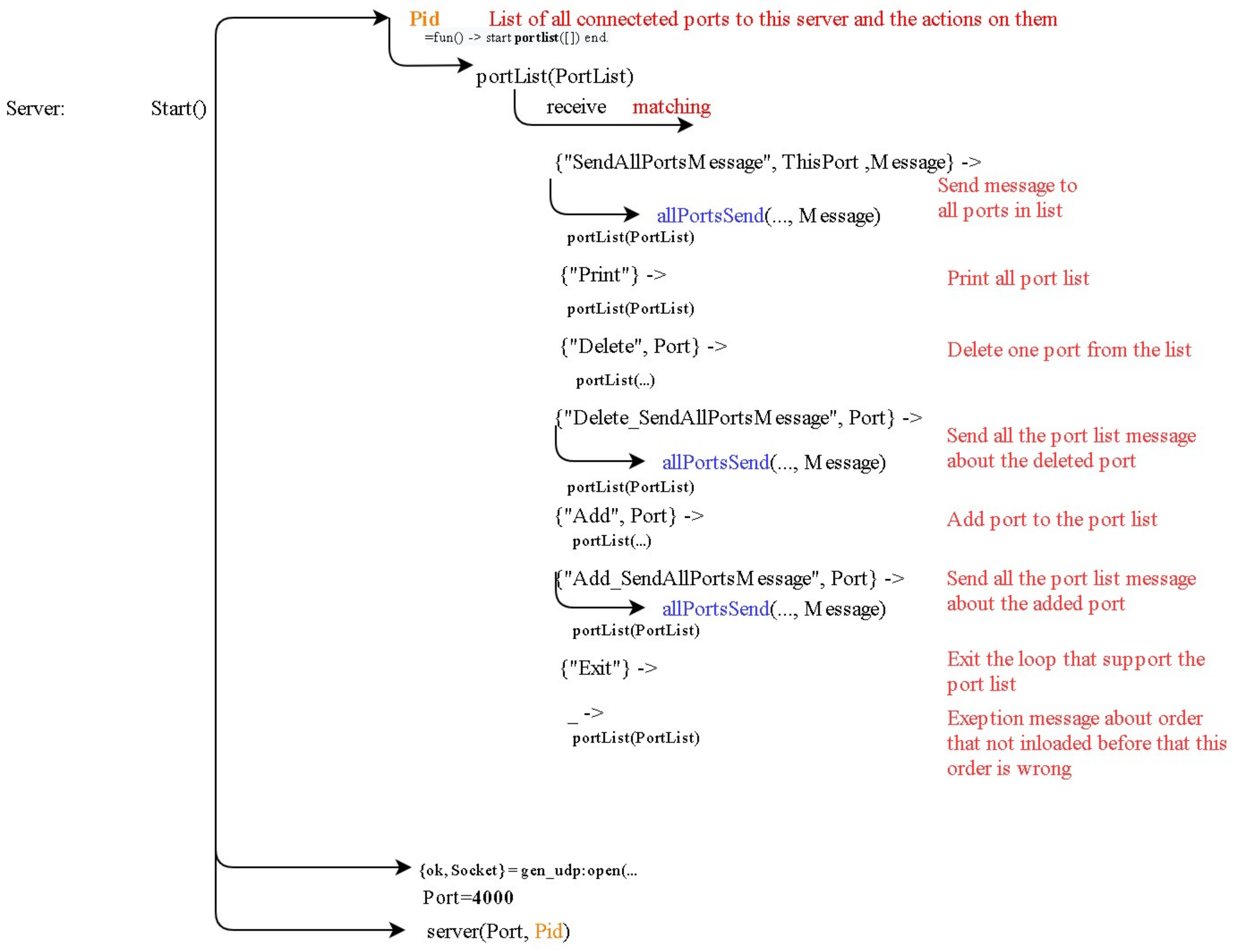
Receiverd List from Pid order (SendAllPortsMessage, Add_SendAllPortsMessage, Delete_SendAllPortsMessage)

```
send(Message, Port)
{ok, Socket} = gen_udp:open(0, [binary])
ok = gen_udp:send(Socket, "localhost", Port, Message),
...
gen_udp:close(Socket)
```

Send Message from the server to the client (one by one)

if the client will not react to received post then he will delete from Pid list

line(Char,N)	Creating line N times with Char Character/String
rectangle(Char, Length, Height)	Creating rectangle from Char string with required Length and Height
triangle(Char, Base)	Creating triangle from Char string with the required Base
fib(N)	returning the N Fibonacci as integer
fact(N)	returning the N fact (!) as integer
binaryToTuple(Binary)	Return tuple from binary input
binaryToString(Binary)	Return string from binary input
stringToTuple(String)	Return tuple from string input



Server: Start()

Pid =fun() -> start **portlist**([]) end.

List of all connecteted ports to this server and the actions on them

portList(PortList)

receive matching

```
{"SendAllPortsMessage", ThisPort ,Message} ->
        allPortsSend(..., Message)
    portList(PortList)
{"Print"} ->
    portList(PortList)
{"Delete", Port} ->
    portList(...)
{"Delete_SendAllPortsMessage", Port} ->
        allPortsSend(..., Message)
    portList(PortList)
{"Add", Port} ->
    portList(...)
{"Add_SendAllPortsMessage", Port} ->
        allPortsSend(..., Message)
    portList(PortList)
{"Exit"} ->
_ ->
    portList(PortList)
```

Send message to all ports in list

Print all port list

Delete one port from the list

Send all the port list message about the deleted port

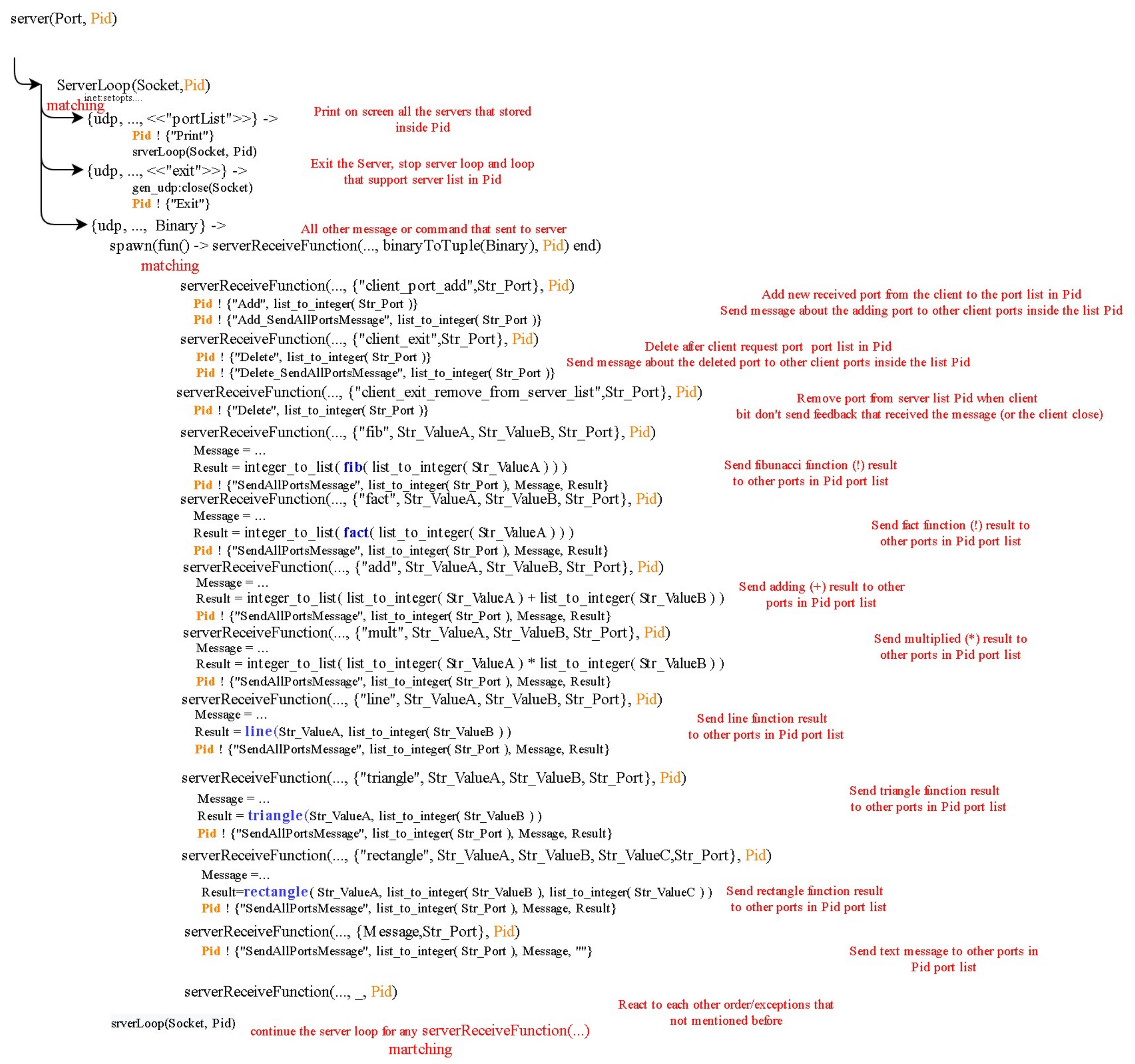
Add port to the port list

Send all the port list message about the added port

Exit the loop that support the port list

Exeption message about order that not inloaded before that this order is wrong

```
{ok, Socket}=gen_udp:open(...
Port=4000
server(Port, Pid)
```



```
server(Port, Pid)

  ServerLoop(Socket,Pid)
          inet:setopts....
  matching
    {udp, ..., <<"portList">>} ->
          Pid ! {"Print"}
          srverLoop(Socket, Pid)
    {udp, ..., <<"exit">>} ->
          gen_udp:close(Socket)
          Pid ! {"Exit"}
    {udp, ..., Binary} ->
      spawn(fun() -> serverReceiveFunction(..., binaryToTuple(Binary), Pid) end)
        matching
          serverReceiveFunction(..., {"client_port_add",Str_Port}, Pid)
            Pid ! {"Add", list_to_integer( Str_Port )}
            Pid ! {"Add_SendAllPortsMessage", list_to_integer( Str_Port )}
          serverReceiveFunction(..., {"client_exit",Str_Port}, Pid)
            Pid ! {"Delete", list_to_integer( Str_Port )}
            Pid ! {"Delete_SendAllPortsMessage", list_to_integer( Str_Port )}
          serverReceiveFunction(..., {"client_exit_remove_from_server_list",Str_Port}, Pid)
            Pid ! {"Delete", list_to_integer( Str_Port )}
          serverReceiveFunction(..., {"fib", Str_ValueA, Str_ValueB, Str_Port}, Pid)
            Message = ...
            Result = integer_to_list( fib( list_to_integer( Str_ValueA ) ) )
            Pid ! {"SendAllPortsMessage", list_to_integer( Str_Port ), Message, Result}
          serverReceiveFunction(..., {"fact", Str_ValueA, Str_ValueB, Str_Port}, Pid)
            Message = ...
            Result = integer_to_list( fact( list_to_integer( Str_ValueA ) ) )
            Pid ! {"SendAllPortsMessage", list_to_integer( Str_Port ), Message, Result}
          serverReceiveFunction(..., {"add", Str_ValueA, Str_ValueB, Str_Port}, Pid)
            Message = ...
            Result = integer_to_list( list_to_integer( Str_ValueA ) + list_to_integer( Str_ValueB ) )
            Pid ! {"SendAllPortsMessage", list_to_integer( Str_Port ), Message, Result}
          serverReceiveFunction(..., {"mult", Str_ValueA, Str_ValueB, Str_Port}, Pid)
            Message = ...
            Result = integer_to_list( list_to_integer( Str_ValueA ) * list_to_integer( Str_ValueB ) )
            Pid ! {"SendAllPortsMessage", list_to_integer( Str_Port ), Message, Result}
          serverReceiveFunction(..., {"line", Str_ValueA, Str_ValueB, Str_Port}, Pid)
            Message = ...
            Result = line(Str_ValueA, list_to_integer( Str_ValueB ) )
            Pid ! {"SendAllPortsMessage", list_to_integer( Str_Port ), Message, Result}

          serverReceiveFunction(..., {"triangle", Str_ValueA, Str_ValueB, Str_Port}, Pid)
            Message = ...
            Result = triangle(Str_ValueA, list_to_integer( Str_ValueB ) )
            Pid ! {"SendAllPortsMessage", list_to_integer( Str_Port ), Message, Result}
          serverReceiveFunction(..., {"rectangle", Str_ValueA, Str_ValueB, Str_ValueC,Str_Port}, Pid)
            Message =...
            Result=rectangle( Str_ValueA, list_to_integer( Str_ValueB ), list_to_integer( Str_ValueC ) )
            Pid ! {"SendAllPortsMessage", list_to_integer( Str_Port ), Message, Result}
          serverReceiveFunction(..., {Message,Str_Port}, Pid)
            Pid ! {"SendAllPortsMessage", list_to_integer( Str_Port ), Message, ""}

          serverReceiveFunction(..., _, Pid)
      srverLoop(Socket, Pid)
```

Print on screen all the servers that stored inside Pid

Exit the Server, stop server loop and loop that support server list in Pid

All other message or command that sent to server

Add new received port from the client to the port list in Pid
Send message about the adding port to other client ports inside the list Pid

Delete after client request port port list in Pid
Send message about the deleted port to other client ports inside the list Pid

Remove port from server list Pid when client bit don't send feedback that received the message (or the client close)

Send fibunacci function (!) result to other ports in Pid port list

Send fact function (!) result to other ports in Pid port list

Send adding (+) result to other ports in Pid port list

Send multiplied (*) result to other ports in Pid port list

Send line function result to other ports in Pid port list

Send triangle function result to other ports in Pid port list

Send rectangle function result to other ports in Pid port list

Send text message to other ports in Pid port list

React to each other order/exceptions that not mentioned before

continue the server loop for any serverReceiveFunction(...) martching

The Max Stern Academic
College of Emek Jezreel

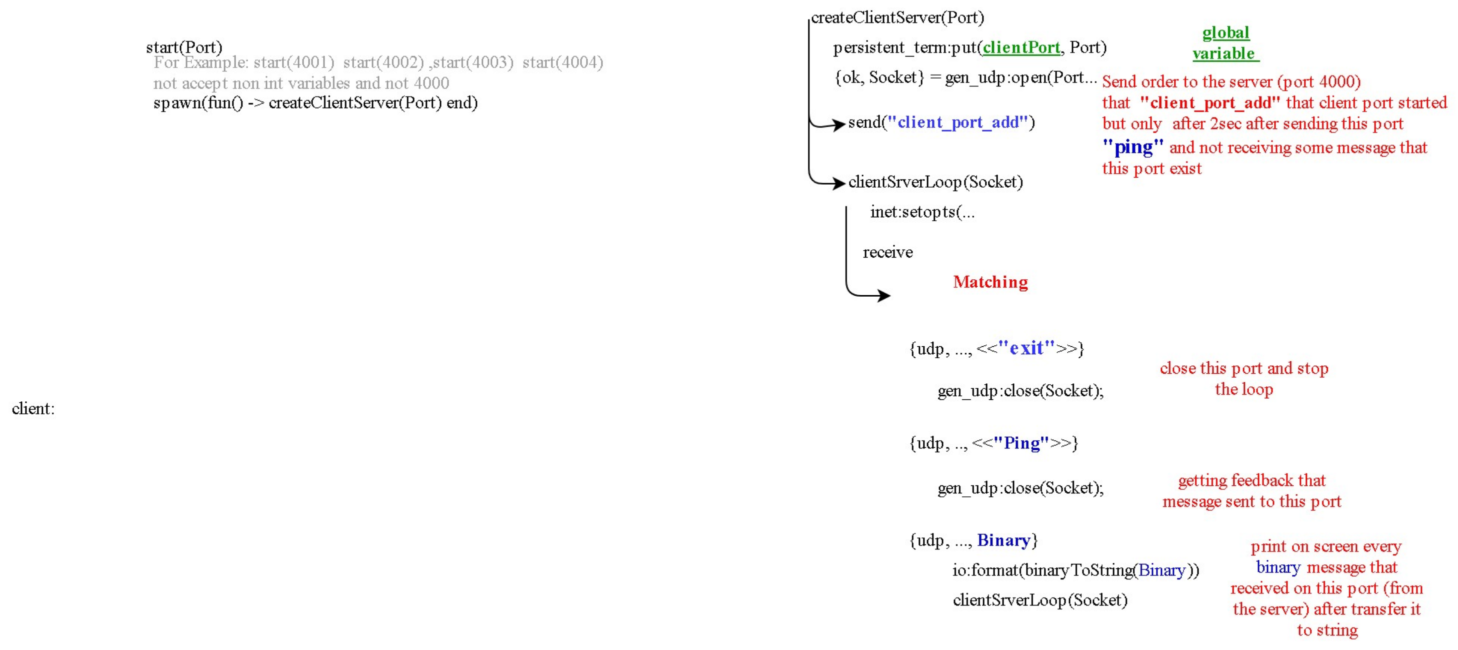
Dept. of Information Systems

המכללה האקדמית עמק יזרעאל

ע"ש מקס שטרן

החוג למערכות מידע

Client file - how it works diagrams



```
client:

start(Port)
 For Example: start(4001)  start(4002) ,start(4003)  start(4004)
 not accept non int variables and not 4000
 spawn(fun() -> createClientServer(Port) end)

createClientServer(Port)
   persistent_term:put(clientPort, Port)        global variable
   {ok, Socket} = gen_udp:open(Port...
 -> send("client_port_add")
 -> clientSrverLoop(Socket)
       inet:setopts(...
       receive
                Matching
       ->
          {udp, ..., <<"exit">>}
             gen_udp:close(Socket);

          {udp, .., <<"Ping">>}
             gen_udp:close(Socket);

          {udp, ..., Binary}
               io:format(binaryToString(Binary))
               clientSrverLoop(Socket)
```

Send order to the server (port 4000) that "client_port_add" that client port started but only after 2sec after sending this port "ping" and not receiving some message that this port exist

close this port and stop the loop

getting feedback that message sent to this port

print on screen every binary message that received on this port (from the server) after transfer it to string

```
send(Data)

    ThisClientPort_String = integer_to_list( persistent_term:get(clientPort) )
    {ok, Socket} = gen_udp:
    ok = gen_udp:send (Socket, "localhost", 4000, "{\""++Data++"\",\""++ThisClientPort_String++"\"}"),
    ...
    gen_udp:close(Socket)
```

Send matching order or message to the server (port 4000) with information about this client port (by global variable)

after **10** sec will activeate: exitSend("") that will close this client port and loop.

```
send(Order, ValueA) -> send(Order++"\", \""++integer_to_list( ValueA ))
```

send matching order to the server with 1 integer
example: send("fib", 5)
for Fibonacci function
or for send("fact", 5) for 5! result

```
send(Order, ValueA, ValueB) when is_integer(ValueA) -> send(Order++"\", \""++ integer_to_list( ValueA ) ++"\", \""++integer_to_list( ValueB ));
```

send matching order to the server with 2 integers
example: send("random", 1, 5)
for random number 1 to 5
*this specific function still not set

```
send(Order, ValueA, ValueB) -> send(Order++"\", \""++ ValueA ++"\", \""++integer_to_list( ValueB )).
```

send matching order to the server with 1 string and 1 integer
example: send("line", "*", 5)
for printing line of *, 5 times *****
also for send("triangle", "b", 3)
that will print triangle with base b 3 times

```
send(Order, ValueA, ValueB, ValueC) -> send(Order++"\", \""++ ValueA ++"\", \""++integer_to_list( ValueB )++"\", \""++integer_to_list( ValueC )).
```

send matching order to the server with 1 string and 2 integer
example: send("rectangle", "a", 5, 4)
for printing rectangle with a length 5 height 4
aaaaa
aaaaa
aaaaa
aaaaa

```
exitSend(AdditionalFunction)

    ThisClientPort = persistent_term:get(clientPort),
    {ok, Socket} = gen_udp: ...
    ok = gen_udp:send(Socket, ..., "exit" )

     AdditionalFunction
gen_udp:close(Socket)
```

Send matching order to this client port (by global variable) that activate the **"exit"** matching

```
exit() -> exitSend( send("client_exit") ).
```

exit() will active additional function that will send the server matching function **"client_exit"** that will delete this port from the port list on the server and will send message that this client port left the conversetion

```
binaryToString(Binary)
```

Helping internal function that returning String after receiving Binary