Transparent Architecture - communication SW for distributed systems.

Examples Description

What is this:

1. TA is purposed for distributed systems development based upon Client-Server and Publish-Subscribe models, low level Messaging is also supported. TA is developed as a substitutive technology for Client-Server CORBA, COM and Publish-Subscribe DDS.

2. TA is exclusively based on non-blocking communication, it is free of hangs, common for COM and CORBA, and needs no separate threads waiting for data available.

3. TA uses SRPS protocol type for LANs and SOAP/HTTP for WANs. On Local physical layer the following are used: UDP multicast, UDP, TCP, ICMP, Shared Memory. DLL/SO is used for inproc communication, in Unix version Unix Domain Sockets and MSGQ are used also. The set of protocols supported can be expanded.

4. TA is a pilot project. Two variants are supplied: TA for C and TA for BlackBox. Here is Ta for C, tested for Linux and Windows. The parts concerning Linux only are marked brown. The parts concerning Windows only are marked blue.

5. TA is distributed in sources under LGPL.

How to make:

TA consists of library in src and examples in app.

For Linux
tar xvzf ta1.tgz
cd ta1/
make

For Windows
Unpack ta1.zip
With VC++ 2010
Open ta1s.sln from ta1\src
Make library ta1s.dll
Copy ta1s.dll to C:\Windows\
Open winapp.sln from ta1\app\winapp
Make examples winapp
Binaries can be found in ta\winapp\Release

How to use:
1. For Linux
TA requires path to library libta.so. The path to src must be set (if your dir is ~/ta1/src):
export LD_LIBRARY_PATH=~/ta1/src:$ LD_LIBRARY_PATH

Run all from app

For Windows
TA requires ta1s.dll library access, it should be copied to C:\Windows or to winapp.

Run all from winapp

2. TA Usage requires basic knowledge of BlackBox and networks configuration.

Examples:
1. **Ping Test** ICMP protocol test
2. **Noahs Ark Test** How to place different types of species in one Ark for long network trip
3. **Nodex Test** Nodes message exchanger
3-A. **Nodex Exe** Nodes message exchanger server
4-u. **RPS Unicast Exe** Rock-Paper-Scissors Contest for unicast
4-m. **RPS Multicast Exe** Rock-Paper-Scissors Contest for multicast
5. **Srch** Pattern Search - Messaging Server
5-F. **FSrch** File Pattern Search - Client-Server Architecture
5-P. **PSrch** File Pattern Search - Publish-Subscribe Architecture
6. **GWeather** SOAP/HTTP, web-client and web-service.

Respectfully, TA SW developer
Dmitry V. Dagaev dvdagaev@yahoo.com

1. PingTest - implementation of PING, file pings.c

Before establishing SW connection you need to check "visibility" of remote nodes via Ethernet. The example below deals with ICMP protocol implementation (PING description can be observed from http://www.ping127001.com/pingpage.htm)

For any Ta task the **Area** object defines virtual area of visibility with name "ping" and number 1983 in our case. It is divided into **Station** objects each containing up to 8 **Channel** objects (each represents communication channel), each of the channels is sending and receiving **Message** objects.
Configuration is implemented via QoS (Quality-of-Service) aqos/sqos/cqos, correspondently for Area, Station, Channel objects.

In order to ping addrs 10.0.201.216 10.0.201.206 (for example) from your computer 3 Stations with IP-addresses 127.0.0.1 10.0.201.216 10.0.201.206 must be set. Protocols are set by "enable_mask" to TA_ENABLE_ICMP. It means that only station.channels[0] with ICMP settings are used. The stationUpdate interval is set in "-i interval" in seconds, so it sends ping 1 time a second by "taSetMessage" call and checks return codes. After “-c count” cycles table of results is printed.
Analysing for 0-channel of the station the number of send/received bytes "complete_send_rc", "complete_recv_rc" and error codes "send_errno", "recv_errno" - we can get the ping result.
Runtime configuration: adrs, count and cycle time. You can set hostnames that can be resolved by DNS.

Keep in the mind, that you need administrative priviledge to work with ICMP and Raw Sockets in general.

For Linux by superuser: chown root pings;chmod 777 pings;chmod u+s pings
For Windows login with administrative priviledge

Naturally your IPs are different! Here and below **set your IPs in commander line**.

```
pings -i 1 -c 3 10.0.220.39 10.0.201.216 10.0.212.121
```

Sometimes you can't detail error data (n_errs and error codes). In particular, Windows does not determine adress causing echo with recv_errno=-3 "Destination Host Unreachable". If you ping more than 1 addr simultaneously, set ping_set_err := _False. You can try to remove this option.

Try:
1. cqos.ping_set_err := _True;
   Ping 1 IP, dead or alive,
   Ping 2 IP, one dead and one alive.

2. Ta_NoahsArkTest - Noahs Ark is a model how to place different species in one Ark for up to 40 days trip, noahs_ark.c

Two nodes are messaging with the complex structure of **NoahsArk**. STRUCT is the transmission unit. The following types can be contained in it:
1 byte: TaBool, TaOctet, TaChar
2 bytes: TaShort, TaWChar
4 bytes: TaLong, TaFloat
8 bytes: TaLongLong, TaDouble
Incapsulated structs as russian trio of horses "Troyka"
pointers to dynamically allocated/updated arrays of records (for example, troykas) or of simple types (for example, olives)

Naturally, the arrays of records can contain the pointers, as "MyString".

This record is serialized into buffer for sending. The destination node may have another packing algorithm and byte ordering.

Interface is determined by global variables of NoahsArk and TaTable types. Type definition macros TA_STRUCT() not only creates NoahsArk, but also saves string description in g_NoahsArk. C-Language does not have RTTI – runtime type information, required for TA performance. Therefore RTTI is created by g_NoahsArk information. Therefore TA works only with structures, defines via TA_STRUCT().

taParseToRTTI(g_NoahsArk, "Tests.INoahsArk", &g_rtti_NoahsArk, 0);

The NoahsArk linkage to table is realized in taAttachTable(&table, &g_rtti_NoahsArk, ...). According to RTTI CRC are calculated for each type as Message.msg_type, Message.iface_code. Crc for sender and receiver must match (except SOAP, that used soft match algorithm)!

The **Area** of visibility has name "NoahsArk" and number 69 (Noahs Ark is from Genesis 6-9). Two **Station** nodes are messaging upon initialized channels.

Start parameters are hosts and –e protocol. The important one is the number of host station (the knowledge of yourself) -n **index_host**. It must be set in one of the following: A. by asterisk ***** before host station IP; B. by option –n index_host.
Load 2 instances of noahs_ark on one local (127.0.0.1) computer as below (first for first console window, second - for second). First noahs_ark sets UDP at 127.0.0.1:7702. Second - sets at 127.0.0.1:7802. Taking into account that communication channel can be instantiated in several protocols simultaneously, for **first_addr** qos (7700 default) the following ports can be bound:
first_addr+1 in case of UDP multicast;
first_addr+2 in case of UDP;
first_addr+3 in case of TCP.

```
noahs_ark -e 2 *127.0.0.1 127.0.0.1:7800
noahs_ark -e 2 127.0.0.1 *127.0.0.1:7800
```

Pay attention to log. Check for sent and shown data. Do you see char arrays 'first', 'the second string'?

Try to change protocol:
2 - UDP,
4 - TCP, server only
8 - TCP, client only
12 - TCP client&server.
TCP is divided into 2, because of the frequent TCP use cases in which it only accepts connections from the client side.

Try to launch on the different computers.
Dont forget to set your proper IPs into command line!

```
noahs_ark -e 2 *10.0.220.39 10.0.201.216
noahs_ark -e 2 10.0.220.39 *10.0.201.216
```

Try to turn on tracing -t 63.
Tracing bits:
1 - RUNTIME cycle
2 - Transport level logging (socket operation, etc.)
4 - State of polling (data existence)
8 - Message formatting level
16 - Message details level
32 - All Errors

3. NodexTest - NODEs eXchange between N stations, nodex.c

Transparent Architecture is developed for realtime tasks also. The HW, OS and application software must meet realtime requirements in this case.

Three nodes with cycle "-c tp_cycle" each are messaging with statistics as the result.
The **must_recv** is expected from each 2 nodes. If a message is not delivered it increments **number_of_lost** counter. Those from the nodes, that can't see you increment **blind**. If a message is beyound the expectation time range, the **not_just_in_time** counter is incremented. In realtime systems such messages are discarded! If you need real-time communication, your must use realtime OS, precisious counters, calling Ta as kernel rpocess, nodes call synchronization.

The message format is created from NodeStatus record, messaging algorithms are implemented through TaTable global variables.

On new data message is processed by HandleMessage(...) routine.
“areaUpdate” program is called each cycle and sends data by taWriteTable. From the other side “printStatistics” program is attached to current station and it is updated 1 time per 10 seconds with statistics recalculation and its dialog update. areaUpdate attachement is to Area, it covers all the process. printStatistics is bound only to host station.

Try to launch 3 instances on different computers if possible (the only 2 comps are below). The cycle time is 100000 mcs, protocol is 12 - TCP.

```
nodex -e 12 -c 100000 *10.0.220.39 10.0.220.39:7800 10.0.201.216
nodex -e 12 -c 100000 10.0.220.39 *10.0.220.39:7800 10.0.201.216
nodex -e 12 -c 100000 10.0.220.39 10.0.220.39:7800 *10.0.201.216
```

The first parameter **deviation_abs** is absolute deviation of random value:
<cycle time> / <required cycle time> * 100%
This parameter reachs very high 50% in my computer.
! keep in mind that all applications must start within 10-seconds dead period for correct statistics.

Try to modify cycle time from 10000 mcs to 1000000 mcs. Fix the values of number_of_lost %, not_just_in_time % for nodes.
Try to set UDP protocol (Ta_Lib.ENABLE_UDP=2). It does not show significant number_of_lost % increase in comparison with TCP. The ultimate reason of losses is cycle time unstability.

As a result of testing you can see close to zero values of number_of_lost, but a significant amount of not_just_in_time. As a consequence of cycle time deviation.

4-u. RPSuExe - Rock-Paper-Scissors contest module for unicast, rps.c

RPS (Rock-Paper-Scissors) is a game known from the childhood. Two players by a command made a gesture which can be one of three (R,P,S). After this the results are compared and local winner is determined. If both plays are equal, the result is 0, the result equals 1, if R,S (rock against scissors) S,P (scissors against paper) P,R (paper against rock). Otherwise the result is -1. The first play is random, but later your can estimate the previous actions of opponent. There are even RPS world tournaments.

Computer programs tournament is hosted in www.rpscontest.com. Anybody can develop Python script and upload it to server. All the scripts are of public access for download. In “py” dir from rpscontest.com are placed. For Linux it can be compiled with Python, with library install and recompile, changing app in Makefile to Makefile.with_python. For Windows the example with Python was not tested.

Without Python all examples work well, but only simple embedded RPS algorithms are used: r(rock always), p(paper always), s(scissors always), rand(flat random select from r,p,s), brand(random, biased according to the last opponent answer).

The algorithm is as following. N tasks are started on one or several computers. Each task requires also a string algorithm name rpspgm_name, in which RPS is realized by function Evaluate (…,TaChar input, TaChar *output), which can make a play with previous opponent play as an input. Each server task plays with each 1000 times, so for N=5 tasks each task plays (N-1)*1000 = 4000 times. Well, the tasks are loaded. Task №0 broadcasts start signal after 10 seconds delay and call DoControl next.

```
taWaitTable(&g_tab_control, taGetTime()+START_DELAY);
taWriteTable(&g_tab_control, &g_control, TA_ADDR_ALL, _False);
taCallTable(&g_tab_control, DoControl, &my_sid_number);
```

All others are waiting for DoControl start signal.

```
taWaitTable(&g_tab_control, TA_WAIT_ANY_MESSAGE);
taCallTable(&g_tab_control, DoControl, &my_sid_number);
```

The subsequent actions are performed by each couple. The are divided into 2 parts: MakeYourChoice, DoCalculate. In the first part each couple makes a play programs[t.dst_oid].Evaluate(outputs[t.dst_oid].input, out); and sends a message to opponent, waiting for an answer which causes transition to second part:

```
taWriteTable(&g_tab_outputs[table->dst_oid],
        &g_outputs[table->dst_oid], TA_TO_ADDR(table->dst_oid), _True);
taWaitTable(&g_tab_outputs[table->dst_oid], TA_WAIT_ANY_MESSAGE);
taCallTable(&g_tab_outputs[table->dst_oid], DoCalculate, NULL);
```

There is a mutual waiting for answer, when ready for rendevoux the second part functions are called:

```
taWriteTable(&g_tab_outputs[table->dst_oid], &g_outputs[table->dst_oid],
        TA_TO_ADDR(table->dst_oid), _True);
taWaitTable(&g_tab_outputs[table->dst_oid], TA_WAIT_ANY_MESSAGE);
taCallTable(&g_tab_outputs[table->dst_oid], MakeYourChoice, NULL);
```

After mutual estimation MakeYourChoice is called if the number of plays is less than 1000. Otherwise print results and finish.

We deal with non realtime OS, so processing and network resources are not distributed uniformly. To compensate this the message timeouts QoS (Quality of Service) are set.

```
mqos.tp_timeout := 60000000;
mqos.tp_send_timeout := 30000000;
```

In this example there is unique format for name - hostname:name-of-script:port.
For Windows File rps_5.bat (code below, file in app\winapp\Release) starts a set of 5 tasks for execution. It takes several minutes to play all the games. So start batch from console to see output and use Task Manager to monitor 5 processes.

```
start /b rps.exe -n 0 -e 12 127.0.0.1:r:7700 127.0.0.1:p:7800 127.0.0.1:s:7900 127.0.0.1:rand:8000 127.0.0.1:brand:8100
start /b rps.exe -n 1 -e 12 127.0.0.1:r:7700 127.0.0.1:p:7800 127.0.0.1:s:7900 127.0.0.1:rand:8000 127.0.0.1:brand:8100
start /b rps.exe -n 2 -e 12 127.0.0.1:r:7700 127.0.0.1:p:7800 127.0.0.1:s:7900 127.0.0.1:rand:8000 127.0.0.1:brand:8100
start /b rps.exe -n 3 -e 12 127.0.0.1:r:7700 127.0.0.1:p:7800 127.0.0.1:s:7900 127.0.0.1:rand:8000 127.0.0.1:brand:8100
rps.exe -n 4 -e 12 127.0.0.1:r:7700 127.0.0.1:p:7800 127.0.0.1:s:7900 127.0.0.1:rand:8000 127.0.0.1:brand:8100
```

For Linux even without Python the same version is in rps_5 (file in app).

```
rps -n 0 -e 12 127.0.0.1:r:7700 127.0.0.1:p:7800 127.0.0.1:s:7900 127.0.0.1:rand:8000 127.0.0.1:brand:8100 &
rps -n 1 -e 12 127.0.0.1:r:7700 127.0.0.1:p:7800 127.0.0.1:s:7900 127.0.0.1:rand:8000 127.0.0.1:brand:8100 &
rps -n 2 -e 12 127.0.0.1:r:7700 127.0.0.1:p:7800 127.0.0.1:s:7900 127.0.0.1:rand:8000 127.0.0.1:brand:8100 &
rps -n 3 -e 12 127.0.0.1:r:7700 127.0.0.1:p:7800 127.0.0.1:s:7900 127.0.0.1:rand:8000 127.0.0.1:brand:8100 &
rps -n 4 -e 12 127.0.0.1:r:7700 127.0.0.1:p:7800 127.0.0.1:s:7900 127.0.0.1:rand:8000 127.0.0.1:brand:8100
```

Run it, wait for results: for each rps it should be: A. scope string, B. ndone string (if everything is within timeouts, last number of done must be ndone:4).
Try to increase number of tasks (up to now max is 32), until you get timeouts in the form of ***error.

4-m. RPSmExe - Rock-Paper-Scissors contest module for multicast, rpsm.c

Try RPS unicast version before. As each task plays 1000 times with the opponent, N tasks play (N-1)*1000 times. As each of 2 parts sends a message to opponent, each task sends 2*(N-1)*1000 messages, especially in first seconds are 2*(N-1) messages each cycle. With big N an "avalanche" of messages occur, causing network resourses overload.

Multicast-version is based upon one-to-many transmission, and those many are multicast group members. The main difference is that **only one** message is send each time cycle. This message contains a record sequence one for each station. 2 implementations are presented:

1. UDP multicast (protocol=1). You need preliminary tuning of your computer to add group address 239.255.0.1 (Administrative priviledge is required, Your IP insert instead of 10.0.220.39):

For Linux by superuser

//sbin/route add -host 239.255.0.1 dev ethN,

where N is interface number: 0, 1

For Windows

\windows\system32\route add 239.255.0.1 mask 255.255.255.255 10.0.220.39

2. SHM multicast (protocol=131072 или 020000H). Shared Memory communication is working within the single computer.

As usual for group messaging, the data transmission is performed without readiness check or acknowledgement messages.

```
taWriteTable(&g_tab_outputs, &g_outputSeq, TA_ADDR_ALL, _False);
```

If exchange is initialized, the data is broadcasted to all. OnMessage is called on each new data and according to the counters values of "dst_sid" station the proper procedure is selected. If opponent calculated previous play, MakeYourChoice is called to make next play. If opponent made his play, DoCalculate is called to calculate the result.

If the opponent response is skipped, it will come next cycle. Here is implemented ambigious messaging instead of guaranteed delivery. It's finished upon reaching 1000 times or on timeout.

For Windows file rpsm_5.bat (below) starts a set of 5 tasks for execution. It takes several minutes to play all the games. So start batch from console to see output and use Task Manager to monitor 5 processes.

```
start /b rpsm -n 0 -e 1 .:r .:p .:s .:rand .:brand
start /b rpsm -n 1 -e 1 .:r .:p .:s .:rand .:brand
start /b rpsm -n 2 -e 1 .:r .:p .:s .:rand .:brand
start /b rpsm -n 3 -e 1 .:r .:p .:s .:rand .:brand
rpsm -n 4 -e 1 .:r .:p .:s .:rand .:brand
```

Для Linux even without Python the same version is in rpsm_5.

```
rpsm -n 0 -e 1 .:r .:p .:s .:rand .:brand &
rpsm -n 1 -e 1 .:r .:p .:s .:rand .:brand &
rpsm -n 2 -e 1 .:r .:p .:s .:rand .:brand &
rpsm -n 3 -e 1 .:r .:p .:s .:rand .:brand &
rpsm -n 4 -e 1 .:r .:p .:s .:rand .:brand
```

Try the same for multicast Shared Memory communication. The option protocol=131072 (Ta_Lib.ENABLE_SHM_MCAST) must be set in config file.

5. S_Srch - Pattern Search, s_srch.c (m_srch.c) messaging server and c_srch.c messaging client.

Messaging server and client are introduced in this example. Server is implemented in Ta_S_Srch module and is a single thread working with up to MAX_STATIONS=5 clients. Server is bound to port 9000 and TCP protocol, it can reside beyound the firewall. Server also is bound to 127.0.0.1 IP, but due to the inactive qos check_host_ip it accepts connections with any addrs.

The data messaging is performed through 2 data structures: SearchQuery and SearchResults. The first must contain queries, second - results for these queries. The server receives " SearchQuery" queries and performs BM-Search, sending client found in " SearchResults". Disk operations are deliberately not used (it receives all data from the clients), which explains why is the single thread architecture the best one for this case. This is the most effective solution without blocking, it is only possible, because Ta supports non-blocking communication.

Server program Ta_S_Srch calls ProcessSearchQuery procedure for each client call,

```
taRepeatTable(&g_tab_query_, ProcessSearchQuery, NULL);
```

which realizes the read-process-write sequence:

```
sq = (SearchQuery *) table->recent_data;
...
taWriteTable(&g_tab_results_, &g_results_, TA_TO_ADDR(table->message.src_sid), _True);
```

Client application Ta_C_Srch calls DoNextQuery procedure with next query algorithm. In case of previous query data:

```
sr = (SearchResults *) table->recent_data; With data printing ...
```

After next portion of strings next server call is performed:

```
taWriteTable(&g_tab_query, &g_query, TA_TO_ADDR(1-my_sid_number), _True);
taWaitTable(&g_tab_results, TA_WAIT_ANY_MESSAGE);
taCallTable(&g_tab_results, DoNextQuery, NULL);
```

It finishes at the end-of-file, but in case of "repeat" option the process is repeated.

Parameters are set in config file srch.cfg. Function args = taArgsFromConfig("srch.cfg") is used instead of usual for command line argc, argv.

Try to run server and 2 clients from separate console windows (copy rpsm.c to your directory):

s_srch

c_srch -repeat 1 Value rpsm.c

c_srch -repeat 1 talnit rpsm.c

Verify the client apps are working independently.

Try to increase number of clients to four, five. Verify that the fifth client can't receive connect.

Try these actions on different computers. The remote client must have -hostname 10.0.220.39 option set (with your IP).

Server **dll/so** can be hosted not only by special tarun program, but by client application. In this case the communication is performed inside the same process. The -protocol option must be set to 65536 (Ta_Lib.ENABLE_LIB* = 10000H). It can be used for Oberon server and C client and vise versa.
For convenience create C folder under working directory and copy to it C-examples executives (s_srch.exe s_srch.dll c_srch.exe).

Try client C-application c_srch with server library s_srch:
c_srch -protocol 65536 -name s_srch -repeat 0 -t 32 talnit rpsm.c

Client application loads library and works with it in its internal address space.

Server library can be developed in different language. If you have Oberon/BlackBox, try client C-application c_srch.exe with server-lib S_SRCH.dll in Oberon:
C\c_srch -protocol 65536 -name S_SRCH Box StartupBlackBox.vbs
With -repeat unset only one query result must be printed to console.

This serialization in TA is optimized for network communication, so the structures are packed and byte ordering is set to network.

5-F. SFSrch - File Pattern Search: sfsrch.c (mfsrch.c) single-threaded server, stsrch.c (mtsrch.c) multi-threaded server and cfsrch.c client.

Here is the client-server application instead of messaging server. Messaging server has a serious drawback - it can't achieve data integrity. The query data are separated from results data (in structures SearchQuery and SearchResults), and there is no guarantee, that the server results correspond the expected client query.

Client-server application has the same data structure for queries and responses and the mechanisms of getting either the response or error message.

```
TA_STRUCT(SearchResults,
        TaChar pattern[128];
        TaChar file_name[128];
        TaLong start_pos;
        TaLong end_pos;
        TaLong request_number;
        FoundString *found;
)
```

The SearchResults structure contains request fields (pattern, file_name, start_pos) and response fields (end_pos, found). Response fields correspond to request. The request_number field is a server side query counter.

Server is implemented in sfsrch.c module, number of clients MAX_STATIONS-1 = 4. Port number 9000, host 127.0.0.1 and protocol TCP.

Server reads data from the file and performs search. It takes time, other clients are waiting. Upon writing response server gets ready for next request processing.

Server program is sfsrch.c ProcessSearchQuery is based upon:

```
taRepeatTable(&g_tab_search, ProcessSearchQuery, NULL);
```

which implement read-process-write:

```
sr = (SearchResults *) table->recent_data;
...
taWriteTable(table, sr, TA_TO_ADDR(table->message.src_sid), _True);
```

Client app cfsrch.c calls DoNextQuery procedure as start moment,

```
taCallTable(&g_tab_search, DoNextQuery, (void *)1);
```

queries are sent with the help of Request method, consisting of Write and responce callback. Request callback call means that response contains recent request number. The next call to server:

```
taRequestTable(&g_tab_search, sr, TA_TO_ADDR(1-my_sid_number), _True, DoNextQuery, NULL)
```

Finalization occurs by the end of file if repeat option is not set.

Parameters - in config file srch.cfg.

Run server application from console (run server app before clients).

Sfsrch

Run client apps from console windows:

cfsrch -repeat 0 a rpsm.c

cfsrch -repeat 1 talnit rpsm.c

Try to start 4 clients with -repeat 1. Pay attention that clients are waiting their queue due to console output. Stop server app, check for #12 error concerning the table.

Try to do the same on different computers (-hostname) and with remote applications.

Multi-threaded version starts several threads, each has its own update cycle and TA-context TaThreadData. In main-program multi-threaded initialization must be set.

```
taInitThreadsData(&taqos, &area);
SearchServerInit(&area);
taThreadsStart(&area, &my_sid_number);
```

In the SearchServerInit section each thread is associated with a TaStation, TaTable (or several tables), SearchResults variable and additional global variables, like g_request_numbers. Different thread data should not intersect.

Inside TA-library the data I/O is performed through the main thread only. Inter-thread communication is limited to copying bytes from area I/O buffers. Send buffer is protected by send_mutex when copying from the worker to main thread. Receive buffer is protected by recv_mutex when copying from main to worker thread. For Windows critical sections are implemented instead of mutexes.

Multi-threaded server has the same interface as single-threaded:
stsrch
Compare **cfsrch** client behavior with the multi-threaded server.

5-P. File Pattern Search, spsrch.c publisher and cpsrch.c subscriber.

Publish-Subscribe Model assumes periodical sending messages to the group of recipients. It is realized through multicast transport protocol. 5-P example uses protocol=1(UDP Multicast) end protocol=131072 (SHM Multicast – shared memory within single host).

In the following example Publisher opens 5 channels, #100 - for permanent publication and #101-104 - for publication by the subscriber queries.

Permanent publication is defined by "pattern" and "file_name" config parameters:

```
strncpy(g_search[i].pattern, pattern, MAX_STRING_LENGTH);
strncpy(g_search[i].file_name, file_name, MAX_STRING_LENGTH);
taPublishTable(&g_tab_search[i],_True, ProcessSearchQuery, NULL);
```

Publication by queries is defined by "pattern" and "file_name" from queries.

```
taPublishTable(&g_tab_search[i],_False, ProcessSearchQuery, NULL);
```

Subscription may have limited or unlimited time. In the example permanent subscription is unlimited, but subscription by query is finished at the end of file by taClearTable command.

```
} else if (g_search[ind].end_pos < 0) {
        taClearTable(table);
        return;
```

A query for subscription start is created in client. From the client side it simply joins the existing subscription, as in the case of #100:

```
taSubscribeTable(&g_tab_search, _False, DoNextQuery, NULL);
```

or controls the subscription, aka sends a query and receives the correspondent data:

```
taWriteTable(&g_tab_search, &g_search, TA_TO_ADDR(1-my_sid_number), _True);
taSubscribeTable(&g_tab_search, _True, DoNextQuery, NULL);
```

By the way other clients can join such a subscription. If it is changed (by another query on the same channel), RESPONSE_MISMATCH error is generated, which means that receiving data does not match query data. Client can finish calling Unsubscribe with optional ability to send publisher message to cancel transmitting.

Run publisher and 2 subscribers from separate console windows:
spsrch
cpsrch
cpsrch

Verify publisher prints exist. Verify subscribers get the same data according to pattern and file_name.
Copy rpsm.c from distribution to working directory.
Run the third subscriber with another file_name and pattern parameters (in the example "a" "rpsm.c").
cpsrch -dst_oid 101 a rpsm.c
Verify channel #101 subscription working. Run again the third client and simultaneously - the forth by:
cpsrch -dst_oid 101
Verify the data is the same for both.

Try to subscribe to #101 the forth (without stopping the third) with params:
cpsrch -dst_oid 101 b rpsm.c
Verify the third got -14 (RESPONSE_MISMATCH) error code.

Naturally, you can subscribe for unicast protocols. The channels in this case should be associated with individual, not with group.

6. Global Weather - SOAP/HTTP: sgweather.c web-service and cgweather.c clients.

Only Client-Server model is implemented for WAN. The following queries are supported:
- HTTP GET,
- HTTP POST,
- SOAP 1.1,
- SOAP 1.2.

Web-service implementation is based on http://www.webservicex.net/globalweather.asmx?op=GetWeather. It is a web service example and client HTML-pages to work with.

In this example TA introduce its clients and server. They also implement GetWeather and GetCitiesByCountry interfaces. TA-clients can work with remote www.webservicex.net, similarly your browser can work with your server. Saved and modified pages GetWeather.htm, GetCitiesByCountry.htm are supplied.

Run server app from console by the following command (first server app, clients - after).
sgweather -port 8080
Run your browser at the same computer with the page GetWeather.htm. Enter Country and City, submit for response. In the page the query was modified:

action='http://127.0.0.1:8080/globalweather.asmx/GetWeather' method="POST".

Edit it if you use another host or port.
Run browser with the page GetCitiesByCountry.htm, submit again. You have tested HTTP GET.

Run you client app from console with -hostname, if remote. In the first case GetCitiesByCountry query must be done, in the second - GetWeather.
cgweather -trace_level 26 -port 8080 -soap GET russia
cgweather -port 8080 -soap GET russia moscow
Run client app with options -soap POST, -soap 11, -soap 12. Verify its performance with SOAP 1.1, SOAP 1.2 protocols .

Try to request remote server (it is bound to port 80):
cgweather -hostname www.webserviceX.NET -soap 11 Japan
cgweather -hostname www.webserviceX.NET -soap 12 Japan Tokyo

Try to request server with params Japan Osaka, Russia Moscow, Russia Vassuki. You must get "No Data" in the last case.