Transparent Architecture - коммуникационное ПО для построения распределенных систем.

Описание к примерам

Что это такое:

1. TA позволяет строить распределенные системы на основе моделей клиент-сервер и издатель-подписчик, а также системы обмена сообщений низкого уровня. TA может заменить клиент-серверные технологии CORBA, DCOM и DDS-технологию издатель-подписчик.

2. TA построена исключительно на неблокирующем режиме сетевого взаимодействия, поэтому не страдает подвисаниями, характерными для COM (да и CORBA) и не требует отдельных тредов, блокируемых во время ожидание прихода данных.

3. TA имеет тип протокола обмена SRPS для локальных сетей и SOAP/HTTP - для глобальных. На физическом уровне в локальных сетях возможно использование протоколов: UDP multicast, UDP, TCP, ICMP, Shared Memory. Возможен обмен через DLL, а в C-Unix версии - через SO, Unix Domain Sockets и MSGQ. Множество поддерживаемых протоколов может расширяться.

4. TA - пилотный проект. Есть 2 варианта: TA для BlackBox и TA для C. Предлагается Вам TA для C протестированная с Linux и Windows. Относящееся только к Linux сборке помечено коричневым цветом. Относящееся только к Windows сборке помечено синим.

5. TA распространяется в исходных текстах по лицензии LGPL.

Как собрать:

TA состоит из библиотеки, собираемой в src и примеров, собираемых в app.

Для Linux
tar xvzf ta1.tgz
cd ta1/
make

Для Windows
Распаковать ta1.zip
VC++ 2010
Открыть ta1s.sln из ta1\src
Собрать библиотеку ta1s.dll
Скопировать ta1s.dll C:\Windows\
Открыть winapp.sln из ta1\app\winapp
Собрать примеры winapp
Бинарники лежат в ta\winapp\Release

Как с этим работать:
1. Для Linux
TA требует для работы доступ к библиотеке libta.so. Для этого нужно прописать путь к src (если у Вас в каталоге ~/ta1/src):
export LD_LIBRARY_PATH=~/ta1/src:$ LD_LIBRARY_PATH

Запускайте все из app

Для Windows
TA требует для работы доступ к библиотеке ta1s.dll, которую нужно скопировать, скажем, в C:\Windows или в winapp.

Запускайте все из winapp

2. Использование TA предполагает начальные сведения по конфигурированию локальных сетей.

Примеры:
1.**Ping Test** Тест протокола ICMP
2.**Noahs Ark Test** Каждой твари по паре в одной посылке
3.**Nodex Test** Обмен сообщений между точками
3-A.**Nodex Exe** Сервер Обмена сообщений
4-u.**RPS Unicast Exe** Соревнование по Камень-Ножницы-Бумага, unicast
4-m.**RPS Multicast Exe** Соревнование по Камень-Ножницы-Бумага, multicast
5.**Srch** Поиск по шаблону - Сервер обмена сообщениями
5-F.**FSrch** Поиск в файле по шаблону - Клиент-серверное приложение
5-P.**PSrch** Поиск в файле по шаблону - модель Издатель-Подписчик
6.**GWeather** Обмен по SOAP/HTTP, веб-клиент и веб-сервис

С уважением, автор ПО TA
Дагаев Дмитрий Викторович dvdagaev@yahoo.com

1. PingTest - реализация PING, файл pings.c

Прежде чем устанавливать связь между компьютерами, проверим видимость по сети Ethernet удаленных абонентов. В примере ниже предлагается реализация протокола ICMP (описание PING можно посмотреть http://www.ping127001.com/pingpage.htm)

Для любой задачи Та определяется виртуальная область видимости **Area** с именем "ping" и номером 1983. Область разбита на станции **Station**, каждая из которых состоит из 1-8 каналов передачи **Channel**, между которыми курсируют сообщения **Message**.
Конфигурирование осуществляется через параметры QoS (Quality-of-Service) aqos/sqos/cqos, относящиеся к Area, Station, Channel.

Для пингования адресов 10.0.201.216 10.0.201.206 с данного компьютера устанавливают 3 станции с IP-адресами 127.0.0.1 10.0.201.216 10.0.201.206. Среди протоколов маской устанавливается TA_ENABLE_ICMP. Это означает, что инициализируется только station.channels[0] протоколом ICMP. Период вызова программы stationUpdate задается в “-i interval” в секундах, 1 раз в секунду она шлет сообщение taSetMessage и проверяет коды возврата. После “-c count” циклов печатается таблица результатов.
Анализируя для 0-канала данной станции число переданных и полученных байт complete_send_rc, complete_recv_rc, а также коды ошибок send_errno, recv_errno - получим результат пингования.
Параметры запуска: адреса, счетчик count и время цикла. Можно задавать и имена хостов в рамках видимости DNS.

Имейте в виду, что требуется логин с правами администратора для возможности работы с ICMP и с Raw Sockets вообще.

Для Linux суперюзером: chown root pings;chmod 777 pings;chmod u+s pings
Для Windows логин с правами администратора

IP-адреса, разумеется, у Вас другие! Здесь и далее **подставляйте Ваши адреса в строку запуска** коммандера.

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

Не всегда возможно получить детализацию ошибок (n_errs и коды ошибок). В частности, для Windows не определяется адрес, для которого пришло эхо с ошибкой recv_errno=-3 "Destination Host Unreachable". Если одновременно пингуется более 1 адреса, устанавливается ping_set_err := _False. Можете поэкспериментировать, убрав эту опцию.

Попробуйте:
1. cqos.ping_set_err := _True;
   Пингуйте 1 адрес, живой и не живой,
   Пингуйте 2 адреса, 1 живой и один не живой.

2. NoahsArkTest - Ноев Ковчев как модель размещения тварей божиих разных видов в ограниченном пространстве судна, noahs_ark.c

Два узла обмениваются сообщения сложной структуры **NoahsArk**. Единица обмена - STRUCT. В состав могут входить типы:
1 байт: TaBool, TaOctet, TaChar
2 байта: TaShort, TaWChar
4 байта: TaLong, TaFloat
8 байт: TaLongLong, TaDouble
Вложенные структуры, как русская тройка лошадей Troyka
Указатели на динамически обновляемые массивы записей (например, troykas) или простых типов (например, olives)
Массивы записей эти сами могут содержать указатели, например, MyString.

Эта структура сериализуется в буфер передачи. На узле-приемнике может быть другой порядок байт и алгоритм упаковки.

Интерфейсно это представляет собой глобальные переменные NoahsArk и глобальные переменные TaTable. При определении типа макрос TA_STRUCT() не только создает NoahsArk, но и сохраняет строковое описание структуры в g_NoahsArk. Язык C не имеет RTTI – runtime type information, необходимой для работы TA. Поэтому RTTI создается по информации, содержащейся в g_NoahsArk. Поэтому TA работает только со структурами, определяемыми через TA_STRUCT().
taParseToRTTI(g_NoahsArk, "Tests.INoahsArk", &g_rtti_NoahsArk, 0);
Привязка структуры NoahsArk к таблице реализуется в функции taAttachTable(&table, &g_rtti_NoahsArk, ...). По RTTI также вычисляются контрольные суммы каждого типа Message.msg_type, Message.iface_code. Контрольные суммы посылающей и принимающей сторон должны совпадать!

Область видимости **Area** имеет имя "NoahsArk" и номер 69 (Бытие 6-9). Два узла **Station** обмениваются сообщениями по каналам, которые инициализированы.

Параметры запуска: адреса и –e протокол. Важный параметр -n номер домашней станции **index_host**. Его нужно устанавливать обязательно одним из способов: A. отметкой ***** перед адресом/портом домашней станции; B. Опцией –n index_host.
Загрузите 2 экземпляра noahs_ark на одном локальном (127.0.0.1) компьютере см. ниже (первый в одном консольном окне – второй во стором). Первый noahs_ark установит UDP на 127.0.0.1:7702. Второй коммандер установит 127.0.0.1:7802. Учитывая, что связь может устанавливаться по нескольким протоколам одновременно, для начального адреса **first_addr** (по умолчанию 7700) будут устанавливаться порты:
first_addr+1 для UDP multicast;
first_addr+2 для UDP;
first_addr+3 для 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
```

Обратите внимание на консоли. Проверьте соответствие посылаемой и выдаваемой информации. Есть ли массивы строк 'first', 'the second string'?

Попробуйте поменять протокол:
2 - UDP,
4 - TCP, только сервер
8 - TCP, только клиент
12 - TCP клиент и сервер.
TCP так разделен, т.к. чаще всего TCP сервер только принимает (accept) соединения, а не ищет связи с клиентами на стороне.

Попробуйте запустить на разных компьютерах.
Не забудьте подставлять правильные Ваши IP-адреса в командную строку!

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

Попробуйте включить опцию трассировки –t 63.
Биты трассировки:
1 - Цикл RUNTIME
2 - Сообщения траспортного уровня (сокеты и пр.)
4 - Состояние поллинга (есть, нет данных)
8 - Уровень сообщений
16 - Детализация сообщений
32 - Все ошибки

3. NodexTest - NODEs eXchange - обмен сообщений между N точками, nodex.c

Transparent Architecture разрабатывался и для того, чтобы использовать его в задачах реального времени. Если, конечно, оборудование, ОС и вызывающее ПО позволяет.

Три узла, каждый с циклом –c tp_cycle, обмениваются сообщениями, после чего приводится статистика.
От каждых двух узлов должно было прийти **must_recv** сообщений. Те, что не пришли, попали в потерянные **number_of_lost**. Те, что пришли от узлов, не видящих Вас, попали в **blind**. Если сообщение пришло раньше или позже ожидаемого срока, увеличивается счетчик **not_just_in_time**. В realtime системах такие сообщения отбрасываются! Если требуется real-time связь, нужны ОС РВ, прецизионные счетчики, запуск Та как процесс ядра, синхронизация такта между узлами.

Формат сообщений представляет собой структуру NodeStatus, механизм работает через глобальные переменные TaTable.

По наличию новых данных программе передается сообщение и обрабатывается хэндлером HandleMessage(...).
Программа areaUpdate вызывается в каждом цикле и пересылает в сеть taWriteTable. А программа printStatistics, относящаяся к текущей станции, обновляет раз в 10 секунд статистику и ее отображение в окне диалога. Привязка программы areaUpdate относится к области Area, она захватывает весь процесс. Привязка программы printStatistics относится только к домашней станции.

Попробуйте запустить 3 экземпляра по-возможности на разных компьютерах (ниже в примере - только 2). Время цикла 100000 мкс, протокол 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
```

Первый параметр **deviation_abs** есть абсолютное отклонение случайной величины
<время цикла> / <требуемое время цикла> * 100%
Этот параметр у меня 50%, что много.
! когда запускаете, имейте в виду 10-секундную паузу, в течении которой все приложения должны быть запущены.

Попробуйте изменять время цикла от 10000 мкс до 1000000. Зафиксируйте показания number_of_lost %, not_just_in_time % для узлов.
Попробуйте установить протокол UDP (Ta_Lib.ENABLE_UDP=2). Потерь number_of_lost % больше (по сравнению с TCP) не станет. Потери от неточного времени цикла значительно превышают потери из-за UDP.

Протестировав, вы увидите, что потерь данных number_of_lost на таких мало, зато есть потери not_just_in_time. Это - следствие большой девиации времени цикла.

4-u. RPSuExe - модуль для игры Камень-Ножницы-Бумага, протокол unicast, rps.c

RPS (Rock-Paper-Scissors) или Камень-Ножницы-Бумага - это игра, знакомая с детства. Двое играющих совершают постановку (R,P,S). После этого сравнивают результаты и определяют победившего. Если постановки одинаковы, результат 0, результат 1, если R,S (камень ломает ножницы) S,P (ножницы режут бумагу) P,R (бумага заворачивает камень). Иначе -1. Первый ход делается случайно, но в последующих Вы имеете возможность сделать ходы, оценивая предыдущие действия противника. В мире даже проводятся соревнования по RPS.

Соревнования компьютерных программ представлены на www.rpscontest.com . Любой может написать свой скрипт на Питоне и забросить к ним на сервер. Скрипты эти доступны, из можно скачать и анализировать. В каталоге ру расположены скрипты от rpscontest.com. Для Linux можно собрать с Питоном, установив библиотеку и пересобрав, заменив в app Makefile на Makefile.with_python. Для Windows пример с Питоном не проверялся.

Без Питона все примеры работают, но функционируют только простые встроенные алгоритмы RPS: r(всегда камень), p(всегда бумага), s(всегда ножницы), rand(случайный выбор), brand(случайный выбор со смещением, в зависимости от предыдущего ответа оппонента).

Алгоритм работы следующий. Загружаются N задач на одном или нескольких компьютерах. Каждой задаче помимо домашней станции задается еще и имя алгоритма rpspgm_name, который реализует RPS через функцию Evaluate (…,TaChar input, TaChar *output), которая делает постановку, получая на вход предыдущее значение оппонента. Каждая серверная задача играет с каждой 1000 раз, т.е. для N=5 задач каждая из них играет (N-1)*1000 = 4000 раз. Итак, задачи загружены. Задача №0 выдает всем сигнал старта через 10 секунд после загрузки и запускает программу старта DoControl.

```
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);
```

Все остальные ожидают сигнала, чтобы тоже запустить DoControl.

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

Дальнейшие действия выполняются попарно. Они разбиты на 2 части: MakeYourChoice, DoCalculate. В первой каждая пара играющих делает постановку Evaluate(table, table->dst_oid, g_outputs[table->dst_oid].input,&g_outputs[table->dst_oid].output); и отправляет оппоненту, ожидая от него ответа, после чего переходит к функции второй части:

```
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);
```

Ожидание ответа взаимно, по мере готовности к рандеву выполняются функции 2 части:

```
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);
```

После взаимного подсчета переходим снова к MakeYourChoice, если число ходов меньше 1000. В противном случае печатаем результаты.

Поскольку серверных приложений может быть много и распределение между ними ресурсов может быть весьма неравномерно (это задачи ОС реального времени), введены таймауты через механизм QoS (Quality of Service) на передачу сообщений.

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

В данном примере свой формат имени - имя-хоста:имя-скрипта:номер-порта.
Для Windows файл rps_5.bat (ниже код, файл в app\winapp\Release) стартует 5 процессов. Несколько минут требуется для того, чтобы провести все игры. Запустите с консоли и используйте Task Manager для мониторинга 5 процессов.

```
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
```

Для Linux даже без Python та же версия - в rps_5 (файл в 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
```

Запустите дождитесь результатов: для каждого rps должны быть: A. строка результатов scope, B. строка ndone (если все уложилось в пределах таймаутов, то ndone:4).

Попробуйте увеличивать число задач (пока 32 - максимально), пока не появятся таймауты в виде ***error.

4-m. RPSmExe - модуль для игры Камень-Ножницы-Бумага, протокол multicast, rpsm.c

Переходите к данному разделу, испытав сначала unicast-версию. Поскольку каждая задача играет с каждым оппонентом 1000 раз, то N задач играют (N-1)*1000 раз. Поскольку каждая из 2-х частей отправляет сообщение оппоненту, то каждая задача отправляет 2*(N-1)*1000 сообщений, причем в первые секунды по 2*(N-1) сообщений в каждом временном цикле. При большом числе N возникает т.н. "лавина" сообщений, сильно загружающая сетевые ресурсы.

Мультикаст-версия основана на передаче данных один-ко-многим, причем эти многие входят в мультикастовую группу. Существенное отличие в том, что отправляется **одно** сообщение в каждом временном цикле. Это сообщение может содержать последовательность структур для каждого абонента. Предлагается 2 реализации:

1. UDP multicast (protocol=1). Нужна предварительная настройка Вашего компьютера, добавляющая групповой адрес 239.255.0.1 (Нужны права Администратора, Ваш IP вставьте вместо 10.0.220.39):

Для Linux суперюзером

/sbin/route add -host 239.255.0.1 dev ethN,

где N – номер интерфейса: 0, 1

Для 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). Механизм общей памяти (Shared Memory) работает, разумеется, в рамках одного компьютера.

Поскольку посылка передается группе адресатов, то передача данных происходит независимо от готовности или подтверждения получения предыдущей посылки.

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

Если обмен инициализирован, данные посылаются всем адресатам. При получении данных вызывается OnMessage и, в зависимости от значений счетчиков абонента dst_sid вызывается процедура. Если оппонент посчитал предыдущий ход, вызывается MakeYourChoice - сделайте следующий ход. Если оппонент сделал свой ход, вызывается DoCalculate - посчитайте результат.

Если ответ оппонента пропущен, он придет в следующем цикле. Здесь работает механизм повтора сообщений вместо гарантированной доставки. Завершение работы происходит по достижению 1000 партий или по таймауту.

Для Windows файл rpsm_5.bat (ниже) стартует 5 процессов. Несколько минут требуется для того, чтобы провести все игры. Запустите с консоли и используйте Task Manager для мониторинга 5 процессов.

```
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 та же версия - в 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
```

Попробуйте те же действия для мультикастового обмена через общую память - Shared Memory. В конфигурационный файл нужно внести протокол 131072 (Ta_Lib.ENABLE_SHM_MCAST).

5. S_Srch - Поиск по шаблону, s_srch.c (m_srch.c) сервер и c_srch.c клиент.

Предлагается сервер обмена сообщениями и клиент к нему. Сервер реализован в модуле Ta_S_Srch и представляет собой отдельный процесс, который работает одновременно с запросами до MAX_STATIONS=5 клиентов. Сервер привязан к адресу порта 9000 и протоколу TCP, он может находиться за файрволом. Сервер также привязан к адресу 127.0.0.1, при этом сброшенная опция check_host_ip принимает соединения к разным адресам.

Обмен осуществляется через 2 структуры данных: SearchQuery и SearchResults. Первая должна содержать запросы, вторая - ответы на эти запросы. Сервер получает запросы query и осуществляет алгоритм БМ-поиска, передавая клиенту найденные куски в results. На сервере специально не используются операции обращения к диску (данные приходят от клиентов), что позволяет выбрать однопоточную архитектуру сервера для обработки запросов от всех клиентов. Это - наиболее эффективное решение без блокировок, возможное только при неблокирующем сетевом взаимодействии Ta.

Серверная программа Ta_S_Srch для каждого запроса клиента вызывает функцию ProcessSearchQuery,

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

которая осуществляет процесс чтение-обработка-запись:

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

Клиентское приложение Ta_C_Srch вызывает процедуру DoNextQuery с алгоритмом очередного запроса. Если есть данные предыдущего запроса, то:

```
sr = (SearchResults *) table->recent_data; Далее печать данных.
```

После чтения очередных строк из файла производится очередное обращение к серверу:

```
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);
```

Окончание происходит по концу файла, но, если задан флаг repeat, то процесс повторяется.

Параметры запуска - в конфигурационном файле srch.cfg. Вместо обычных argc, argv используется разбор конфигурационного файла args = taArgsFromConfig("srch.cfg").

Попробуйте запустить серверное приложение и 2 клиентских (скопируйте rpsm.c в ваш каталог):

s_srch

c_srch -repeat 1 Value rpsm.c

c_srch -repeat 1 talnit rpsm.c

Убедитесь, что клиентские приложения работают независимо друг от друга.

Попробуйте увеличить число клиентов до четырех, пяти. Убедитесь, что пятый клиент не получает соединения.

Попробуйте произвести те же действия на разных компьютерах. Удаленный клиент должен содержать опцию вида -hostname 10.0.220.39

Серверное приложения в виде **dll/so** может загружаться непосредственно клиентским приложением. При этом обмен происходит внутри единого процесса и протокол обмена нужно установить Ta_Lib.ENABLE_LIB* = 10000H (опция -protocol 65536).

Попробуйте клиентское C-приложение c_srch с библиотекой-сервером s_srch:
c_srch -protocol 65536 -name s_srch -repeat 0 -t 32 talnit rpsm.c

Клиентское приложение загружает библиотеку s_srch в свое адресное пространство и работает с ней через память приложения.

Библиотека может быть написана на другом языке. Если у Вас есть Оберон/BlackBox, попробуйте клиентское C-приложение c_srch.exe с библиотекой-сервером S_SRCH.dll на Обероне:
C\c_srch -protocol 65536 -name S_SRCH Box StartupBlackBox.vbs
При сброшенном флаге repeat на консоль будет выдача результатов одного запроса.

Конвертация данных в TA предполагает сетевой обмен между компьютерами, поэтому структуры преобразуются в последовательности байт еще и с учетом сетевого порядка байт (network byte ordering).

5-F. SFSrch - Поиск в файле по шаблону: sfsrch.c (mfsrch.c) однопоточный сервер, stsrch.c (mtsrch.c) многопоточный сервер и cfsrch.c клиент.

Предлагается клиент-серверное приложение вместо сервера обмена сообщений. У сервера обмена сообщений есть существенный недостаток - отсутствие целостности данных. Данные запроса от данных ответа разделены (в структурах SearchQuery и SearchResults), а также нет гарантии, что результаты полученные позже от сервера, относятся к нужному запросу.

Клиент-серверное приложение имеет общую структуру данных для запросов и ответов, а также механизмы, гарантирующие либо ответ на запрос, либо сообщение об ошибке.

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

В структуре SearchResults есть поля как запроса (pattern, file_name, start_pos), так и ответа (end_pos, found). Поля ответа соответствуют запросу. Поле request_number - счетчик запросов на сервере.

Сервер реализован в модуле sfsrch.c, число клиентов MAX_STATIONS-1 = 4. Порт 9000, хост 127.0.0.1 и протокол TCP.

Сервер читает данные из файла на диске и осуществляет поиск. Это занимает время, другие клиенты в ожидании. Передав данные, сервер переходит к обработке следующего запроса.

Серверная программа в sfsrch.c ProcessSearchQuery:

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

которая осуществляет процесс чтение-обработка-запись:

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

Клиентское приложение cfsrch.c вызывает процедуру DoNextQuery в момент старта,

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

а запросы отправляются с помощью вызова Request, состоящего из вызова Write и ответа, гарантирующего содержание в данных ответа номера последнего запроса. Очередное обращение к серверу:

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

Окончание происходит по признаку конца файла, но, если задан флаг repeat, то процесс повторяется.

Параметры запуска - в конфигурационном файле srch.cfg.

Запуск серверного приложения из консоли командой (запускайте сначала серверное приложение, затем - клиентские).

sfsrch

Запуск клиентских приложений с консольных окон:

cfsrch -repeat 0 a rpsm.c

cfsrch -repeat 1 talnit rpsm.c

Попробуйте запустить 4 клиента с -repeat 1. Обратите внимание, что клиенты ожидают друг друга, что заметно по консольному выводу. Остановите серверное приложение, убедитесь в наличии ошибки # 12, относящейся к таблице.

Попробуйте произвести те же действия на разных компьютерах (-hostname) и с удаленными приложениями.

Многопоточный вариант стартует несколько потоков, каждый имеет свой цикл обновления и TA-контекст TaThreadData. В main-программе задается многопоточная инициализация.

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

В секции инициализации SearchServerInit каждому потоку поставлена в соответствие станция (station), таблица (или несколько) переменные SearchResults и какие-то глобальные переменные, как g_request_numbers. Данные от разных потоков не должны пересекаться.

Внутри TA-библиотеки ввод/вывод данных осуществляется только через основной поток. Взаимодействие между потоками сведено к копированию байт из буферов ввода/вывода. Буфер вывода во время копирования из потока обработки в основной защищен посредством send_mutex, буфер ввода во время копирования из основного в поток обработки посредством recv_mutex. Для Windows мьютексы заменены на критические секции.

Многопоточный сервер имеет тот же интерфейс, что и однопоточный:
stsrch
Сравните поведение клиентов **cfsrch** при работе с многопоточным сервером.

5-P. Поиск в файле по шаблону на основе Публикации-Подписки, spsrch.c издатель и cpsrch.c подписчик.

Модель издатель-подписчик предполагает публикацию данных группе абонентов-подписчиков. Это реализуется через мультикастовый протокол обмена. Пример 5-P использует протоколы обмена 1 (UDP Multicast), 131072 (SHM Multicast – через shared memory в рамках одного компьютера).

В примере Издатель открывает 5 каналов, #100 - для перманентной публикации и #101-104 - для публикации по запросам подписчиков.

Перманентная публикация определяется конфигурационными параметрами шаблона и файла:

```
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);
```

Публикация по запросам определяется параметрами (шаблон и файл) запросов.

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

Подписка может иметь ограниченное время, а может быть неограниченной. В примере перманентная подписка - неограниченна, а подписка по запросу завершается по концу файла командой taClearTable.

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

Запрос на старт подписки издателя формируется в клиенте. Со своей стороны клиент либо присоединяется к существующей подписке, как в случай с #100:

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

либо управляет подпиской, т.е. делает запрос и принимает относящиеся к нему данные:

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

При этом другие клиенты могут присоединиться к такой подписке. Если подписка изменилась (например, запросом другого подписчика на тот же канал), формируется ошибка RESPONSE_MISMATCH, означающая, что данные запроса не соответствуют данным подписки. Клиент может завершить подписку командой Unsubscribe с передачей запроса на останов издателю (при необходимости).

Запустите приложение-издателя и двух подписчиков с консольных окон:

spsrch

cpsrch

cpsrch

Убедитесь, что идет печать издателя программы. Убедитесь, что приложения-подписчики получают одинаковые данные в соответствии с параметрами pattern, file_name.

Скопируйте "rpsm.c" из дистрибутива в рабочий каталог.

Запустите третьего подписчика с параметрами другого файла и шаблона (для примера "a" "rpsm.c").

cpsrch -dst_oid 101 a rpsm.c

Убедитесь, что по каналу 101 пошла подписка в соответствие с запросом. Запустите еще раз третьего подписчика (желательно, чтоб он выдал несколько страниц) и одновременно с ним - четвертого:

cpsrch -dst_oid 101

Убедитесь, что оба подписчика получают одни и те же данные.

Попробуйте запустить на 101 канал одновременно с третьим четвертого подписчика со своими параметрами

cpsrch -dst_oid 101 b rpsm.c

Убедитесь, что третий подписчик получил ошибки с кодом -14 (RESPONSE_MISMATCH).

Можно, разумеется, сделать подписку и для unicast-протоколов. Каналы будут в этом случае индивидуальны, т.е. ассоциироваться с номером абонента, а не с группой.

6. Global Weather - обмен по SOAP/HTTP: sgweather.c сервер и cgweather.c клиент.

В глобальной сети реализована только модель клиент-сервер. Поддерживаются запросы:
- HTTP GET,
- HTTP POST,
- SOAP 1.1,
- SOAP 1.2.

Реализация веб-сервиса основана на http://www.webservicex.net/globalweather.asmx?op=GetWeather. Это - пример веб-сервиса и клиентские HTML-страницы, позволяющие с ним работать.

В данном примере на ТА предлагаются свои клиенты и сервер. Они также реализуют интерфейсы GetWeather, GetCitiesByCountry. ТА-клиенты могут работать с удаленным www.webservicex.net, равно как и браузер с Вашим сервером. Для этого есть сохраненные и модифицированные странички GetWeather.htm, GetCitiesByCountry.htm.

Запустите серверное приложения из консоли командой (запускайте сначала серверное приложение, затем - клиентские).
sgweather -port 8080
Запустите браузер на том же компьютере страничкой GetWeather.htm. Введите страну и город, получите HTML-ответ. В страничке установлен запрос:

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

Измените, если запускаете с другого компьютера или порта.
Запустите браузер страничкой GetCitiesByCountry.htm, проделайте те же операции. Вы испытали запрос HTTP GET.

Запустите клиентское приложение с консольного окна с опцией -hostname, если удаленно. В первом случае выполнится запрос GetCitiesByCountry, во втором - GetWeather.
cgweather -trace_level 26 -port 8080 -soap GET russia
cgweather -port 8080 -soap GET russia moscow
Запустите клиентское приложение с опциями -soap POST, -soap 11, -soap 12. Убедитесь в работе обмена по протоколам SOAP 1.1, SOAP 1.2 .

Попробуйте запросить удаленный сервер (тот работает на порте 80):
cgweather -hostname www.webserviceX.NET -soap 11 Japan
cgweather -hostname www.webserviceX.NET -soap 12 Japan Tokyo

Попробуйте запросить сервер параметрами Japan Osaka, Russia Moscow, Russia Vassuki. В последнем случае должны получить No Data.