maXbox Starter 15

Start with Serial Programming

1.1 Set an Interface

Today you begin to learn about serial programming in Object Pascal. If you are new to peripheral programming, at first glance it might appear astonishing.

Today I'll try to eliminate any confusion by presenting a clear picture of the labyrinth known as serial programming. First, I'll give you an overview of the RS 232 standard. After that I'll go over some of the data specification with the serial tester included in maXbox.

Serial communication is based on a protocol and the standard RS 232. A protocol is one or a few sets of hardware and software rules agreed to by all communication parties for exchanging data correctly and efficiently.

RS-232C, EIA RS-232, or simply RS-232, refers to the same standard defined by the Electronic Industries Association in 1969 for serial communication.

We will concentrate on creating a serial call of a component using a class `TSerial` from a component which calls the `winapi dll` library. But, first of all I'll explain you what "blocking clock" and "non-blocking signal" calls are. In fact there are 2 programming models used in communication driven applications:

- Synchronous or share the same clock
- Asynchronous or non timing signal

1.1.1 Synchronous and Asynchronous Communications

Don’t confuse it with parallel programming (simultaneous) in which we can have two synchronous functions but each in a separate thread to split and speed up the results.
Sure, two asynchronous functions can be started at the same time if they can handle parallel processing.
Synchronous Communication requires the sender and receiver to share the same clock. The sender provides a timing signal to the receiver so that the receiver knows when to "read" the data.
Synchronous Communication generally has higher data rates and greater error-checking capability. A printer is a form of Synchronous Communication.

Asynchronous Communication has no timing signal or clock. Instead, it inserts Start/Stop bits into each byte of data to "synchronize" the communication. As it uses less wires for communication (no clock signals), Asynchronous Communication is simpler and more cost-effective. RS-232/RS-485/RS-422/TTL are the forms of Asynchronous Communications.

Hope you did already work with the Starter 1 to 14 available at:

http://www.softwareschule.ch/maxbox.htm

Non Blocking means that the application will not be blocked when the application plays a sound or a socket read/write data. This is efficient, because your application don't have to wait for a sound or signal result or a connection. Unfortunately, using this technique is little complicated to develop a protocol. If your protocol has many commands or calls in a sequence, your code will be very unintelligible.

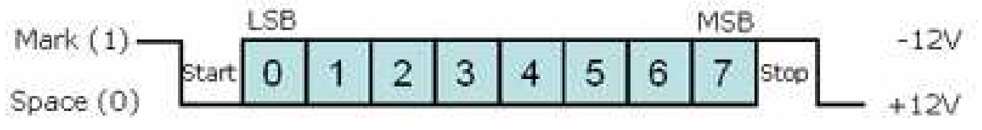
☞ A function which returns no data (has no result value) can always be called asynchronously, cause we don't' have to wait for a result or there's no need to synchronise the functions.

1.1.2 Drill Down: Bits and Bytes

Internal computer communications consists of digital electronics, represented by only two conditions: ON or OFF. We represent these with two well known numbers: 0 and 1, which in the binary system is termed a Bit.
A Byte consists of 8 bits, which represents decimal number 0 to 255, or Hexadecimal number 0 to FF. As described above, a byte is the basic unit of Asynchronous communications.
The baud rate is the communication speed that measures the number of bit transfers per second. For example, 19200 baud is 19200 bits per second.

Data bits are a measurement of the actual data bits in a communication packet. For example, the above graphic shows eight (8) data bits in a communication packet. A communication packet refers to a single byte transfer, including Start/Stop bits, Data bits and Parity. If you are transferring a standard ASCII code (0 to 127), 7 data bits are enough. If it is an extended ASCII code (128 to 255), then 8 data bits are required.

Parity is a simple way to error-check. There are Even, Odd, Mark and Space indicators. You can also use no parity. For Even and Odd parity, the serial port sets the parity bit (the last bit after the data bit) to a value to ensure that the data packet has an Even or Odd number of logic-high bits. For example, if the data is 10010010, for even parity, the serial port sets the parity bit as 1 to keep the number of logic-high bits Even. For Odd parity, the parity bit is 0 so that the number of logic-high bits is Odd. Mark parity simply sets the parity bit to logic-high and Space sets the parity bit to logic-low, so that the receiving party can determine if the data is corrupted.



Stop bits are used to signal the end of a communication packet. This also helps to synchronize different clocks on the serial devices.

Let's begin with the communication structure process in general of an API call:
Signal functions in the EIA232 standard can be subdivided into six categories.

1. Signal ground and shield.

2. Primary communications channel (includes flow control signals).

3. Secondary communications channel. When implemented, this is used for control of the remote modem, requests for retransmission when errors occur, and governance over the setup of the primary channel.

4. Modem status and control signals. These signals indicate modem status and provide intermediate checkpoints as the telephone voice channel is established.

5. Transmitter and receiver timing signals. If a synchronous protocol is used, these signals provide timing information for the transmitter and receiver, which may operate at different baud rates.

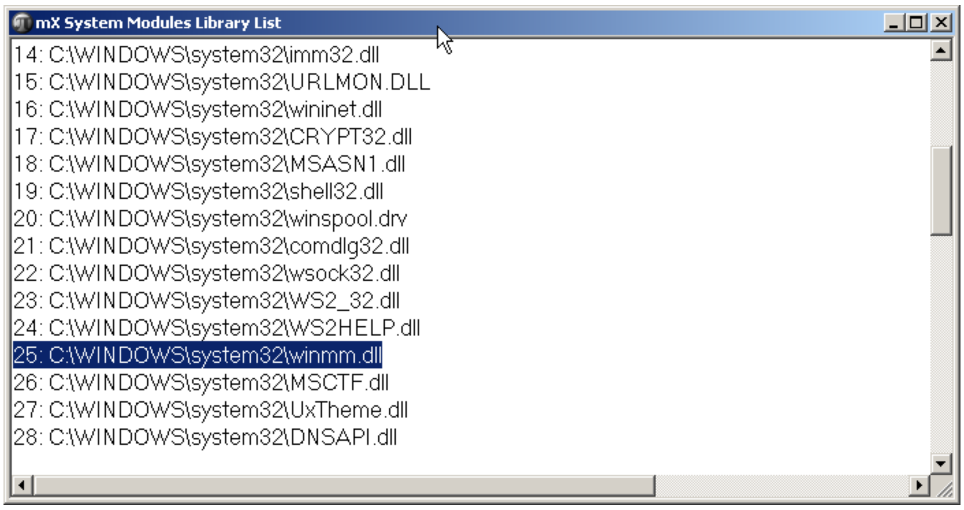
6. Channel test signals.

The standard does not define such elements as the character encoding or the framing of characters, or error detection protocols. The standard does not define bit rates for transmission, except that it says it is intended for bit rates lower than 20,000 bits per second. Many modern devices support speeds of 115,200 bit/s and above.
Over the 40+ years since this standard was developed, the Electronic Industries Association published three modifications, the most recent being the EIA232F standard introduced in 1997. Besides changing the name from RS232 to EIA232, some signal lines were renamed and various new ones were defined, including a shield conductor.

If you click now on the menu `</Output/Serial RS232/…>` you can see almost all the settings (configuration) loaded by the app.

```
TSerialConfig = record
    Valid: boolean;             // true: parameters are valid
    COMPort: integer;           // interface
    Baudrate: integer;          // transfer rate
    DataBits: TDataBits;        // count of data bits
    ParityBit: TParityBit;      // parity
    StopBits: TStopBits;        // count of stop bits
    BufSizeTrm: integer;        // size of send buffer
    BufSizeRec: integer;        // size of receive buffer
    HandshakeRtsCts: boolean;   // Hardwire-Handshake RTS/CTS
    HandshakeDtrDsr: boolean;   // Hardwire-Handshake DTR/DSR
    HandshakeXOnXOff: boolean;  // Handshake XOn/XOff-protocol
    RTSActive: boolean;         // RTS-Output if no handshaking
    DTRActive: boolean;         // DTR-Output if no handshaking
    XOnChar: char;              // XOn-Sign
    XOffChar: char;             // XOff-Sign
    XOffLimit: integer;         // XOff-Limit
    XOnLimit: integer;          // XOn-Limit
    ErrorChar: char;            // error sign character
    EofChar: char;              // EOF-sign
    EventChar: char;            // event sign
    ContinueOnXOff: boolean;    // continue transfer despite XOff
    UseErrorChar: boolean;      // use error sign
    EliminateNullChar: boolean; // delete null sign
    AbortOnError: boolean;      // break on error
    RecTimeOut: integer;        // timeout (ms) for data receive (0=no timeout)
    TrmTimeOut: integer;        // timeout (ms) for data transfer (0=no timeout)
  end
```

Signal names that imply a direction, such as Transmit Data and Receive Data, are named from the point of view of the DTE device. If the EIA232 standard were strictly followed, these signals would have the same name for the same pin number on the DCE side as well. Unfortunately, this is not done in practice by most engineers; probably because no one can keep straight which side is DTE and which DCE is.


mX System Modules Library List

```
14: C:\WINDOWS\system32\imm32.dll
15: C:\WINDOWS\system32\URLMON.DLL
16: C:\WINDOWS\system32\wininet.dll
17: C:\WINDOWS\system32\CRYPT32.dll
18: C:\WINDOWS\system32\MSASN1.dll
19: C:\WINDOWS\system32\shell32.dll
20: C:\WINDOWS\system32\winspool.drv
21: C:\WINDOWS\system32\comdlg32.dll
22: C:\WINDOWS\system32\wsock32.dll
23: C:\WINDOWS\system32\WS2_32.dll
24: C:\WINDOWS\system32\WS2HELP.dll
25: C:\WINDOWS\system32\winmm.dll
26: C:\WINDOWS\system32\MSCTF.dll
27: C:\WINDOWS\system32\UxTheme.dll
28: C:\WINDOWS\system32\DNSAPI.dll
```

1: Result as of the loaded Modules

1.2 Code with Output Menu

As you already know the tool is split up into the toolbar across the top, the editor or code part in the centre and the output window shell like a song length protocol result at the bottom.
Included in the package maXbox is the integrated Test Unit as an add-on listed on the menu `/Output/Serial RS232/`. We show the tool to test some settings, see below `TSerial` Testbox.

If you set the flag `SND_NOSTOP` we get an interesting constellation.
This specified sound event will yield to another sound event that is already playing. If a sound cannot be played because the resource needed to generate that sound is busy playing another sound, the function immediately returns FALSE without playing the requested sound.
If this flag is not specified, `PlaySound` attempts to stop the currently playing sound so that the device can be used to play the new sound (again).

Discussion:
I have an app that has a background sound and a sound when someone clicks something. But when they click the something sound it stops the background sound, which is supposed to loop. How can I play both sounds and leave the background looping?

Answer:
I'm 99% sure that the `PlaySound` API (which `SoundPlayer.Play` is wrapping) does not support this no matter how you do it. You will have to look at something like DirectX or multithreading if you really want to do this.
I'm trying to get the `SoundPlayer` to play two wav files at the same time. The documentation says is should play asynchronously but when I try, it seems to end the play of the first `SoundPlayer` as soon as the second one starts to play.
I read the documentation, and it doesn't imply that it will merge the two sounds, merely that the method call is asynchronous. Internally `PlaySound` calls the `PlaySound` API from `mmsystem.h`, with the parameters `SND_ASYNC` and `SND_NODEFAULT`. This means so far that "The sound is played asynchronously and `PlaySound` returns immediately after beginning the sound..." and "No default sound event is used. If the sound cannot be found, `PlaySound` returns silently without playing the default sound."

So, as far as I can see it implies that the API call queues up the sounds to be played, and the method returns when your sound starts playing, so there’s no parallel processing possible!

Before this starter code will work you will need to download maXbox from the website. It can be down-loaded from http://sourceforge.net/projects/maxbox site. Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is. If you double-click `maxbox3.exe` the box opens a default program. Make sure the version is at least 3.8 because `Modules Count` use that. Test it with F2 / F9 or press **Compile** and you should hear a sound a browser will open. So far so good now we'll open the example:

`263_async_sound.txt`

If you can't find the file use the link (size 3 KB, 2666 Bytes):

http://www.softwareschule.ch/examples/263_async_sound.txt

Or you use the `Save Page as…` function of your browser[1] and load it from `examples` (or wherever you stored it). One important thing: The mentioned DLL must reside on your disk. Now let's take a look at the code of this project first with the declaration of the DLL. Our first line is

```
6 Program ASYNC_Sound_Parallel;
```

We have to name the program it's called `ASYNC_Sound_Parallel;` next we jump to line 8:

```
7
8 function PlaySound(sres: pchar; hmod: HMODULE; syncflag_: dword): bool;
9   external 'PlaySound@winmm.dll stdcall';
```

First in line 8 we ask to set function parameters of a DLL. Parameters are important for all IT systems that offer interfaces for user or programming access. Particularly, this includes machines in the private and commercial sector as well as services like electronics or multimedia. In these cases the `external` declaration is usually done by entering a DLL name and the according function name.

As we now know, `PlaySound` can't play two sounds at the same time, even if you do use `async` flag. You might be able to get this to work by having two separate threads both calling `PlaySound` synchronously.

1.2.1 Media Player

The "MediaPlayer" object will not let you play two sounds at once, even if you create two instances. You will need to bring in the native windows API "`mciSendString`".

You can test it with F4 or menu `/Output/New Instance` which opens a second instance of maXbox (see the following screenshot) with the same script!
An example of the low-level `mciSendString()`:

```
mciSendString(@"open C:\Users\Desktop\applause.wav type waveaudio alias
applause", null, 0, IntPtr.Zero);
mciSendString(@"play applause", null, 0, IntPtr.Zero);

mciSendString(@"open C:\Users\Desktop\foghorn.wav type waveaudio alias
foghorn", null, 0, IntPtr.Zero);
mciSendString(@"play foghorn", null, 0, IntPtr.Zero);
```

So your code will now show all three possibilities in a sequence. First with start with a dialog which stops the control flow because it's a modal dialog we have to wait and pass it. Second on line 28 our function is called in sync-mode `sync` and we have to wait or in other words we are blocked. Third in line 30 we call the same function with the `async` flag set to 1 and now you can follow at the same time the music plays and a loop of numbers on the output can be seen.
In line 33 we start almost the same time a sound twice, yes it's the same sound therefore you can hear the delay or an echo to prove its parallel! Ok. It's a trick to open another function with the same song to show the simultaneous mode.

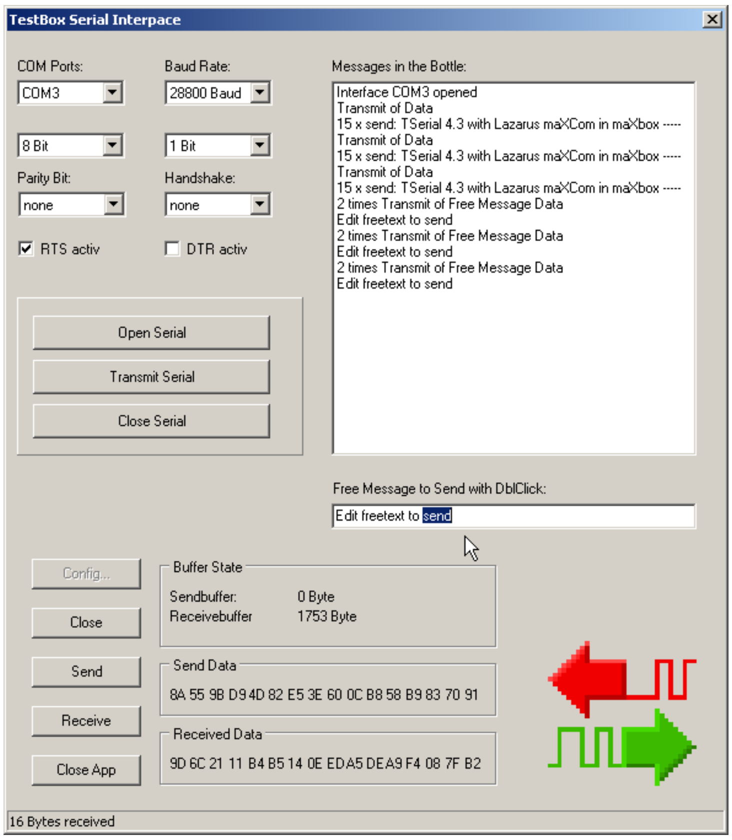
[1] Or copy & paste

If you increase the delay of sound (for example with sleep or with a massive CPU payload), the echo effect space grows to more than milliseconds time of its start offset.

```
//Result:= Writeln(FloatToStr(IntPower(62,8))) = 218340105584896
  ShowMessage('the boX rocks ' + MYS)
  PlaySound(pchar(ExePath+'examples\maxbox.wav'), 0, 0);  //Sync
  Sleep(20)
  PlaySound(pchar(ExePath+'examples\maxbox.wav'), 0, 1);  //Async
  //Parallel
  //sleep(20)
  closeMP3;
  playMP3(mp3path);                                       //Parallel Trick
  for i:= 1 to 2300 do
    writeln(intToStr(i) + ' Aloha from ObjectPascal Bit');
  sleep(1250)
  inFrm.color:= clblue;
  //inFrm.close;
End.
```

☞ This `sleep` is here just so you can distinguish the two sounds playing simultaneously.

By the way: You can also set a hash function around a sound file to distinguish it.
A hash function is a (mathematical) function which receives an input of arbitrary length and returns a function value (hash value) of a fixed length (usually 128 or 160 bits).


TestBox Serial Interpace

COM Ports: COM3
Baud Rate: 28800 Baud
8 Bit
1 Bit
Parity Bit: none
Handshake: none
[✓ RTS activ]
DTR activ

Messages in the Bottle:

```
Interface COM3 opened
Transmit of Data
15 x send: TSerial 4.3 with Lazarus maXCom in maXbox -----
Transmit of Data
15 x send: TSerial 4.3 with Lazarus maXCom in maXbox -----
Transmit of Data
15 x send: TSerial 4.3 with Lazarus maXCom in maXbox -----
2 times Transmit of Free Message Data
Edit freetext to send
2 times Transmit of Free Message Data
Edit freetext to send
2 times Transmit of Free Message Data
Edit freetext to send
```

Free Message to Send with DblClick: Edit freetext to send

Buffer State
Sendbuffer: 0 Byte
Receivebuffer 1753 Byte

Send Data
8A 55 9B D9 4D 82 E5 3E 60 0C B8 58 B9 83 70 91

Received Data
9D 6C 21 11 B4 B5 14 0E EDA5 DEA9 F4 08 7F B2

16 Bytes received

2. TSerial Testbox

Till now we are discussing the topics of `sync` and `async` calls and the way it processes the calls by the receiver parallel or not. Asynchronous calls or connections allow your app to continue processing without waiting for the process (function) to be completely closed but not always in a simultaneous mode. Therefore we come now to the discussion about real parallel programming with threads (multithreading) or multiprocessing programming.

1.2.2 Checksum

Many serial protocols use checksum (additional bytes added at the end of the data string) to check the data integrity, as errors might occur during data transmission.

There are many types of checksum, from the simplest uses of it in Modula or BCC to sophisticated CRC calculation. Using Modula as an example, we learn that before data transmission, the sender adds all command bytes together then mod it by 255 (decimal) to get an additional byte. This is to be added at the end of the command string. When the receiver receives the command string, it will first check the added byte to see whether data remain unchanged or not. If that is the case, it will accept the data, and if not, it will ask the sender to resend the data.

1.2.3 Examples of protocol commands

A protocol command is a data string sent from one serial device (e.g. a Computer) to another (i.e. a Modem). Here are some examples:

ASCII command example: `ATI1<CR><LF>` to query Modem manufacturer's information. (Note: `<CR><LF>` are the control codes: Carriage Return and Line Feed)

Convert the command string above to Hexadecimal and it becomes:
41 54 49 31 0D 0A

Convert the command string above to Decimal and it becomes:
065 084 073 049 013 010

Convert the command string above to Octal and it becomes:
101 124 111 061 015 012

Convert the command string above to Binary and it becomes:
01000001 01010100 01001001 00110001 00001101 00001010

1.3 Notes about Threads (remake)

Multi-threaded applications are applications that include several simultaneous paths of execution. While using multiple threads requires careful thought, it can enhance your programs by:

- Avoiding bottlenecks. With only one thread, a program must stop all execution when waiting for slow processes such as accessing files on disk, communicating with other machines, or displaying multimedia content. The CPU sits idle until the process completes. With multiple threads, your application can continue execution in separate threads while one thread waits for the results of a slow process.
- Organizing program behaviour. Often, a program's run can be organized into several parallel processes that function independently. Use threads to launch a single section of code simultaneously for each of these parallel cases.
- Multiprocessing. If the system running your program has multiple processors, you can improve performance by dividing the work into several threads and letting them run simultaneously on separate processors.

One word concerning a processing thread: In the internal architecture there are 2 threads categories.

- Threads with synchronisation (blocking at the end)
- Threads without synchronisation (non blocking at all)

In case you're new to the concept, a thread is basically a very beneficial alternative (in most cases) to spawning a new process. Programs you use every day make great use of threads whether you know it or not but you have to program it.

The most basic example is whenever you're using a program that has a lengthy task to achieve (say, downloading a file or backing up a database), the program (on the server) will most likely spawn a thread to do that job.
This is due to the fact that if a thread was not created, the main thread (where your main function is running) would be stuck waiting for the event to complete, and the screen would freeze.
For example first is a listener thread that "listens" and waits for a connection. So we don't have to worry about threads, the built in thread will be served by for example Indy though parameter:

```
   IdTCPServer1Execute(AThread: TIdPeerThread)
```

When our DWS-client is connected, these threads transfer all the communication operations to another thread. This technique is very efficient because your client application will be able to connect any time, even if there are many different connections to the server.
The second command "`CTR_FILE`" transfers the app to the client:

Or another example is AES cryptography which is used to exchange encrypted data with other users in a parallel way but not a parallel function. It does not contain functions for key management. The keys have to be exchanged between the users on a secure parallel channel. In our case we just use a secure password!

```
procedure EncryptMediaAES(sender: TObject);


 with TStopwatch.Create do begin
    Start;
    AESSymetricExecute(selectFile, selectFile+'_encrypt',AESpassw);
    mpan.font.color:= clblue;
    mpan.font.size:= 30;
    mpan.caption:= 'File Encrypted!';
    Screen.Cursor:= crDefault;
    Stop;
    clstBox.Items.Add('Time consuming: ' +GetValueStr +' of: '+
          inttoStr(getFileSize(selectFile))+' File Size');
    Free;
 end;
```

And that's how to get a feeling of the speed of AES with a protocol extract:

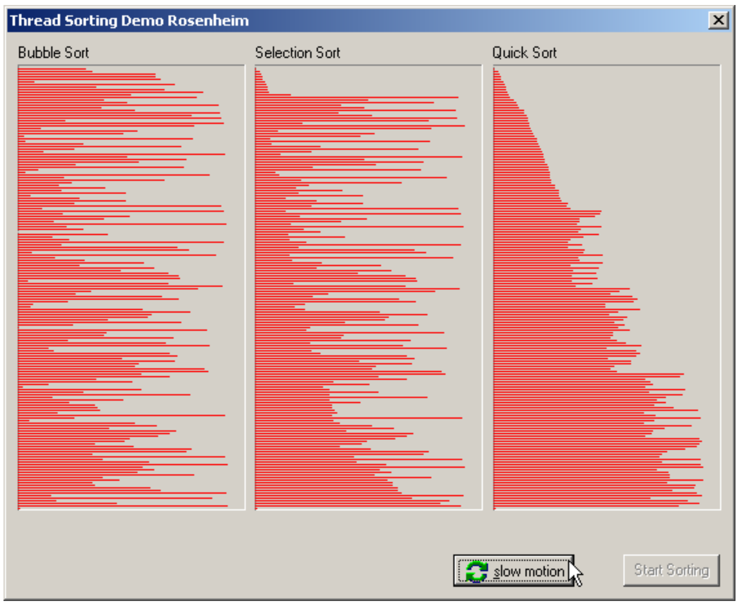
```
Use native.AES-256 cipher and native.CBC chaining mode.
Ciphertext size = 21726 bytes.
Decryption succeeded. 21726 bytes processed.
Speed of decryption was 1326 KiB per second.  //4 GByte about 40 Minutes
```

To understand threads one must think of several programs running at once. Imagine further that all these programs have access to the same set of global variables and function calls.

Each of these programs would represent a thread of execution and is thus called a thread. The important differentiation is that each thread does not have to wait for any other thread to proceed. If they have to wait we must use a synchronize mechanism.

All the threads proceed simultaneously.
To use a metaphor, they are like runners in a race, no runner waits for another runner. They all proceed at their own rate.

☞ Because Synchronize uses a form message loop, it does not work in console applications. For console applications use other mechanisms, such as a mutex (see graph below) or critical sections, to protect access to RTL or VCL objects.



2: 3 Threads at work with Log

The point is you can combine asynchronous calls with threads! For example asynchronous data fetches or command execution does not block a current thread of execution.

But then your function or object has to be thread safe. So what's thread-safe. Because multiple clients can access for example your remote data module simultaneously, you must guard your instance data (properties, contained objects, and so on) as well as global variables.

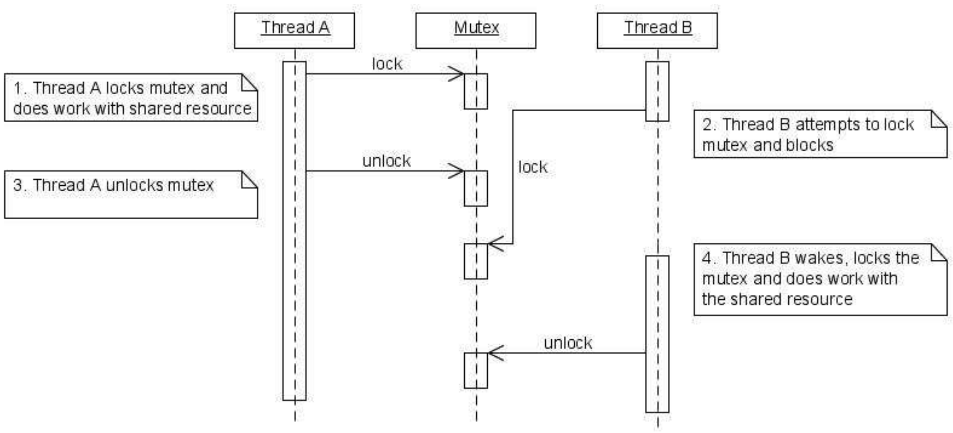
Tasks for advanced studies:

How can you crack a password with a massive parallel concept? Study all about a salt. A 512-bit salt is used by password derived keys, which means there are `2^512` keys for each password. So a same password doesn't generate always a same key. This significantly decreases vulnerability to 'off-line' dictionary' attacks (pre-computing all keys for a dictionary is very difficult when a salt is used). The derived key is returned as a hex or base64 encoded string. The salt must be a string that uses the same encoding.

We also use a lot of more multi scripts to teach and the wish to enhance it with a thread, simply take a look in the meantime at `141_thread.txt`, `210_public_private_cryptosystem.txt` and `138_sorting_swap_search2.txt`. At least let's say a few words about massive threads and parallel programming and what functions they perform.

Do not create too many threads in your apps. The overhead in managing multiple threads can impact performance. The recommended limit is 16 threads per process on single processor systems. This limit assumes that most of those threads are waiting for external events. If all threads are active, you will want to use fewer.

You can create multiple instances of the same thread type to execute parallel code. For example, you can launch a new instance of a thread in response to some user action, allowing each thread to perform the expected response.

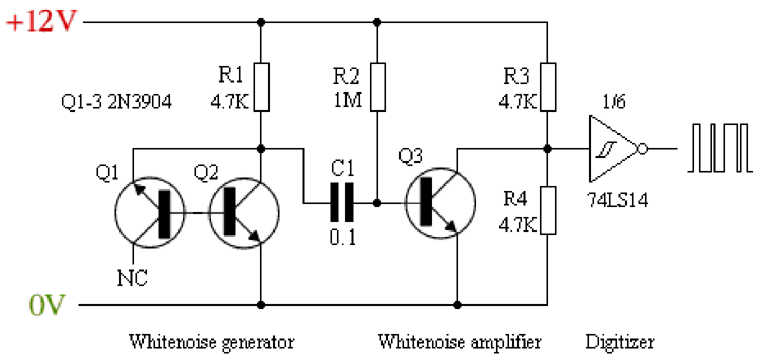


Thread A
Mutex
Thread B
lock
1. Thread A locks mutex and does work with shared resource
2. Thread B attempts to lock mutex and blocks
unlock
lock
3. Thread A unlocks mutex
4. Thread B wakes, locks the mutex and does work with the shared resource
unlock

4: A real Thread with a synchronisation object

One note about async execution with fork on Linux with libc-commands; there will be better solutions (execute and wait and so on) and we still work on it, so I'm curious about comments, therefore my aim is to publish improvements in a basic framework on sourceforge.net depends on your feedback ;)

Try to find out more about symmetric/asymmetric encryption systems and the probability to crack it in a serial way. Due to their large computational requirements, a asymmetric key system is used in practice only for encrypting so called "session keys". The pay load is encrypted with a symmetric algorithm using the session key. This combination of two methods is called hybrid encryption (asymmetric/symmetric). This implies no security deficit if you use a good random number generator for the session key, like a HW generator below!



Try to change the sound file in order to compare two sounds at the same time:

```
04 pchar(ExePath+'examples\maxbox.wav'));
```

Try to find out more about the HW schema above and the following question: does it works in a synchronous or asynchronous mode?

Check the source of LockBox to find out how a thread is used:

```
05 E:\maxbox\maxbox3\source\TPLockBoxrun\ciphers\uTPLb_AES.pas;
```

Try to change the form message with a string literal const and a modal form:

```
with inFrm do begin
  position:= poScreenCenter;
  color:= clred;
  caption:= 'Delphi in a Box';
  show;
end;
```

Next starter will be the number 16 with the topic code reviews.

max@kleiner.com
Links of maXbox, TSerial and Asynchronous Threads of DelphiWebStart:

`059_timerobject_starter2_ibz2_async.txt`

http://sourceforge.net/projects/delphiwebstart

http://www.commfront.com/RS232_Protocol_Analyzer_Monitor/RS232_Analyzer_Monitor_Tester_TUTORIAL3.HTM

http://toolbox.reworld.eu

http://www.softwareschule.ch/maxbox.htm
http://sourceforge.net/projects/maxbox
http://sourceforge.net/apps/mediawiki/maxbox/

My Own Experience:
http://www.softwareschule.ch/download/armasuisse_components.pdf

`SHA1 Hash of maXbox 3.8.5 Win: CDDA808D3B29B0D517CFE6AF2B68AF0A6D6B35D1`