smxNS FAQ

by Richard Ames
Senior Embedded Software Engineer

1. How do I use non-blocking calls?

Both the Dynamic Protocol Interface API (proprietary) and the Sockets API support non-blocking calls. Here are some examples.

DPI

```
SOCKET_NOBLOCK(conn);
bytecount = Nwrite(conn, buff, sizeof(buff);
```

Sockets

```
returncode = fcntlsocket(s, F_SETFL, O_NDELAY);
returncode = send(s, buff, sizeof(buff), 0);
```

or

```
int flag = 1;
returncode = ioctlsocket(s, FIONBIO, O_NDELAY, &flag);
returncode = send(s, buff, sizeof(buff), 0);
```

Even though non-blocking calls are supported, we recommend the use of blocking calls because they usually allow the network logic to be implemented with a more straightforward design.

2. How are tasks used in smxNS?

smxNS can be used either with or without multitasker support. When we are developing smxNS, we usually link it with the SMX® RTOS, but other configurations are supported. When smxNS is used with a multitasker, tasks are used as follows.

Incoming network frame processing and protocol layer timeouts are performed as part of a network task. At start up time, a single NetTask() is launched. This task responds to incoming frames that are queued by interrupt driven network interface drivers. This task also maintains a list of timeouts, and calls into protocol layers as needed to allow them to perform their timeout functions.

Outgoing frames are constructed within the context of the application task. So, for example, all of the following may execute in the context of the application:

- Socket layer processing. Break down the application data into pieces that fit the TCP segment size.
- TCP layer processing. Build a TCP header.
- IP layer processing. Build an IP header and select a network interface for transmission.
- Link layer processing. Create an Ethernet header.
- Device driver processing. Queue the frame for transmission and set up the controller to send the frame that contains the application level data.

3. What sort of porting will I need to do for my system?

Micro Digital supplies the stack ready to run on a range of evaluation boards. Typical customization may include adding logic for retrieving a serialized MAC address for the Ethernet controller and tuning parameters such as the number of network connections and network frame buffers. These customizations are discussed in the smxNS User's Manual, which can be reviewed by prospective customers. The manual also includes chapters on advanced work, such as developing a new network controller driver. Prospective customers should make their needs clear if they intend to extend the package this way, and Micro Digital can make business arrangements if needed.

4. How easy is it to build smxNS?

The code that Micro Digital supplies is targeted to a particular evaluation board and tool set. It should compile and link without any errors or warnings. This is our standard for release.

Features such as smxNS networking support are set up as a project that builds into a library, and this library is pulled into a top level project that builds a sample application. The main smxNS sample application is named NSDEMO.C, and it includes support for an FTP client, FTP server, Telnet server, Web server and others.

If features such as network clients or servers are not enabled in the sample application, the supporting code in the smxNS library will not be linked, and the result will be a smaller image file.

We also include a simple application that just implements the Telnet server by itself (NSTELS.C). This is intended to be a less cluttered example that is easier to understand. It should also be a good starting point for developers that like to build up from "Hello, World!".

5. How does smxNS handle error conditions?

The stack generally follows the TCP/IP Robustness Principle "Be liberal in what you accept, and conservative in what you send". Unrecognized information in a frame may lead to the frame being dropped or the information being ignored or declined, depending upon the circumstances.

The smxNS library includes a panic function that is intended for use during development. If enabled, the panic function is called when a data structure integrity check fails. This should not happen, but it is possible in catastrophic situations such as memory corruption due to a bad pointer or an incorrectly implemented driver.

6. What link layers are support by smxNS?

smxNS supports Ethernet, SLIP and PPP for link layers. The system can be configured to include any mix of network interfaces, and doesn't require any particular link layer. A loopback interface can be configured, but is optional.

7. How many connections can smxNS support?

The number of connections is configured by NCONNS which could be set as high as 254. This is a compile time setting that specifies the number of connection data structures, so setting an appropriate low value will conserve memory. Currently, the connection data structure is around 290 bytes in size. The current default setting for NCONNS is 6 for RAM limited systems. We increased this from 4 when we added to the number of clients and servers in the demo application.

NCONNS should be set to the maximum number of concurrent TCP and UDP connections, plus an additional number to allow for TCP connections in the TIME_WAIT state. TCP client applications typically enter the TIME_WAIT state on closing, and the default timeout in smxNS for this state is a relatively abbreviated 7.5 seconds. Still, if you are turning over connections quickly, then you should budget a few extra connections here.

Also note that servers that use the Sockets API backlog feature will consume an additional connection for each established connection that is held as part of the backlog.

8. How should I determine the number of frame buffers to configure (NBUFFS)?

This is an important consideration, since frame buffers make up the bulk of the smxNS memory requirements. As a rule of thumb, we recommend 5 frame buffers per active TCP connection. This should yield good performance for applications that transfer bulk data over TCP, but it may be overkill in other cases. If some connections might be idle, or more sporadic in data transfer, then you could get equivalent performance with fewer frame buffers.

One more complicating issue with frame buffers is that many Ethernet controllers are designed to use pre-allocated buffers that receive incoming frames. These buffers would be in addition to the rule of thumb "5 per connection".

Recent releases of smxNS include a variable to track the "low water" point of the the frame buffer pool, so you could stress your system and then check this value to see how much headroom you have in the setting of NBUFFS. This value can even be accessed through a Telnet session to the device, or using the optional smxAware "RTOS Awareness" plug in.

9. Can I use a smaller frame buffer size?

The frame buffer size (MAXBUF, typically 1536) is configurable at compile time. The most flexible setting (also the default) is based on the size of an Ethernet frame, but you can also set this lower, and the Maximum Segment Size option of the TCP handshake will allow TCP bulk transfers to work well. On the downside, if you have UDP applications that expect to be able to send large UDP messages, these may fail.

10. What features does smxNS PPP have?

smxNS PPP (Point to Point Protocol) is based on industry standard RFC 1661 for establishing a link to a single remote host, such as an Internet Service Provider (ISP), often via a serial or modem link. A dialer and sample dial-up scripts are included, as well as a runtime script interpreter that allows the system to be tailored to proprietary modems or ISP configurations. The system can be localized by pulling in certain script parameters (such as a dial up number) from non-volatile memory. smxNS PPP includes the following features:

LCP (Link Control Protocol) Options

- Protocol Field Compression
- Magic Number
- Maximum Receive Unit
- Authentication Protocol
- Maximum LCP Termination Requests
- Maximum LCP Configure Requests
- LCP Retransmission Timeout

IPCP (Internet Protocol Control Protocol) Options

- IPCP Compression Protocol
- IP Address
- Maximum IPCP Configuration Requests
- IPCP Retransmission Timeout
- Van Jacobson Compression

The PPP link can be established and taken down automatically in response to information being queued for transfer, or it can be controlled directly by application level logic.