NetSilicon
A Digi International Company

How to use the `select()` function call in an environment that contains Low Memory Network Heap conditions

Introduction

The `select()` function is useful for single-threaded concurrent clients and servers. This function:

- Allows a client or server to gain visibility into the activity on more than one socket at a time
- Prevents unnecessary calls to the sockets API
- Eliminates potential unintended thread blocking.

Low memory network heap conditions occur either when the amount of TCP/IP network heap allocation is inadequate or the level of activity on the network is unusual - specifically, when a burst of short-lived TCP connections are established and closed. With TCP, a closed socket requires memory that was associated with its state prior to closing for as much as several minutes after the program closes it . As a result, as short-lived connections begin to accumulate, the network heap diminishes and eventually results in a low memory network heap condition.

When client-server software executes in a low memory condition, the sockets API functions have potential to fail because they can't complete their processing if the network heap is not able to allocate memory.

How does a Low Memory Network Heap condition affect `select()` ?

Some client-server applications never consider the case in which `select()` fails. Furthermore, if they do consider `select()` failing, it's usually related to invalid sockets being passed into read, write, or exception set arguments. Error handling for these types of failures can include closing the socket, or even worse, causing the client-server to terminate execution.

Netsilicon recommendation

Here is a guideline to writing proper single-threaded concurrent client-servers:

> *When you use `select()`, if the function returns an error, check the system errno. If the errno equals ENOMEM,NetSilicon recommends a `tx_thread_sleep(0)` or a `tx_thread_relinquish`, followed by another iteration through the `select()` call.*

Note that if the client-server application thread was running at a higher priority than the other system or application threads, you would call t`x_thread_sleep(1)` instead of the `tx_thread_relinquish()`or `tx_thread_sleep(0)` calls. This is because the iteration will loop continuously until memory is freed. Relinquishing (or sleeping 0 ticks) allows other threads to run. If, however this thread is high priority, other threads will not run, making a 1 tick sleep necessary.

Sample source code fragment

The source code fragment shown next demonstrates how to call `select()`in the case of the application thread running at a higher priority:

```
do{
    FD_ZERO(&fdSet);
    FD_SET(fd, & fdSet);

    timeout.tv_sec = 10;
    timeout.tv_usec = 0;

    hits = select (fd + 1, &fdSet, NULL, NULL, &timeout);
    if (hits < 0){
        if (getErrno()== ENOMEM){
            /*  sleep -- otherwise this will spin furiously */
            tx_thread_sleep (1);
        }
    }
} while (hits < 0);
```

This source code fragment demonstrates the case of all application threads running at the same priority:

```
do{
    FD_ZERO(&fdSet);
    FD_SET(fd, & fdSet);

    timeout.tv_sec = 10;
    timeout.tv_usec = 0;

    hits = select (fd + 1, &fdSet, NULL, NULL, &timeout);
    if (hits < 0){
        if (getErrno()== ENOMEM){
            /*  go back to the ready to run state */
            tx_thread_sleep (0);
        }
    }
} while (hits < 0);
```