Network Tasks Configuration

This section defines the configuration structures related to C/TCP-IP but that are application-specific. All these configurations relate to the RTOS. For many OSs, the C/TCP-IP task priorities and stack sizes will need to be explicitly configured for the particular OS (consult the specific OS’s documentation for more information).

These configurations are defined in the `net_cfg.c` file.

**Network Task Configuration**

µC/TCP-IP use the following structure to configure its network tasks.

```c
typedef struct net_task_cfg {
    CPU_INT32U    Prio;          /* Task priority. */
    CPU_INT32U    StkSizeBytes;  /* Size of the stack. */
    void        *StkPtr;         /* Pointer to base of the stack. */
} NET_TASK_CFG;
```

Listing - µC/TCP-IP task configuration structure

µC/TCP-IP stack has three internal tasks that need to be configured: the Receive task, the Transmit De-allocation task and the Timer task. Each task has its own `NET_TASK_CFG` object defining the task priority, the task's stack size and the pointer to start of task stack.

**Task Priorities**

We recommend you configure the Network Protocol Stack task priorities & Network application task priorities as follows:

- Network TX De-allocation task (highest priority)
• Network application tasks, such as HTTPs instance.

• Network timer task

• Network RX task (lowest priority)

We recommend that the uC/TCP-IP Timer task and network interface Receive task to be lower priority than almost all other application tasks; but we recommend that the network interface Transmit De-allocation task to be higher priority than all application tasks that use uC/TCP-IP network services.

However better performance can be observed when the network application task is set with the lowest priority. Some experimentation could be required to identify the best task priority configuration.

**Task Stack Size**

In general, the size of µC/TCP-IP task stacks is dependent on the CPU architecture and compiler used.

The only guaranteed method of determining the required task stack sizes is to calculate the maximum stack usage for each task. Obviously, the maximum stack usage for a task is the total stack usage along the task’s most-stack-greedy function path plus the (maximum) stack usage for interrupts. Note that the most-stack-greedy function path is not necessarily the longest or deepest function path.

The easiest and best method for calculating the maximum stack usage for any task/function should be performed statically by the compiler or by a static analysis tool since these can calculate function/task maximum stack usage based on the compiler’s actual code generation and optimization settings. So for optimal task stack configuration, we recommend to invest in a task stack calculator tool compatible with your build toolchain.

On ARM processors, experience has shown that configuring the task stacks to 1024 `OS_STK` entries (4,096 bytes) is sufficient for most applications. Certainly, the stack sizes may be examined and reduced accordingly once the run-time behavior of the device has been analyzed and additional stack space deemed to be unnecessary.
Task Stack Location and Allocation

If a specific memory location is desired for a task stack, the StkPtr parameter can be set to point to this specific memory segment. Else, if StkPtr is set to NULL, the task stack will be allocate on µC/LIB Heap.