NDK Dynamic Memory Manager

Contents

The NDK uses heaps created by SYS/BIOS for run-time allocation. The main items allocated from these heaps are:

- **SYS/BIOS allocations:**
  - **Task related items:** Tasks objects and task stacks. These are allocated by SYS/BIOS from the default SYS/BIOS system heap (unless you specify a different heap using the `Task.common$.instanceHeap` or `Defaults.common$.instanceHeap` advanced configuration properties).
  - A small amount of additional RAM is used to create a few other SYS/BIOS objects, such as Semaphores and Hwi objects. The amount of memory used for these objects is trivial compared to the other items allocated from heaps.
- **NDK allocations:** For the remaining items, heap allocation is performed by the NDK `mmBulkAlloc()` function. This function uses the default SYS/BIOS heap (for example, by calling `Memory_alloc(NULL, ...)`). You can call the `_mmCheck()` API to get a snapshot of the `mmBulkAlloc()` statistics.
  - **Socket related items:** For example, Rx/Tx buffers.
  - **Nettools services internal structures:** DHCP client/server, DNS server, etc.
  - **Tools services:** The packages\ti\ndk\tools directory contains sample code that might be useful to an application.

The `mmAlloc()` API is also used for NDK allocations, but `mmAlloc()` gets memory from the NDK Static Internal Memory Manager.

The zero-copy option or jumbo frames within NDK are not covered in this discussion.

Tasks

The NDK creates the following Task objects:

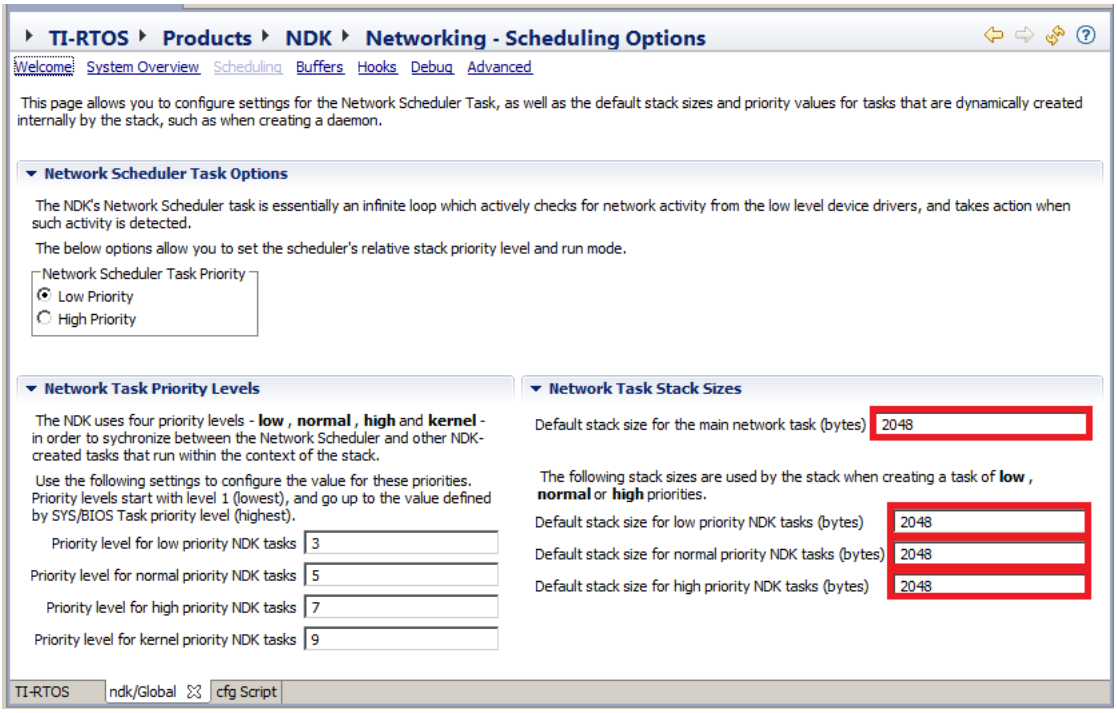
1. **Main NDK Task:** This is the main NDK task.
2. **Boot Task:** This task is responsible from some initialization activities and is terminated once it is completed. The memory allocated for this Task object is freed when the task terminates.
3. **DHCP Client Task** (if the DHCP Client is used): This task makes the initial DHCP request and subsequent lease renewals.
4. **DaemonNew calls:** This task is created as a result of the application calling `DaemonNew()`. The NDK's HTTP and Telnet servers also call this function.
5. **dchild Tasks:** A daemon creates a dchild task to handle new activity on a socket.
6. **DHCP Server Task** (if the DHCP Server is used): This task processes DHCP requests.
7. **DNS Server Task** (if the DNS Server is used): This task processes DNS requests.
8. **Tools Services:** Example code in the packages\ti\ndk\tools directory creates tasks also.

All the above task objects and task stacks are allocated out of the default system heap (unless you are an advanced user and specify a different heap using the `Task.common$.instanceHeap` or `Defaults.common$.instanceHeap` configuration properties). The size of the task stacks are determined by the following configuration properties:

- `Global.ndkThreadStackSize`
- `Global.lowTaskStackSize`
- `Global.normTaskStackSize`
- `Global.highTaskStackSize`

The size of the Boot Task is not configurable. Its stack size is hard-coded to 2048 bytes. Once completed, the Boot Task terminates and the memory is freed back to the system heap.

You can set these properties by editing the configuration file (*.cfg) directly or graphically using XGCONF. To use the graphical editor, open the NDK settings in your application's configuration file (*.cfg) as described in Configuring NDK Memory Use. Once you see the **TI-RTOS > Products > NDK > Networking - Welcome** configuration panel, choose the **Scheduling** link.



The default values depend on the target. For example, on C6000 targets, larger values are used.

Note: Assigning different stack sizes for different priorities is generally not necessary. The NDK was designed to work with different operating systems. This feature might have more value on non-SYS/BIOS operating systems.

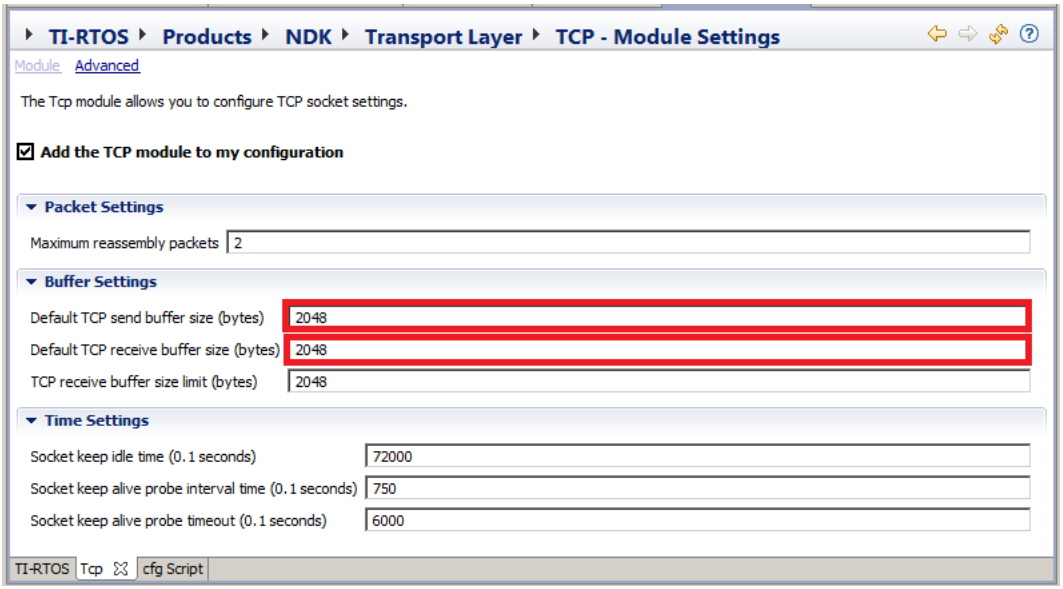
Sockets

When a socket is created, memory is allocated from the default SYS/BIOS heap by the `mmBulkAlloc()` function. The default values for the below configuration properties are target specific (for example, the M3 value is smaller than the C6000 value).

TCP sockets

When a TCP socket is created, two buffers are allocated from the default SYS/BIOS heap. The sizes of these buffers (one Tx and one Rx) are dictated by the `Tcp.transmitBufSize` and `Tcp.receiveBufSize` properties.

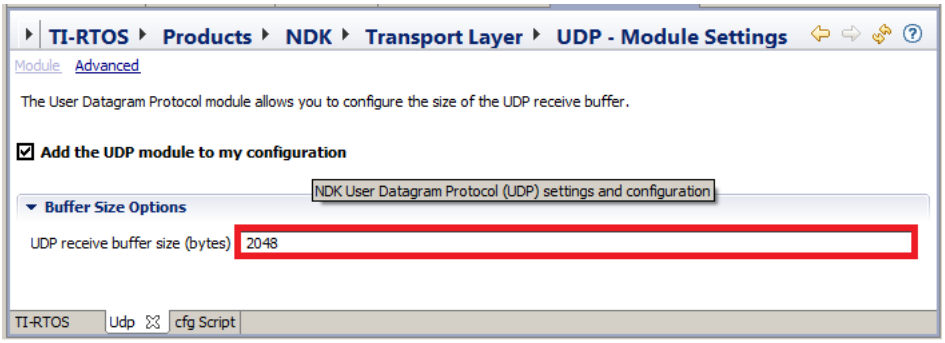
To modify these properties with CCS, select the **Tcp** module in the Outline pane. If the Tcp module is not listed in the Outline pane, expand the tree in the Available Products pane to see the **TI-RTOS > Products > NDK > Transport Layer > Tcp** module. Right-click on the **Tcp** module and choose **Use Tcp**.



UDP sockets

The `Udp.receiveBufSize` property dictates the maximum number of cumulative bytes contained in packet buffers than can be queued up at any given UDP (or RAW based) socket.

To modify this properties with CCS, select the **Udp** module in the Outline pane. If the Udp module is not listed in the Outline pane, expand the tree in the Available Products pane to see the **TI-RTOS > Products > NDK > Transport Layer > Udp** module. Right-click on the **Udp** module and choose **Use Udp**.



Nettool services

The code (for example, telnet and dhcp client) in the nettools directories allocate memory using the `mmBulkAlloc()` API. The size of the allocated memory is based on internal structure sizes and cannot be easily reduced.

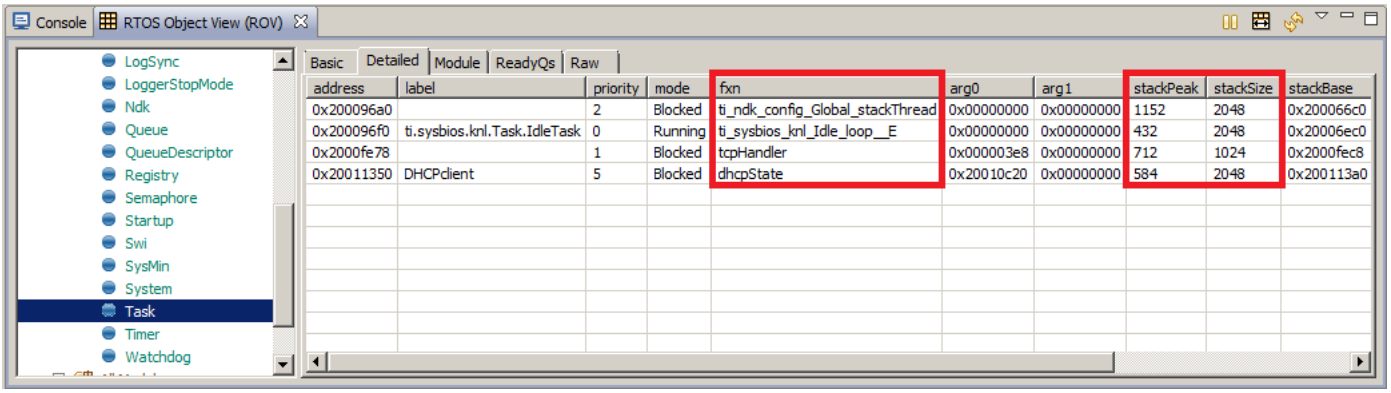
Tools services

The packages\ti\ndk\tools directory contains sample code that might be useful to an application. For example it contains TCP echo and console code. You are free to change the requested memory allocations as you see fit.

How to reduce RAM usage

Reducing Task stack sizes

The most common way to reduce the memory used from the dynamic heaps is to reduce the stack sizes. In CCS Debug mode, you can use the RTOS Object View (ROV) tool to see the "Detailed" view of the Task module. This view allows you to determine whether a task size can be reduced. The TI-RTOS tcpEcho example is used in this figure.



The above figure shows that the following are probably safe assumptions:

- NDK's stackThread stack can be reduced to 1536 bytes (controlled by the `Global.ndkThreadStackSize` property).
- NDK's DHCP stack can be reduced to 768 bytes (controlled by the `Global.lowTaskStackSize` property).
- The applications tcpHandler stack can be reduced to 896 bytes (controlled by the `stackSize` property in the application's `Task_create()` function).
- SYS/BIOS Idle stack can be reduced to 512 (controlled by `Task.idleTaskStackSize` property).

Note: Run the target for awhile under the worst expected conditions before halting it. This provides a more accurate value for the true stack peak.

Reducing socket sizes

Depending on your worst case socket usage, you can reduce the buffers sizes using the `Tcp.transmitBufSize`, `Tcp.receiveBufSize` and `Udp.receiveBufSize` properties.

Final step in reducing dynamic memory usage

Once the above modifications are made, the heaps can be examined in ROV to see how much "extra" memory is present and not used. This heap size can then be reduced as needed.

Additional SYS/BIOS RAM usage reductions are discussed in the "Reducing Data Size" in the *SYS/BIOS User's Guide* (SPRUEX3).

Note: The NDK requires a heap for dynamic allocation. Do not remove heaps from SYS/BIOS.

{{	Keystone=	C2000=*For technical support on the C2000*	DaVinci=*For technical support on DaVincoplease*	MSP430=*For technical support on MSP430*	OMAP35x=*For technical support on OMAP please*	OMAPL1=*For technical support on OMAP please*	MAVRK=*For technical support on MAVRK*	*For technical s please post yo questions at http://e2e.ti.cor*
1. switchcategory:MultiCore=	▪ For technical support on MultiCore devices,							

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **NDK Dynamic Memory Manager** here.

please post your questions in the C6000 MultiCore Forum

- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **NDK Dynamic Memory Manager** here.

*please post your questions on The C2000 Forum. Please post only comments about the article **NDK Dynamic Memory Manager** here.*

*post your questions on The DaVinci Forum. Please post only comments about the article **NDK Dynamic Memory Manager** here.*

*please post your questions on The MSP430 Forum. Please post only comments about the article **NDK Dynamic Memory Manager** here.*

*post your questions on The OMAP Forum. Please post only comments about the article **NDK Dynamic Memory Manager** here.*

*post your questions on The OMAP Forum. Please post only comments about the article **NDK Dynamic Memory Manager** here.*

*please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article **NDK Dynamic Memory Manager** here.*

*Please post on comments abo article **NDK Dy Memory Mana here.*

}}

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters
DLP & MEMS
High-Reliability
Interface
Logic
Power Management
Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity