12/19/2022

NexOS Quick Start Guide

Table of Contents

1.0 Introduction

The goal of this document is to provide the bare minimum amount of information to successfully get a project up and running with the NexOS. It does not go over customization, configuration, functionality, or documentation of the NexOS. It just lists the steps necessary to setup and compile a project with the NexOS. These instructions are valid for kernel version 1.02.00.

1.1 Release Notes

December 19, 2022 – Updated the list of example projects to reflect full list.
August 11, 2022 – Updated the project structure to use GenericTypes.h. Also, the FatFs ff.h file needs modification to work with the GenericTypes.h.
May 25, 2022 – Updated project structure and added IO Buffers example.
August 30, 2020 – Original release.

1.2 License

Below is the license associated with use the of NexOS.

2.0 Required Software

The latest NexOS project must be downloaded from GitHub at the following location https://GitHub.com/pic32/NexOS. This provides the source code for the RTOS along with example projects. Below is a breakdown of the different projects and folders of the repository downloaded from GitHub.

Example Projects:

- Advanced Task Creation
- Callback Timer
- Event Timer
- External Event
- File System
- Handling Exceptions
- Heap Event and Monitoring
- IO Buffer
- Kernel Project
- Pipe
- Simple Semaphore
- Simple Task Creation
- Task Check In

The NexOS folder contains the source code for the RTOS. The contents of this folder must be kept intact to be able to successfully compile the RTOS. The Generic Libraries folder contains code that the NexOS needs to successfully compile. This code is not specific to the OS and could be reused for other applications. The Generic Libraries folder should be located in the same parent folder as the NexOS.

The following section is the unique software required for each port of the NexOS.

2.1 Port Specific Software

Each port has individual dependencies based off of the microcontroller/CPU. The following sections outline each supported port of the NexOS along with the required software to compile the port.

2.1.1 PIC32MX Port

At the time of this document the following versions of software were used to verify the PIC32MX port.

- MPLAB X IDE v5.40 https://www.microchip.com/mplab/mplab-x-ide
- XC32 v2.41 https://www.microchip.com/mplab/compilers

The PIC32 Legacy Peripheral Libraries (located at the bottom of the compiler webpage) are also required to run the PIC32MX port. These libraries must be installed in the same directory as the compiler being used. If the XC32 compiler is located at F:\Microchip\xc32\v2.41, then the libraries should be installed in the location F:\Microchip\xc32\v2.41 too. There's been rumor in far and remote corners of the internet about PLIB warnings induced psychosis when people did not include the _SUPPRESS_PLIB_WARNING option in the build. If you want to avoid this condition please be sure to include it in the build options.

If any of the above software is outdated at the time of you reading this, then it can be found in the below location on Microchips software archive website page:

https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive

2.2 Required Files

The following list are the minimum required source files in order to use the NexOS.

- Kernel.c
- KernelTasks.c
- DoubleLinkedList.c
- Memory.c
- ContexSwitch.S
- CriticalSection.c
- Port.c
- Task.c

The following list is the minimum required header files in order to use the NexOS.

- Kernel.h
- KernelTasks.h
- DoubleLinkedList.h
- Memory.h
- CriticalSection.h
- Port.h
- Task.h

- TaskObject.h
- RTOSConfig.h
- DoubleLinkedListConfig.h
- GenericTypes.h

The file RTOSConfig.h is required in each project and should be added to the project (not referenced). This file is what allows the user to configure the RTOS as needed for the application. The RTOSConfig.h file found in the example program SimpleTaskCreation shows almost the bare minimum configuration to compile the NexOS (USING_MILLISECONDS_TO_TICKS_METHOD, USING_TASK_DELAY_TICKS_METHOD, and USING_KERNEL_VERSION_TO_STRING are not required to compile the NexOS). From here you can modify the configuration file as you need to.

2.3 Project Structure

The directory structure used by the NexOS files requires them to stay together in the folder labeled NexOS. This is due to includes that utilize the .. in the path. Below is an example of this in CriticalSection.c

#include "../../Kernel/Kernel.h"

Due to this organization do not add the OS files to your project folder, only reference them. Your project and the NexOS folder should be under the same parent folder for the paths in the source to work. In the below example the user has created two new projects in folders called “RTOS Hello World 1” and “RTOS Hello World 2” which is also in the parent folder “Projects”. As can be seen by this example the NexOS and Generic Libraries folder are also located in the parent folder “Projects”:

- C:\Projects\
  |----------→RTOS Hello World 1
  |----------→RTOS Hello World 2
  |----------→NexOS
  |----------→Generic Libraries

Both RTOS Hello World 1 and RTOS Hello World 2 have their own RTOSConfig.h file. This allows them to both have their own configuration and reference the same NexOS source code. There are other files which also need to be added to the folders RTOS Hello World 1 and 2 in order for them to work with NexOS. These files are other configuration files and the port which will be used. As of now there is only one valid port and that is located in ..\NexOS\NexOS\Ports\PIC32MX. Below is a list of files that are required to be present in each project folder:

- ContextSwitch.S
- CriticalSection.c
- Port.c
- CriticalSection.h
- Port.h
- DoubleLinkedListConfig.h

- RTOSConfig.h
- GenericTypes.h

2.4 Program Structure

There are 2 essential OS methods that must be called to get the NexOS ready and running. These are the methods InitOS() and StartOSScheduler(). InitOS() must be called before any other method to the OS is called. Once the method StartOSScheduler() is called the NexOS will enable the system timer and look for the highest priority TASK to execute from. Below is a rough example of what the smallest program may look like.

```
#include "../NexOS/Kernel/Task.h"

void TaskCode(void *Args)
{
	while(1)
	{
		// do something
	}
}

int main(void)
{
	// initialize the OS
	InitOS();

	// create 1 TASK
	if(CreateTask(TaskCode, 300, 1, (void*)NULL, (TASK*)NULL) == (TASK*)NULL)
	{
		// handle the error
	}

	// now start the OS and scheduler
	StartOSScheduler();

	// you will never return to here after StartOSScheduler() is called
}
```

Technically no TASKs need to be made by the user and are totally optional. You could just use the Idle Task and/or EVENT callbacks along with CALLBACK_TIMERs and/or EVENT_TIMERs. This would make for a rather dull environment but it is still valid. In all practicality you will be creating at least 1 or more TASKs when using any RTOS.

2.5 Closing Notes

Below are some rules to make sure you follow for proper OS behavior.

- Never call a method that begins with OS.
- If any calls are made to the OS from an interrupt service routine be sure that the method name has ISR in it. Otherwise the OS method should not be called from an interrupt service routine.
- Never call a method that may block from the Idle Task. This is because the OS scheduler makes the assumption that at least 1 valid TASK is always eligible to run.
- Never call a method that may block from a user callback method.
- Only OS methods that have ISR in their name should be called from a user callback.
- Make sure any config file is located locally in your project. This allows multiple projects to use different configurations for the same libraries or OS.
- Make sure to use the ff.h file from the File System project as it contains changes.