Micriµm, Inc.
949 Crestview Circle
Weston, FL 33327
U.S.A.
www.Micrium.com

MicroC/OS-II

The Real-Time kernel

V2.04

Release Notes

Phone: +1 954 217 2036 FAX: +1 954 217 2037

This page is intentionally blank.

V2.04

(2000/10/31)

MISCELLANEOUS:

Removed revision history from all the source code. The revision history is now described in this document. This was done to reduce the amount of 'clutter' from the source files.

Added `OS_ARG_CHK_EN` to enable (when `1`) MicroC/OS-II argument checking. By setting this configuration constant to `0`, you would be able to reduce code size and improve on performance by not checking the range of the arguments passed to MicroC/OS-II functions. However, it is recommended to leave argument checking enabled.

Added Mutual Exclusion Semaphores (OS_MUTEX.C) that are described in AN1002.PDF.

Added support for `OS_CRITICAL_METHOD` #3 that allows the status register of the CPU to be saved in a local variable. The status register is assumed to be saved by `OS_ENTER_CRITICAL()` in a local variable called `cpu_sr` of type `OS_CPU_SR`. The data type `OS_CPU_SR` is assumed to be declared in `OS_CPU.H`. The status register (and thus the state of the interrupt disable flag) is assumed to be restored by `OS_EXIT_CRITICAL()` from the contents of this variable. The macros would be declared as follows:

```
#define OS_ENTER_CRITICAL()  (cpu_sr = OSCPUSaveSR())
#define OS_EXIT_CRITICAL()   (OSCPURestoreSR(cpu_sr))
```

Note that the functions `OSCPUSaveSR()` and `OSCPURestoreSR()` would be written in assembly language and would typically be found in `OS_CPU_A.ASM` (or equivalent).

The check for `OSIntNesting` in all µC/OS-II services is now being done without disabling interrupts in order to reduce interrupt latency. In other words, the following code:

```
OS_ENTER_CRITICAL();
if (OSIntNesting > 0) {
    .
    .
    OS_EXIT_CRITICAL();
}
```

Has been replaced by:

```
if (OSIntNesting > 0) {
    .
    .
}
```

The reason is that ALL currently known processors will treat this byte size variable (OSIntNesting) indivisibly.

OS_CORE.C:

Moved all local variables to uCOS_II.H making them all global variables. This helps when testing.

Calls to OSTaskCreate() and OSTaskCreateExt() in OSInit() now return (void) to indicate that the return value is not being used. This prevents warnings from LINT.

Although not critical, OSInit() was optimized for speed.

Added OSInitHookBegin() at the beginning of OSInit() to allow for a processor port to provide additional 'OS" specific initialization which would be done BEFORE MicroC/OS-II is initialized.

Added OSInitHookEnd() at the end of OSInit() to allow for a processor port to provide additional 'OS" specific initialization which would be done AFTER MicroC/OS-II is initialized.

Initialized .OSEventType to OS_EVENT_TYPE_UNUSED in OSInit().

Added boundary check for OSIntNesting in OSIntEnter() to prevent wrapping back to 0 if OSIntNesting is already at 255.

Added boundary check on OSIntNesting in OSIntExit() to prevent wrapping back to 255 if OSIntNesting is already at 0.

Changed the test for rescheduling in OSIntExit() and OSSched() from:

```
if ((--OSIntNesting | OSLockNesting) == 0) {
```

to

```
if ((OSIntNesting == 0) && (OSLockNesting == 0)) {
```

for sake of clarity.

Removed unreachable code in `OSTaskStat()` for CPU usage > 100%.

Added call to `OSTCBInitHook()` in `OSTCBInit()` to allow user (or port) specific TCB extension initialization.

Moved the increment of `OSTimeTick()` immediately after calling `OSTimeTickHook()`.

Made `OSTime` `volatile`.

OS_MBOX.C:

Removed checking of `pevent` from the critical section to reduce interrupt latency.

Removed checking of `msg` from the critical section to reduce interrupt latency.

Added `OSMBoxDel()` to delete a message mailbox and free up its Event Control Block. All tasks pending on the mailbox will be readied. This feature is enabled by setting `OS_MBOX_DEL_EN` to `1`.

Changed test:

```
     if (pevent->OSEventGrp)
```

to

```
     if (pevent->OSEventGrp != 0x00).
```

OS_MEM.C:

Moved the local variables `OSMemFreeList` and `OSMemTbl[]` to `uCOS_II.H`.

Added code to initialize all the fields of the last node in OSMemInit().

OS_MUTEX.C:

Added services to support Mutual Exclusion Semaphores that are used to reduce priority inversions.

OS_Q.C:

Removed checking of `pevent` from the critical section to reduce interrupt latency.

Removed checking of `msg` from the critical section to reduce interrupt latency.

Added `OSQDel()` to delete a message queue and free up its Event Control Block. All tasks pending on the queue will be readied. This feature is enabled by setting `OS_Q_DEL_EN` to `1`.

Changed test:

```
     if (pevent->OSEventGrp)
```

to

```
     if (pevent->OSEventGrp != 0x00).
```

Moved the definition of the data type `OS_Q` to `uCOS_II.H`.

OS_SEM.C:

Removed checking of `pevent` from the critical section to reduce interrupt latency.

Added `OSSemDel()` to delete a semaphore and free up its Event Control Block. All tasks pending on the semaphore will be readied. This feature is enabled by setting `OS_SEM_DEL_EN` to `1`.

Changed test:

```
     if (pevent->OSEventGrp)
```

to

```
     if (pevent->OSEventGrp != 0x00).
```

OS_TASK.C:

Task stack is now cleared in `OSTaskCreateExt()` when either options `OS_TASK_OPT_STK_CHK` or `OS_TASK_OPT_STK_CLR` is set. The new code is:

```
     if (((opt & OS_TASK_OPT_STK_CHK) != 0x0000) ||
         ((opt & OS_TASK_OPT_STK_CLR) != 0x0000)) {
```

`OSTaskCreateHook()` has been removed from `OSTaskCreate()` and `OSTaskCreateExt()` and moved to `OSTCBInit()` so that the hook is called BEFORE the task is made ready-to-run. This avoids having the possibility of readying the task before calling the hook function.

If you don’t specify any Mailboxes (`OS_MBOX == 0`), Queues (`OS_Q == 0`), Semaphores (`OS_SEM == 0`) or Mutexes (`OS_MUTEX == 0`) in `OS_CFG.H` in order to create a minimal system, `OSTaskChangePrio()` and `OSTaskDel()` will no longer reference `OSTCBEventPtr`.

`OS_TIME.C`:

Added cast to `INT16U` for all references of `tick` in `OSTimeDlyHMSM()`.

`uCOS_II.C`:

Added `OS_MUTEX.C`.

`uCOS_II.H`:

Changed `OS_VERSION` to `204`.

Moved all 'local' variables from `OS_MEM.C`, `OS_Q.C` and `OS_TASKS.C` to simplify debugging and unit testing.

Added constants, data types and function prototypes to support Mutual Exclusion Semaphores.

V2.03

(1999/09/09)

MISCELLANEOUS:

The distribution of µC/OS-II now assumes the Borland C/C++ V4.51 or higher compiler instead of the V3.1 compiler. The code should, however, compile and run using V3.1.

This release contains a slightly different directory structure. The name of the compiler is added to the directory structure in order to support multiple compilers and have the same directory structure for all of these.

`\SOFTWARE\uCOS-II\SOURCE`
Contains the source files for the processor independent code of uC/OS-II.

`\SOFTWARE\uCOS-II\Ix86L\BC45`
Contains the source files for the 80x86 real mode, large model port. The port now contains the function `OSTaskStkInit_FPE_x86()` which needs to be called before you create a task that will use Borland C/C++'s floating-point emulation (FPE) library. See application note AN-1001 found on www.Micrium.com.

`\SOFTWARE\uCOS-II\Ix86L-FP\BC45`
Contains the source files for the 80x86 real mode, large model port. This port also contains hardware floating-point support. In other words, µC/OS-II performs a context switch on the floating-point registers as well as the integer registers. This port was not present on the original distribution of µC/OS-II (i.e. V2.00).

`\SOFTWARE\uCOS-II\EX1_x86L\BC45\SOURCE`
Contains the source code for the sample code of Example #1

`\SOFTWARE\uCOS-II\EX1_x86L\BC45\TEST`
Contains the build files (`MAKETEST.BAT` and `TEST.MAK`) as well as the executable for Example #1. To build the executable for example #1, simply type `MAKETEST` at the DOS prompt. You may have to change `TEST.MAK` to tell it where the Borland C/C++ V4.51 compiler is located. My compiler was located in the `E:\BC45` directory. To execute example #1, type `TEST` at the DOS prompt.

`\SOFTWARE\uCOS-II\EX2_x86L\BC45\SOURCE`
Contains the source code for the sample code of Example #2

`\SOFTWARE\uCOS-II\EX2_x86L\BC45\TEST`

Contains the build files (`MAKETEST.BAT` and `TEST.MAK`) as well as the executable for Example #2. To build the executable for example #2, simply type `MAKETEST` at the DOS prompt. You may have to change `TEST.MAK` to tell it where the Borland C/C++ V4.51 compiler is located. My compiler was located in the `E:\BC45` directory. To execute example #2, type `TEST` at the DOS prompt.

`\SOFTWARE\uCOS-II\EX3_x86L\BC45\SOURCE`
Contains the source code for the sample code of Example #3

`\SOFTWARE\uCOS-II\EX3_x86L\BC45\TEST`
Contains the build files (`MAKETEST.BAT` and `TEST.MAK`) as well as the executable for Example #3. To build the executable for example #3, simply type `MAKETEST` at the DOS prompt. You may have to change `TEST.MAK` to tell it where the Borland C/C++ V4.51 compiler is located. My compiler was located in the `E:\BC45` directory.

To execute example #3, type `TEST` at the DOS prompt.

`\SOFTWARE\uCOS-II\EX4_x86L.FP\BC45\SOURCE`
Contains the source code for the sample code of Example #4

`\SOFTWARE\uCOS-II\EX4_x86L\BC45\TEST`
Contains the build files (`MAKETEST.BAT` and `TEST.MAK`) as well as the executable for Example #4. Example #4 demonstrate the use of `Ix86L-FP`, the port that saves/restores the 80x86's floating-point registers during a context switch. This of course applies for 80x86 processors having a floating-point unit. You may have to change `TEST.MAK` to tell it where the Borland C/C++ V4.51 compiler is located. My compiler was located in the `E:\BC45` directory. To execute example #1, type `TEST` at the DOS prompt.

`\SOFTWARE\BLOCKS\PC\BC45`
Contains the source files for the PC services used to display characters on the screen, read the keyboard etc.

EXAMPLES:

Example #1 (V2.00)

`TEST.C` was previously called `EX1L.C`

`PC_DispClrLine()` has been changed to `PC_DispClrRow()`.

`TaskClk()` now calls `PC_GetDateTime()`.

The floating-point code in `TaskStart()` has been removed so that the task only executes integer arithmetic instructions.

Example #2 (V2.00)

`TEST.C` was previously called `EX2L.C`
Added `TaskStartCreateTasks()` to create all the application tasks.
`TaskStart()` now uses the Borland C/C++ Floating-Point Emulation library and thus, the stack needs to be 'preconditioned' by calling the function `OSTaskStkInit_FPE_x86()` (see `www.Micrium.com`, AN-1001).

`PC_DispClrLine()` has been changed to `PC_DispClrRow()`.

`TaskClk()` now calls `PC_GetDateTime()`.

Example #3 (V2.00)

`TEST.C` was previously called `EX3L.C`

Added `TaskStartCreateTasks()` to create all the application tasks.

`PC_DispClrLine()` has been changed to `PC_DispClrRow()`.

TaskClk() now calls PC_GetDateTime().

Floating-point operations have been replaced with integer operations.

Example #4 (V2.00)

Example #4 is a new example using hardware assisted floating-point.

`TEST.C` was previously called `EX4L.C`

`PC_DispClrLine()` has been changed to `PC_DispClrRow()`.

`TaskClk()` now calls `PC_GetDateTime()`.

PC Services (V2.00)

`PC.C`:

Functions are now listed in alphabetical order in the file.

`PC_ElapsedStart()` and `PC_ElapsedStop()` now protect the critical section of code that accesses the timer ports.

`PC_VectGet()` and `PC_VectSet()` no longer depend on the Borland C/C++ functions `getvect()` and `setvect()`. This should make these functions more portable.

Changed the name of `PC_DispClrLine()` to `PC_DispClrRow()`.

Added function `PC_DispClrCol()`.

The following function now cast `MK_FP()` to `(INT8U far *)`:

```
PC_DispChar()
PC_DispClrLine()
PC_DispClrScr()
PC_DispStr()
```

`PC_ElapsedStop()`, cast `inp()` to `INT8U`.

`PC_GetKey()`, cast `getch()` to `INT16S`.

`PC.H`:

Function prototypes are now listed in alphabetical order.

Added prototype for `PC_DispClrCol()`.

OS_CORE.C:

Changed the return type of `OSEventTaskRdy()` from `void` to `INT8U` to return the priority of the task readied even though the current version of MicroC/OS-II doesn't make use of this feature. This change was done to support future versions.

Moved `OSDummy()` from `OS_TASK.C` to `OS_CORE.C` to be able to call `OSDummy()` from other services.

OS_MBOX.C:

Added check in `OSMboxPost()` to see if the caller is attempting to post a `NULL` pointer. By definition, you should NOT send a `NULL` pointer message. If you attempt to post a `NULL` pointer, `OSMboxPost()` will return `OS_ERR_POST_NULL_PTR`.

Added checks to make sure `pevent` is not a `NULL` pointer. If `pevent` is a `NULL` pointer, each of the following functions will return `OS_ERR_PEVENT_NULL`:

```
OSMboxPost()
OSMboxQuery()
```

Note that `OSMboxAccept()` will return a `NULL` pointer because it doesn't provide the capability of returning an error code.

`OSMboxPend()` sets `*err` to `OS_ERR_PEVENT_NULL` if `pevent` is a `NULL` pointer.

OS_Q.C:

Added check in `OSQPost()` and `OSQPostFront()` to see if the caller is attempting to post a `NULL` pointer. By definition, you should NOT send a `NULL` pointer message. If you attempt to post a `NULL` pointer, `OSQPost()` and `OSQPostFront()` will return `OS_ERR_POST_NULL_PTR`.

Added checks to make sure `pevent` is not a `NULL` pointer. If `pevent` is a `NULL` pointer, each of the following functions will return `OS_ERR_PEVENT_NULL`:

```
OSQFlush()
OSQPost()
OSQPostFront()
OSQQuery()
```

Note that `OSQAccept()` simply returns a `NULL` pointer because it doesn't provide the capability of returning an error code.

`OSQPend()` sets `*err` to `OS_ERR_PEVENT_NULL` if `pevent` is a `NULL` pointer.

OS_SEM.C:

Added checks to make sure `pevent` is not a `NULL` pointer. If `pevent` is a `NULL` pointer, each of the following functions will return `OS_ERR_PEVENT_NULL`:

```
OSSemPost()
OSSemQuery()
```

Note that `OSSemAccept()` returns `0` because it doesn't provide the capability to return an error code.

`OSSemPend()` sets `*err` to `OS_ERR_PEVENT_NULL` if `pevent` is a `NULL` pointer.

OS_TASK.C:

Moved `OSDummy()` to `OS_CORE.C`

uCOS_II.H:

Added error code `OS_ERR_POST_NULL_PTR` (value is `3`).

Changed the return type of `OSEventTaskRdy()` from `void` to `INT8U` to return the priority of the task readied.

Added function prototype for `OSDummy()`.

Added error code `OS_ERR_PEVENT_NULL` (value is `4`)

V2.02

(1999/07/18)

OS_MBOX.C:

Removed last `else` statement in `OSMboxPend()` because the code is unreachable.

OS_Q.C:

Removed last `else` statement in `OSQPend()` because the code is unreachable.

OS_TASK.C:

`OSTaskCtr` is always included.

uCOS_II.C:

Added check for definition of macro `OS_ISR_PROTO_EXT` so that the prototype of `OSCtxSw()` and `OSTickISR()` can be changed based on compiler specific requirements. To use a different prototype, simply add:

```
    #define OS_ISR_PROTO_EXT
```

in `OS_CPU.H` of the port and then define the new prototype format for `OSCtxSw()` and `OSTickISR()` in `OS_CPU.H` of the port.

`OSTaskCtr` is always included. Previously it was conditionally compiled only if `OS_TASK_CREATE_EN`, `OS_TASK_CREATE_EXT_EN` or `OS_TASK_DEL_EN` was set to `1`. It turns out that you MUST always have either `OS_TASK_CREATE_EN` or `OS_TASK_CREATE_EXT_EN` set to `1` anyway!

This page is intentionally blank.

V2.01

(1999/07/15)

OS_CORE.C:

Changed `for` loop inside `OSEventWaitListInit()` to inline code for speed. This eliminates the loop overhead.

The argument `stk_size` in `OSTCBInit()` has been changed from `INT16U` to `INT32U` to accommodate large stacks.

OS_MBOX.C:

Changed `'for'` loop inside `'OSMboxQuery()'` to inline code for speed. This eliminates the loop overhead.

OS_Q.C:

Added typecast to avoid compiler error/warning:

```
    pq = (OS_Q *)pevent->OSEventPtr;
         ^^^^^^^^
```

Affected functions:

```
     OSQAccept()
     OSQFlush()
     OSQPend()
     OSQPost()
     OSQPostFront()
```

Changed `for` loop inside `OSQQuery()` to inline code for speed. This eliminates the loop overhead.

Added `msg = (void *)0;` in `if (OSIntNesting > 0)` case.

OS_SEM.C:

Second `if` statement in function `OSSemPend()` needed to be and `if`/`else` clause.

OS_TASK.C:

Stack filling is now done using the ANSI C function `memset()` for speed.

Copying of the `OS_TCB` structure in `OSTaskQuery()` is now done using `memcpy()` for speed.

Function `OSTaskStkChk()` now cast the value `0` to `(OS_STK)0` in while loops.

uCOS_II.C:

Changed the comment for `OSTCBStkSize` in the `OS_TCB` structure to indicate that the size is in number of elements and not bytes.

The argument `stk_size` in `OSTCBInit()` has been changed from `INT16U` to `INT32U` to accommodate large stacks.