Micriμm

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

μC/OS-II

The Real-Time kernel

V2.85

Release Notes

© Copyright 2007, Micriμm
All Rights reserved

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

V2.85

(2007/06/15)

In this release, we made some minor changes and are summarized below:

Added `OS_APP_HOOKS_EN` in `OS_CFG.H` to allow µC/OS-II to call application define hook functions.

`msg` was changed to `pmsg`.

`err` was changed to `perr`.

`OS????NameGet()` and `OS????NameSet()` can no longer be called from ISRs

`OSTimeDly()` and `OSTimeDlyHMSM()` now contain checks to prevent them from being called from an ISR.

`OSMutexAccept()` now returns a `BOOLEAN` instead of an `INT8U`.

Hook functions in port files now should call application specific hooks.

Added new error codes.

UPGRADING TO V2.85

You should follow these steps in order to upgrade from a previous version to V2.85. Some of the items below are the same since V2.81 but are repeated here to make sure you are aware of them:

1) **Timer Manager:**

Timers **MUST** now be created by `OSTmrCreate()` before they can be used. In V2.82, a timer was created and started when you called `OSTmrStart()`. Now you **MUST** call `OSTmrCreate()` and then `OSTmrStart()` to create and start the timer, respectively.

It is now your responsibility to delete a timer when it is no longer being used.

The Timer Manager user-available functions are now:

```
OSTmrCreate()
OSTmrDel()
OSTmrNameGet()
OSTmrRemainGet()
OSTmrStart()
OSTmrStop()
```

To create and start a timer, you need to call `OSTmrCreate()` and then `OSTmrStart()`. When you are done using a timer, you can delete it by calling `OSTmrDel()`.

2) **TRUE and FALSE changed to OS_TRUE and OS_FALSE:**

µC/OS-II now uses and returns `OS_TRUE` and `OS_FALSE` instead of `TRUE` and `FALSE`. If you were using `TRUE` and `FALSE` in your application you will either need to define `TRUE` and `FALSE` yourself or change those to `OS_TRUE` and `OS_FALSE`.

3) **Create `APP_CFG.H`:**

As of V2.81, you need to create a file called `APP_CFG.H` which would reside in your project. `APP_CFG.H` is used to hold configuration information about your project. Specifically, we expect that you place task priorities, task stack sizes and other application related configuration information. The following page shows an example of the contents of `APP_CFG.H`.

4) **Include `OS_TMR.C` in your project:**

As of V2.81, you need to include `OS_TMR.C` in your builds in order to obtain the new services provided in `OS_TMR.C` and avoid compiler warnings/errors.

5) **New #defines are needed in `OS_CFG.H`:**
You will need to include the following #defines (they are found in `OS_CFG_R.H`, the reference file for `OS_CFG.H`). See also the configuration manual:

```
OS_APP_HOOKS_EN

OS_TMR_EN
OS_TMR_CFG_MAX
OS_TMR_CFG_NAME_SIZE
OS_TMR_CFG_WHEEL_SIZE
OS_TMR_CFG_WHEEL_SIZE

OS_MBOX_PEND_ABORT_EN
OS_Q_PEND_ABORT_EN
OS_SEM_PEND_ABORT_EN
```

6) **Add `OS_TASK_TMR_STK_SIZE`:**
If you use the timer manager, you will need to define the size of the timer task stack, i.e. `OS_TASK_TMR_STK_SIZE`. This is declared in your project's `OS_CFG.H`.

7) **Add `OS_TASK_TMR_PRIO`:**
If you use the timer manager, you will need to define the priority of the timer manager task, i.e. `OS_TASK_TMR_PRIO`. This is declared in your project's `APP_CFG.H`.

8) **Place prototypes in `OS_CPU.H`:**
As of V2.81, it's **IMPORTANT** that you place the prototypes for `OSCtxSw()`, `OSIntCtxSw()` and `OSStartHighRdy()` in `OS_CPU.H`. Typically, these functions would be prototyped as follows but, depending on the compiler, they may need to be different:

```
void  OSStartHighRdy(void);
void  OSIntCtxSw(void);
void  OSCtxSw(void);
```

9) **Start using `OS_ERR_???` as error return values**:
We recommend that you start using the new #define constants for error return values. All error return values start with `OS_ERR_` for consistency.

10) **`OS????NameGet()` and `OS????NameSet()` not callable from ISRs**:
Since `OS????NameGet()` and `OS????NameSet()` can no longer be called from ISRs, make sure your code didn't make use of those in ISRs.

11) **`OSMutexAccept()` returns a `BOOLEAN`**:

Since `OSMutexAccept()` now returns a `BOOLEAN` make sure you change your code accordingly.

CHANGES IN V2.85

ALL

We removed all checks for `OS_VERSION` in the code. The reason is that you should actually upgrade your application when you upgrade your version of µC/OS-II. The extra checks for `OS_VERSION` created 'pollution' in the code which was not deemed necessary.

Replaced the 'magic number' `(OS_TCB *)1` to `OS_TCB_RESERVED` when reserving a TCB.

`msg` was changed to `pmsg`. `err` was changed to `perr` to reflect that these are pointers.

os_core.c

`OSEventNameGet()` and `OSEventNameSet()` can no longer be called from an ISR. The reason is to keep ISRs as short as possible. Getting and setting ASCII names is performed through loops and could increase ISR times. This change should not cause backwards compatibility issues since names for OS objects are typically set once when the object is created. If you call `OSEventNameGet()` from an ISR, the function will not be performed and you will get an `OS_ERR_NAME_GET_ISR` error code. If you call `OSEventNameSet()` from an ISR, the function will not be performed and you will get an `OS_ERR_NAME_SET_ISR` error code.

os_flag.c

`OSFlagNameGet()` and `OSFlagNameSet()` can no longer be called from an ISR. The reason is to keep ISRs as short as possible. Getting and setting ASCII names is performed through loops and could increase ISR times. This change should not cause backwards compatibility issues since names for OS objects are typically set once when the object is created. If you call `OSFlagNameGet()` from an ISR, the function will not be performed and you will get an `OS_ERR_NAME_GET_ISR` error code. If you call `OSFlagNameSet()` from an ISR, the function will not be performed and you will get an `OS_ERR_NAME_SET_ISR` error code.

os_mem.c

In `OSMemCreate()` we no longer require that a memory block be a multiple of a pointer size. However, we still require that a memory block contains storage for at least one pointer.

`OSMemNameGet()` and `OSMemNameSet()` can no longer be called from an ISR. The reason is to keep ISRs as short as possible. Getting and setting ASCII names is performed through loops and could increase ISR times. This change

should not cause backwards compatibility issues since names for OS objects are typically set once when the object is created. If you call `OSMemNameGet()` from an ISR, the function will not be performed and you will get an `OS_ERR_NAME_GET_ISR` error code. If you call `OSMemNameSet()` from an ISR, the function will not be performed and you will get an `OS_ERR_NAME_SET_ISR` error code.

os_mutex.c

We now cast `OS_MUTEX_KEEP_LOWER_8`, `OS_MUTEX_KEEP_UPPER_8` and `OS_MUTEX_MUTEX_AVAILABLE` to (`INT16U`) to prevent compiler warnings.

`OSMutexAccept()` now returns `OS_TRUE` or `OS_FALSE` based on whether the resource was available or not, respectively.

`OSMutexQuery()` now sets the `.OSValue` field to `OS_TRUE` or `OS_FALSE` based on whether the resource was available or not, respectively.

os_task.c

`OSTaskNameGet()` and `OSTaskNameSet()` can no longer be called from an ISR. The reason is to keep ISRs as short as possible. Getting and setting ASCII names is performed through loops and could increase ISR times. This change should not cause backwards compatibility issues since names for OS objects are typically set once when the object is created. If you call `OSTaskNameGet()` from an ISR, the function will not be performed and you will get an `OS_ERR_NAME_GET_ISR` error code. If you call `OSTaskNameSet()` from an ISR, the function will not be performed and you will get an `OS_ERR_NAME_SET_ISR` error code.

os_time.c

Even though `OSTimeDly()` and `OSTimeDlyHMSM()` should never have been called from ISRs, there were no checks to that effect. This has been corrected and we now check that `OSIntNesting` is `0` in order to allow this function to execute. If called from an ISR, `OSTimeDlyHMSM()` will return an error code of `OS_ERR_TIME_DLY_ISR`.

os_tmr.c

`OSTmrNameGet()` can no longer be called from an ISR. The reason is to keep ISRs as short as possible. Getting and setting ASCII names is performed through loops and could increase ISR times. This change should not cause backwards compatibility issues since names for OS objects are typically set once when the object is created. If you call `OSTaskNameGet()` from an ISR, the function will not be performed and you will get an `OS_ERR_NAME_GET_ISR` error code.

OSTmrStop() now returns OS_TRUE when the timer is stopped even if you passed an invalid 'opt' argument or we used the callback argument specified in the call. In other words, if we stop the timer, we return OS_TRUE.

ucos_ii.h

Added OS_ERR_??? #define constants for consistency.

Use …	Instead of …	Value
OS_ERR_NONE	OS_NO_ERR	0
OS_ERR_TIMEOUT	OS_TIMEOUT	10
OS_ERR_TASK_NOT_EXIST	OS_TASK_NOT_EXIST	11
OS_ERR_NAME_GET_ISR		17
OS_ERR_NAME_SET_ISR		18
OS_ERR_MBOX_FULL	OS_MBOX_FULL	20
OS_ERR_Q_FULL	OS_Q_FULL	30
OS_ERR_Q_EMPTY	OS_Q_EMPTY	31
OS_ERR_PRIO_EXIST	OS_PRIO_EXIST	40
OS_ERR_PRIO	OS_PRIO_ERR	41
OS_ERR_PRIO_INVALID	OS_PRIO_INVALID	42
OS_ERR_SEM_OVF	OS_SEM_OVF	50
OS_ERR_TASK_DEL	OS_TASK_DEL_ERR	60
OS_ERR_TASK_DEL_IDLE	OS_TASK_DEL_IDLE	61
OS_ERR_TASK_DEL_REQ	OS_TASK_DEL_REQ	62
OS_ERR_TASK_DEL_ISR	OS_TASK_DEL_ISR	63
OS_ERR_NO_MORE_TCB	OS_NO_MORE_TCB	70
OS_ERR_TIME_NOT_DLY	OS_TIME_NOT_DLY	80
OS_ERR_TIME_INVALID_MINUTES	OS_TIME_INVALID_MINUTES	81
OS_ERR_TIME_INVALID_SECONDS	OS_TIME_INVALID_SECONDS	82
OS_ERR_TIME_INVALID_MS	OS_ERR_TIME_INVALID_MS	83
OS_ERR_TIME_ZERO_DLY	OS_ERR_TIME_ZERO_DLY	84
OS_ERR_TIME_DLY_ISR		85
OS_ERR_TASK_SUSPEND_PRIO	OS_ERR_TASK_SUSPEND_PRIO	90
OS_ERR_TASK_SUSPEND_IDLE	OS_ERR_TASK_SUSPEND_IDLE	91
OS_ERR_TASK_RESUME_PRIO	OS_ERR_TASK_RESUME_PRIO	100
OS_ERR_TASK_NOT_SUSPENDED	OS_ERR_TASK_NOT_SUSPENDED	101
OS_ERR_MEM_INVALID_PART	OS_ERR_MEM_INVALID_PART	110
OS_ERR_MEM_INVALID_BLKS	OS_ERR_MEM_INVALID_BLKS	111
OS_ERR_MEM_INVALID_SIZE	OS_ERR_MEM_INVALID_SIZE	112
OS_ERR_MEM_NO_FREE_BLKS	OS_ERR_MEM_NO_FREE_BLKS	113
OS_ERR_MEM_FULL	OS_ERR_MEM_FULL	114
OS_ERR_MEM_INVALID_PBLK	OS_ERR_MEM_INVALID_PBLK	115
OS_ERR_MEM_INVALID_PMEM	OS_ERR_MEM_INVALID_PMEM	116
OS_ERR_MEM_INVALID_PDATA	OS_ERR_MEM_INVALID_PDATA	117
OS_ERR_MEM_INVALID_ADDR	OS_ERR_MEM_INVALID_ADDR	118
OS_ERR_MEM_NAME_TOO_LONG	OS_ERR_MEM_NAME_TOO_LONG	119
OS_ERR_TASK_OPT	OS_ERR_TASK_OPT	130
OS_ERR_FLAG_INVALID_PGRP	OS_ERR_FLAG_INVALID_PGRP	150
OS_ERR_FLAG_WAIT_TYPE	OS_ERR_FLAG_WAIT_TYPE	151
OS_ERR_FLAG_NOT_RDY	OS_ERR_FLAG_NOT_RDY	152
OS_ERR_FLAG_INVALID_OPT	OS_ERR_FLAG_INVALID_OPT	153
OS_ERR_FLAG_GRP_DEPLETED	OS_ERR_FLAG_GRP_DEPLETED	154

The .OSValue field of OS_MUTEX_DATA is now a BOOLEAN instead of an INT8U.

OSMutexAccept() now returns a BOOLEAN.

Hook functions have been changed in the **ports** as follows. In other words, we now assume that hook functions are declared in application code INSTEAD of port code. Hooks are thus enabled when `OS_APP_HOOKS_EN > 0` in `OS_CFG.H`.

In Hook …	Changed From …	To …
`OSTaskCreateHook()`	`OSView_TaskCreateHook()`	`App_TaskCreateHook()`
`OSTaskDelHook()`		`App_TaskDelHook()`
`OSTaskIdleHook()`		`App_TaskIdleHook()`
`OSTaskStatHook()`		`App_TaskStatHook()`
`OSTaskSwHook()`	`OSView_TaskSwHook()`	`App_TaskSwHook()`
`OSTCBInitHook()`		`App_TCBInitHook()`
`OSTimeTickHook()`	`OSView_TickHook()`	`App_TimeTickHook()`

Added a '`SAFETY CRITICAL USE`' section to detect configuration issues when performing safety critical tests. This section does not generate any code and is thus harmless.

V2.84

(2007/01/31)

In this release, we added new functionality to Mailbox, Queue and Semaphore management.

We also added new #define constants for error return values. For example, you can now use `OS_ERR_TIMEOUT` instead of `OS_TIMEOUT`. In fact, we added `OS_ERR_???` for consistency and you should now always use `OS_ERR_???` when checking for error codes. Note that you can still use the previous #define values since those were kept for backwards compatibility. However, we might remove those is a future version.

UPGRADING TO V2.84

You should follow these steps in order to upgrade from a previous version to V2.84. Some of the items below are the same since V2.81 but are repeated here to make sure you are aware of them:

1) **Timer Manager:**

Timers **MUST** now be created by `OSTmrCreate()` before they can be used. In V2.82, a timer was created and started when you called `OSTmrStart()`. Now you **MUST** call `OSTmrCreate()` and then `OSTmrStart()` to create and start the timer, respectively.

It is now your responsibility to delete a timer when it is no longer being used.

The Timer Manager user-available functions are now:

```
OSTmrCreate()
OSTmrDel()
OSTmrNameGet()
OSTmrRemainGet()
OSTmrStart()
OSTmrStop()
```

To create and start a timer, you need to call `OSTmrCreate()` and then `OSTmrStart()`. When you are done using a timer, you can delete it by calling `OSTmrDel()`.

2) **TRUE and FALSE changed to OS_TRUE and OS_FALSE:**

µC/OS-II now uses and returns OS_TRUE and OS_FALSE instead of TRUE and FALSE. If you were using TRUE and FALSE in your application you will either need to define TRUE and FALSE yourself or change those to OS_TRUE and OS_FALSE.

3) **Create APP_CFG.H:**

As of V2.81, you need to create a file called APP_CFG.H which would reside in your project. APP_CFG.H is used to hold configuration information about your project. Specifically, we expect that you place task priorities, task stack sizes and other application related configuration information. The following page shows an example of the contents of APP_CFG.H.

4) **Include OS_TMR.C in your project:**

As of V2.81, you need to include OS_TMR.C in your builds in order to obtain the new services provided in OS_TMR.C and avoid compiler warnings/errors.

5) **New #defines are needed in OS_CFG.H:**

You will need to include the following #defines (they are found in OS_CFG_R.H, the reference file for OS_CFG.H). See also the configuration manual:

```
OS_TMR_EN
OS_TMR_CFG_MAX
OS_TMR_CFG_NAME_SIZE
OS_TMR_CFG_WHEEL_SIZE
OS_TMR_CFG_WHEEL_SIZE

OS_MBOX_PEND_ABORT_EN
OS_Q_PEND_ABORT_EN
OS_SEM_PEND_ABORT_EN
```

6) **Add OS_TASK_TMR_STK_SIZE:**

If you use the timer manager, you will need to define the size of the timer task stack, i.e. OS_TASK_TMR_STK_SIZE. This is declared in your project's OS_CFG.H.

7) **Add OS_TASK_TMR_PRIO:**

If you use the timer manager, you will need to define the priority of the timer manager task, i.e. OS_TASK_TMR_PRIO. This is declared in your project's APP_CFG.H.

8) **Place prototypes in OS_CPU.H:**

As of V2.81, it's **IMPORTANT** that you place the prototypes for OSCtxSw(), OSIntCtxSw() and OSStartHighRdy() in OS_CPU.H. Typically, these

functions would be prototyped as follows but, depending on the compiler, they may need to be different:

```
void  OSStartHighRdy(void);
void  OSIntCtxSw(void);
void  OSCtxSw(void);
```

9) **Start using `OS_ERR_???` as error return values**:
We recommend that you start using the new #define constants for error return values. All error return values start with `OS_ERR_` for consistency.

FIXED BUGS IN V2.83

`os_task.c`

Corrected a subtle bug in `OSTaskChangePrio()`. We now check if an event control block exist after readying a task at the new priority.

`os_tmr.c`

`OSTmr_Unlock()` was missing in a couple of places in `OSTmrStop()`.

CHANGES IN V2.84

os_core.c

We are now locking the scheduler when in `OSTimeTick()` instead of disabling interrupts. This reduces interrupt latency when this function is called.

ucos_ii.h

Added `OS_ERR_???` #define constants for consistency. It turns out that not all error return values started with the prefix `OS_ERR_`. To correct this, new #define constants have been added. It's highly recommended that you start using the new `OS_ERR_???` error codes instead of their previous counterparts, see table below.

Use …	Instead of …	Value
OS_ERR_NONE	OS_NO_ERR	0
OS_ERR_TIMEOUT	OS_TIMEOUT	10
OS_ERR_TASK_NOT_EXIST	OS_TASK_NOT_EXIST	11
OS_ERR_MBOX_FULL	OS_MBOX_FULL	20
OS_ERR_Q_FULL	OS_Q_FULL	30
OS_ERR_Q_EMPTY	OS_Q_EMPTY	31
OS_ERR_PRIO_EXIST	OS_PRIO_EXIST	40
OS_ERR_PRIO	OS_PRIO_ERR	41
OS_ERR_PRIO_INVALID	OS_PRIO_INVALID	42
OS_ERR_SEM_OVF	OS_SEM_OVF	50
OS_ERR_TASK_DEL	OS_TASK_DEL_ERR	60
OS_ERR_TASK_DEL_IDLE	OS_TASK_DEL_IDLE	61
OS_ERR_TASK_DEL_REQ	OS_TASK_DEL_REQ	62
OS_ERR_TASK_DEL_ISR	OS_TASK_DEL_ISR	63
OS_ERR_NO_MORE_TCB	OS_NO_MORE_TCB	70
OS_ERR_TIME_NOT_DLY	OS_TIME_NOT_DLY	80
OS_ERR_TIME_INVALID_MINUTES	OS_TIME_INVALID_MINUTES	81
OS_ERR_TIME_INVALID_SECONDS	OS_TIME_INVALID_SECONDS	82
OS_ERR_TIME_INVALID_MS	OS_ERR_TIME_INVALID_MS	83
OS_ERR_TIME_ZERO_DLY	OS_ERR_TIME_ZERO_DLY	84
OS_ERR_TASK_SUSPEND_PRIO	OS_ERR_TASK_SUSPEND_PRIO	90
OS_ERR_TASK_SUSPEND_IDLE	OS_ERR_TASK_SUSPEND_IDLE	91
OS_ERR_TASK_RESUME_PRIO	OS_ERR_TASK_RESUME_PRIO	100
OS_ERR_TASK_NOT_SUSPENDED	OS_ERR_TASK_NOT_SUSPENDED	101
OS_ERR_MEM_INVALID_PART	OS_ERR_MEM_INVALID_PART	110
OS_ERR_MEM_INVALID_BLKS	OS_ERR_MEM_INVALID_BLKS	111
OS_ERR_MEM_INVALID_SIZE	OS_ERR_MEM_INVALID_SIZE	112
OS_ERR_MEM_NO_FREE_BLKS	OS_ERR_MEM_NO_FREE_BLKS	113
OS_ERR_MEM_FULL	OS_ERR_MEM_FULL	114
OS_ERR_MEM_INVALID_PBLK	OS_ERR_MEM_INVALID_PBLK	115
OS_ERR_MEM_INVALID_PMEM	OS_ERR_MEM_INVALID_PMEM	116
OS_ERR_MEM_INVALID_PDATA	OS_ERR_MEM_INVALID_PDATA	117
OS_ERR_MEM_INVALID_ADDR	OS_ERR_MEM_INVALID_ADDR	118
OS_ERR_MEM_NAME_TOO_LONG	OS_ERR_MEM_NAME_TOO_LONG	119
OS_ERR_TASK_OPT	OS_ERR_TASK_OPT	130
OS_ERR_FLAG_INVALID_PGRP	OS_ERR_FLAG_INVALID_PGRP	150
OS_ERR_FLAG_WAIT_TYPE	OS_ERR_FLAG_WAIT_TYPE	151
OS_ERR_FLAG_NOT_RDY	OS_ERR_FLAG_NOT_RDY	152
OS_ERR_FLAG_INVALID_OPT	OS_ERR_FLAG_INVALID_OPT	153
OS_ERR_FLAG_GRP_DEPLETED	OS_ERR_FLAG_GRP_DEPLETED	154

V2.83

(2006/06/02)

In this release, we made significant changes to the timer manager module. Please consult the Reference Manual for the new APIs of functions `OSTmrCreate()`, `OSTmrDel()`, `OSTmrStop()` and `OSTmrStart()`.

UPGRADING TO V2.83

You should follow these steps in order to upgrade from a previous version to V2.83. Some of the items below are from V2.81 and V2.82 but are repeated here to make sure you are aware of them:

1) **Timer Manager:**

Timers **MUST** now be created by `OSTmrCreate()` before they can be used. In V2.82, a timer was created and started when you called `OSTmrStart()`. Now you **MUST** call `OSTmrCreate()` and then `OSTmrStart()` to create and start the timer, respectively.

It is now your responsibility to delete a timer when it is no longer being used.

The Timer Manager user-available functions are now:

```
OSTmrCreate()
OSTmrDel()
OSTmrNameGet()
OSTmrRemainGet()
OSTmrStart()
OSTmrStop()
```

To create and start a timer, you need to call `OSTmrCreate()` and then `OSTmrStart()`. When you are done using a timer, you can delete it by calling `OSTmrDel()`.

2) **TRUE and FALSE changed to OS_TRUE and OS_FALSE:**

µC/OS-II now uses and returns `OS_TRUE` and `OS_FALSE` instead of `TRUE` and `FALSE`. If you were using `TRUE` and `FALSE` in your application you will either need to define `TRUE` and `FALSE` yourself or change those to `OS_TRUE` and `OS_FALSE`.

3) **Create `APP_CFG.H`:**

As of V2.81, you need to create a file called `APP_CFG.H` which would reside in your project. `APP_CFG.H` is used to hold configuration information about your project. Specifically, we expect that you place task priorities, task stack sizes and other application related configuration information. The following page shows an example of the contents of `APP_CFG.H`.

4) **Include `OS_TMR.C` in your project:**

As of V2.81, you need to include `OS_TMR.C` in your builds in order to obtain the new services provided in `OS_TMR.C` and avoid compiler warnings/errors.

5) **New #defines are needed in `OS_CFG.H`:**

You will need to include the following #defines (they are found in `OS_CFG_R.H`, the reference file for `OS_CFG.H`). See also the configuration manual:

```
OS_TMR_EN
OS_TMR_CFG_MAX
OS_TMR_CFG_NAME_SIZE
OS_TMR_CFG_WHEEL_SIZE
OS_TMR_CFG_WHEEL_SIZE
```

6) **Add `OS_TASK_TMR_STK_SIZE`:**

If you use the timer manager, you will need to define the size of the timer task stack, i.e. `OS_TASK_TMR_STK_SIZE`. This is declared in your project's `OS_CFG.H`.

7) **Add `OS_TASK_TMR_PRIO`:**

If you use the timer manager, you will need to define the priority of the timer manager task, i.e. `OS_TASK_TMR_PRIO`. This is declared in your project's `APP_CFG.H`.

8) **Place prototypes in `OS_CPU.H`:**

As of V2.81, it's **IMPORTANT** that you place the prototypes for `OSCtxSw()`, `OSIntCtxSw()` and `OSStartHighRdy()` in `OS_CPU.H`. Typically, these functions would be prototyped as follows but, depending on the compiler, they may need to be different:

```
void  OSStartHighRdy(void);
void  OSIntCtxSw(void);
void  OSCtxSw(void);
```

FIXED BUGS IN V2.82

os_tmr.c

You could not call `OSTmrNameGet()` and `OSTmrRemainGet()` when a timer was in one-shot mode and the timer expired because the timer was automatically deleted. This has now been fixed because timers are created and deleted by the user.

CHANGES IN V2.83

os_tmr.c

When a timer times out, it will no longer be deleted. In other words, it is now your responsibility to delete unused timers.

`OSTmrStop()` no longer deletes the timer.

You can now safely call `OSTmrRemainGet()` and `OSTmrNameGet()` whenever a timer is created until it gets deleted. In V2.81 and V2.82, you could not use these functions when a timer was configured in one-shot mode.

We added an entry in the `OS_TMR` data structure to allow us to verify that you are passing a pointer to an `OS_TMR` structure when you call timer manager services.

`OSTmrStart()` now ONLY starts (or restarts) a timer and does NOT create a timer. A timer must now be created before it can be started.

You must call `OSTmrDel()` to delete any unused timers.

Added `OSTmrStateGet()` which returns the state of a timer.

V2.82

(2006/03/24)

This is a minor release. However, we change the name of two (2) API calls: `OSTmrGetName()`, `OSTmrGetRemain()` and, we added an argument to `OSTmrStart()`.

UPGRADING TO V2.82

You should follow these steps in order to upgrade from a previous version to V2.82. Some of the items below are from V2.81 but are repeated here to make sure you are aware of them:

1) **TRUE and FALSE changed to OS_TRUE and OS_FALSE:**
µC/OS-II now uses and returns `OS_TRUE` and `OS_FALSE` instead of `TRUE` and `FALSE`. If you were using `TRUE` and `FALSE` in your application you will either need to define `TRUE` and `FALSE` yourself or change those to `OS_TRUE` and `OS_FALSE`.

2) **Create `APP_CFG.H`:**
As of V2.81, you need to create a file called `APP_CFG.H` which would reside in your project. `APP_CFG.H` is used to hold configuration information about your project. Specifically, we expect that you place task priorities, task stack sizes and other application related configuration information. The following page shows an example of the contents of `APP_CFG.H`.

3) **Include `OS_TMR.C` in your project:**
As of V2.81, you need to include `OS_TMR.C` in your builds in order to obtain the new services provided in `OS_TMR.C` and avoid compiler warnings/errors.

4) **Place prototypes in `OS_CPU.H`:**
As of V2.81, it's **IMPORTANT** that you place the prototypes for `OSCtxSw()`, `OSIntCtxSw()` and `OSStartHighRdy()` in `OS_CPU.H`. Typically, these functions would be prototyped as follows but, depending on the compiler, they may need to be different:

```
void  OSStartHighRdy(void);
void  OSIntCtxSw(void);
void  OSCtxSw(void);
```

Example APP_CFG.H

```
/*
*********************************************************************************************************
*                                            TASK PRIORITIES
*********************************************************************************************************
*/

#define  APP_TASK_START_PRIO          4

#define  OS_TASK_TMR_PRIO             5

#define  APP_TASK_USER_IF_PRIO        6

#define  APP_TASK_KBD_PRIO            7

#define  CLK_TASK_PRIO                8        /* This defines the priority of ClkTask()              */


#define  OS_VIEW_TASK_PRIO            9
#define  OS_VIEW_TASK_ID              9

/*
*********************************************************************************************************
*                                            TASK STACK SIZES
*********************************************************************************************************
*/

#define  APP_TASK_START_STK_SIZE    100
#define  APP_TASK_USER_IF_STK_SIZE  100
#define  APP_TASK_KBD_STK_SIZE      100

#define  CLK_TASK_STK_SIZE          100        /* Stack size in OS_STK for ClkTask_OS()               */

#define  OS_VIEW_TASK_STK_SIZE      100

/*
*********************************************************************************************************
*                                          uC/Clk CONFIGURATION
*********************************************************************************************************
*/

#define  CLK_TS_BASE_YEAR          2005        /* Time stamps start year                              */

#define  CLK_DLY_TICKS       OS_TICKS_PER_SEC  /* # of clock ticks to obtain 1 second                 */

#define  CLK_DATE_EN                  1        /* Enable DATE (when 1)                                */
#define  CLK_TS_EN                    1        /* Enable TIME-STAMPS (when 1)                         */
#define  CLK_USE_DLY                  1        /* Task will use OSTimeDly() instead of pend on sem.   */

/*
*********************************************************************************************************
*                                          uC/LCD CONFIGURATION
*********************************************************************************************************
*/

#define  DISP_BUS_WIDTH               4        /* LCD controller is accessed with a 4 bit bus         */

/*
*********************************************************************************************************
*                                        uC/OS-View CONFIGURATION
*********************************************************************************************************
*/

#define  OS_VIEW_PARSE_TASK           1        /* Parsing of received packets will be done by a task  */

#define  OS_VIEW_TMR_32_BITS          1        /* uC/OS-View timer is 32 bits                         */

#define  OS_VIEW_UART_0               0
#define  OS_VIEW_UART_1               1
#define  OS_VIEW_UART_2               2

#define  OS_VIEW_COMM_SEL             OS_VIEW_UART_1
```

FIXED BUGS IN V2.81

Fixed an error in `OSMutexDel()` (see `OS_MUTEX.C` below).

CHANGES IN V2.82

`Miscellaneous:`

Changed `TRUE` and `FALSE` to `OS_TRUE` and `OS_FALSE`. µC/OS-II should not be dictating the value of `TRUE` and `FALSE`.

`os_dbg_r.c`

Added new constants to monitor the size of some variables and data structures, specifically related to the new Timer management module introduced in V2.81.

`os_flag.c`

Added a check in `OSFlagPend()` to ensure that this function is not called from an ISR. Note that the documentation clearly warned about this but, we added the code just to be sure.

`os_mbox.c`

Added a check in `OSMboxPend()` to ensure that this function is not called from an ISR. Note that the documentation clearly warned about this but, we added the code just to be sure.

In `OSMboxPostOpt()` we added a new option called `OS_POST_OPT_NO_SCHED` which, when set, indicates that you do not want `OSMboxPostOpt()` to call the scheduler when you have completed the post.

`os_mutex.c`

Added a check in `OSMutexDel()` and the `OS_DEL_ALWAYS` case to make the owner of the mutex ready-to-run (if there was an owner). Because of some code similarities found in `OSMutexPost()`, we created the local function called `OSMutex_RdyAtPrio()` to perform this operation and thus not increase the code by too much.

Added a check in `OSMutexPend()` to ensure that this function is not called from an ISR. Note that the documentation clearly warned about this but, we added the code just to be sure.

os_q.c

Added a check in `OSQPend()` to ensure that this function is not called from an ISR. Note that the documentation clearly warned about this but, we added the code just to be sure.

In `OSQPostOpt()` we added a new option called `OS_POST_OPT_NO_SCHED` which, when set, indicates that you do not want `OSQPostOpt()` to call the scheduler when you have completed the post.

os_sem.c

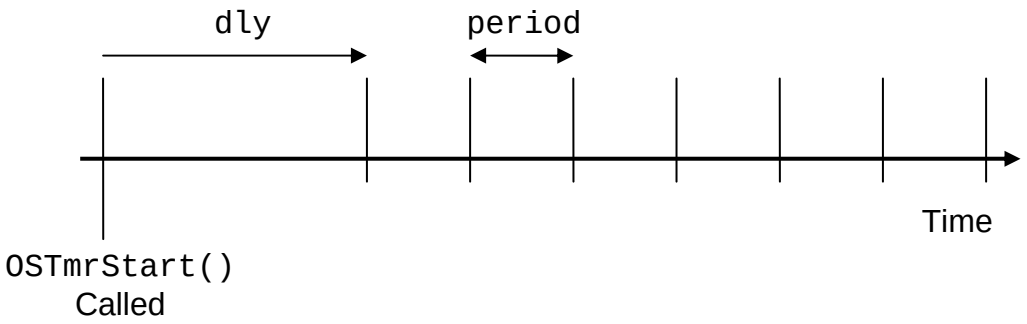
Added a check in `OSSemPend()` to ensure that this function is not called from an ISR. Note that the documentation clearly warned about this but, we added the code just to be sure.

os_tmr.c

Changed the name of `OSTmrGetName()` to `OSTmrNameGet()` to be consistent with other similar services.

Changed the name of `OSTmrGetRemain()` to `OSTmrRemainGet()` to be consistent with other similar services.

Added an argument (`dly`) to `OSTmrStart()`. This, of course, will make the compiler issue an error if you previously used the timer manager in V2.81. The argument specifies an initial delay before the timer enters periodic mode (see drawing below). If you set the `dly` to the same value as the period then you will obtain the same result as in V2.81. If you specify a `dly` of `0`, `period` will be used as the initial delay:



ucos_ii.h

Added `OS_POST_OPT_NO_SCHED` and its value is `0x04`.

Added `.OSTmrDly` in the `OS_TMR` data structure

V2.81

(2005/09/06)

µC/OS-II now provides support for periodic as well as one-shot timers. This functionality is found in OS_TMR.C. For more information about this new feature, consult the "*New Features and Services since V2.00*" document.

UPGRADING TO V2.81

You should follow these steps in order to upgrade from a previous version to V2.81:

1) **Copy OS_CFG_R.H to your project's OS_CFG.H:**
OS_CFG_R.H is a 'reference' file containing all the #define constants expected by µC/OS-II. Copying OS_CFG_R.H to your project's OS_CFG.H will ensure that you are not forgetting or mistyping anything.

2) **Create APP_CFG.H:**
As of V2.81, you need to create a file called APP_CFG.H which would reside in your project. APP_CFG.H is used to hold configuration information about your project. Specifically, we expect that you place task priorities, task stack sizes and other application related configuration information. The following page shows an example of the contents of APP_CFG.H.

3) **Include OS_TMR.C in your project:**
In order to obtain the new services provided in OS_TMR.C, you will need to make sure that you include OS_TMR.C in your build file(s).

4) **Place prototypes in OS_CPU.H:**
It's **IMPORTANT** that you place the prototypes for **OSCtxSw()**, **OSIntCtxSw()** and **OSStartHighRdy()** in **OS_CPU.H**. Typically, these functions would be prototyped as follows but, depending on the compiler, they may need to be different:

```
void  OSStartHighRdy(void);
void  OSIntCtxSw(void);
void  OSCtxSw(void);
```

Example APP_CFG.H

```
/*
*********************************************************************************************************
*                                            TASK PRIORITIES
*********************************************************************************************************
*/

#define  APP_TASK_START_PRIO          4

#define  OS_TASK_TMR_PRIO             5

#define  APP_TASK_USER_IF_PRIO        6

#define  APP_TASK_KBD_PRIO            7

#define  CLK_TASK_PRIO                8        /* This defines the priority of ClkTask()              */


#define  OS_VIEW_TASK_PRIO            9
#define  OS_VIEW_TASK_ID              9

/*
*********************************************************************************************************
*                                            TASK STACK SIZES
*********************************************************************************************************
*/

#define  APP_TASK_START_STK_SIZE    100
#define  APP_TASK_USER_IF_STK_SIZE  100
#define  APP_TASK_KBD_STK_SIZE      100

#define  CLK_TASK_STK_SIZE          100        /* Stack size in OS_STK for ClkTask_OS()               */

#define  OS_VIEW_TASK_STK_SIZE      100

/*
*********************************************************************************************************
*                                          uC/Clk CONFIGURATION
*********************************************************************************************************
*/

#define  CLK_TS_BASE_YEAR          2005        /* Time stamps start year                              */

#define  CLK_DLY_TICKS       OS_TICKS_PER_SEC  /* # of clock ticks to obtain 1 second                 */

#define  CLK_DATE_EN                  1        /* Enable DATE (when 1)                                */
#define  CLK_TS_EN                    1        /* Enable TIME-STAMPS (when 1)                         */
#define  CLK_USE_DLY                  1        /* Task will use OSTimeDly() instead of pend on sem.   */

/*
*********************************************************************************************************
*                                          uC/LCD CONFIGURATION
*********************************************************************************************************
*/

#define  DISP_BUS_WIDTH               4        /* LCD controller is accessed with a 4 bit bus         */

/*
*********************************************************************************************************
*                                        uC/OS-View CONFIGURATION
*********************************************************************************************************
*/

#define  OS_VIEW_PARSE_TASK           1        /* Parsing of received packets will be done by a task  */

#define  OS_VIEW_TMR_32_BITS          1        /* uC/OS-View timer is 32 bits                         */

#define  OS_VIEW_UART_0               0
#define  OS_VIEW_UART_1               1
#define  OS_VIEW_UART_2               2

#define  OS_VIEW_COMM_SEL             OS_VIEW_UART_1
```

FIXED BUGS IN V2.80

Fixed a number of errors introduced when we increased the number of task to 255.

CHANGES IN V2.81

`os_cfg.h` (see template in `os_cfg_r.h`)

Re-arranged the order of #defines in this file.

Added a number of `#define` constants to support timer management:

```
OS_ISR_PROTO_EXT

OS_TMR_EN
OS_TMR_CFG_MAX
OS_TMR_CFG_WHEEL_SIZE
OS_TMR_CFG_NAME_SIZE
OS_TMR_CFG_TICKS_PER_SEC
OS_TASK_TMR_STK_SIZE
```

`os_core.c`

Added call to `OSTmr_Init()`.

`ucos_ii.h`

Added `OS_TASK_TMR_ID` and its value is `65533`.

Changed `OS_IDLE_PRIO` to `OS_TASK_IDLE_PRIO`

Changed `OS_STAT_PRIO` to `OS_TASK_STAT_PRIO`

Added `OS_ERR_TMR_???` and `OS_TMR_OPT_???`.

Added the `OS_TMR`, `OS_TMR_WHEEL` and `OS_TMR_CALLBACK` data types needed to support timer management.

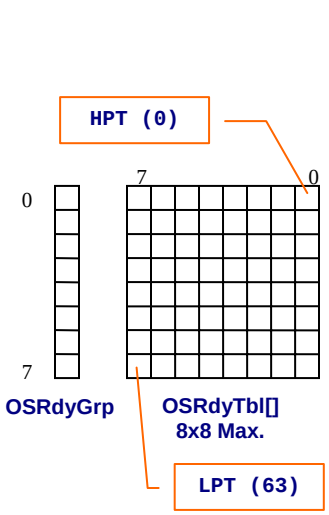
V2.80

(2005/03/21)

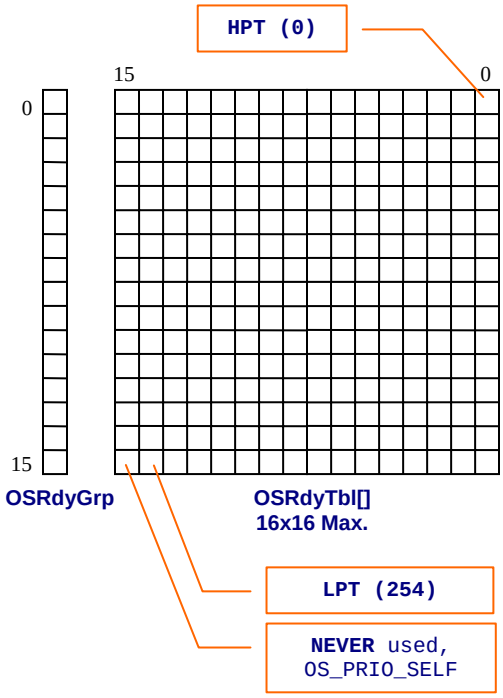
This is a big release because µC/OS-II now supports up to **255** tasks.

We also made a number of minor changes related to MISRA C rules.

To support up to 255 tasks, we simply increased the ready list and event wait lists to a matrix of 16x16 instead of 8x8. In fact, the actual size of the matrix depends on the value of `OS_LOWEST_PRIO` in `OS_CFG.H`. If `OS_LOWEST_PRIO` is less than or equal to 63, we use an 8x8 matrix and thus µC/OS-II behaves exactly the same as it used to. If you specify a value for `OS_LOWEST_PRIO` to be greater than 63, we use the 16x16 matrix as show below.



OS_LOWEST_PRIO <= 63



OS_LOWEST_PRIO > 63

You should note that the actual size of the matrix depends on `OS_LOWEST_PRIO`. For example, if `OS_LOWEST_PRIO` is 10 then the matrix is actually 2x8 (two bytes of 8 bits). Similarly, if `OS_LOWEST_PRIO` is set to 47, the matrix will be 6x8. When `OS_LOWEST_PRIO` is above 63, we use 16-bit wide entries. For example, if you specify `OS_LOWEST_PRIO` to be 100 then the matrix will be 7x16 (7 entries of 16 bits each). You **CANNOT** have `OS_LOWEST_PRIO` at 255 because that value is reserved for `OS_PRIO_SELF`.

FIXED BUGS IN V2.77

No bugs were reported in V2.77.

CHANGES IN V2.80

OS_CFG.H (see template in OS_CFG_R.H)

OS_LOWEST_PRIO in OS_CFG.H can now be up to 254 thus supporting up to 255 tasks (including the idle task).

You now need to add the #define OS_FLAGS_NBITS which **MUST** be either 8, 16 or 32. This #define defines the number of bits used for event flags.

We **REMOVED** the type definition of OS_FLAGS and thus, you will also have to remove it in your OS_CFG.H file.

OS_CORE.C

We removed the OSMapTbl[] and replaced its use in the code with a 1 << n operation.

Added a new function called OS_SchedNew() to find the new highest priority task ready-to-run. In other words, this function determines the value of the variable OSPrioHighRdy. OS_SchedNew() is called by OS_Sched(), OSIntExit() and OSStart().

ucos_ii.h

Moved the #define OS_VERSION before the #include statements of OS_CFG.H and OS_CPU.H to allow these files to have definitions based on which version of µC/OS-II.

Added OS_TASK_OPT_NONE to allow this to be used in OSTaskCreateExt() instead of 0.

GENERAL

Functions that used char now use INT8U to satisfy one of the MISRA C rules.

V2.77

(2004/11/29)

This release corrects a number of very minor issues with V2.76.

FIXED BUGS IN V2.76

Bug V2.76-001:

There were a number of typos and incorrect comments that were fixed.

CHANGES IN V2.77

V2.77 adds a few minor enhancements to V2.76. However, none of these enhancements were critical.

IMPORTANT

The prototypes for **`OSStartHighRdy()`**, **`OSCtxSw()`** and **`OSIntCtxSw()`** are NOW assumed to be placed in **`OS_CPU.H`** since they have been removed from `ucos_ii.h`. The reason this was done was to allow different declarations for these functions. For example, with the IAR ARM compiler, these functions are declared as follows:

```
__arm  void  OSStartHighRdy(void);
__arm  void  OSCtxSw(void);
__arm  void  OSIntCtxSw(void);
```

The 'standard' declarations should be:

```
void  OSStartHighRdy(void);
void  OSCtxSw(void);
void  OSIntCtxSw(void);
```

Please add these prototypes in **YOUR** `os_cpu.h` file.

OS_CFG.H

We now expect the presence of `OS_VIEW_MODULE` in your OS configuration file. This is such that you can more easily add µC/OS-View to your product. Defining `OS_VIEW_MODULE` to `1` indicates that you will include µC/OS-View in your product's build. Setting `OS_VIEW_MODULE` to `0` indicates that you will not be using µC/OS-View.

If you DO NOT add this `#define`, the compiler will complain via a `#error` directive that we added in `ucos_ii.h`.

OS_CORE.C

We now assign a name to the µC/OS-II idle task and statistics task if `OS_TASK_NAME_SIZE` is defined as being greater than `14` in `OS_CFG.H`. This is used for debugging purposes. The idle task is called: `"uC/OS-II Idle"` and the statistics task is called `"uC/OS-II Stat"`.

GENERAL

In ALL the functions that pass `*err` so than an error code is returned to the caller, `err` is checked to make sure it's not a `NULL` pointer. The function returns if it is. Unfortunately, you are not told why because we have no way to give you an error code.

In ALL the functions that pass a pointer, we now check to make sure that the pointer is not a `NULL` pointer. This was previously done for some of the pointers but not all.

V2.76

(2004/02/06)

This release corrects a number of minor issues with V2.75 and also add a new Semaphore interface function (`OSSemSet()`).

FIXED BUGS IN V2.75

Bug V2.75-001:

`OSTaskDlyResume()` makes the same test as the new `OSTimeTick()` in that if a task was delayed and was pending on an event then, `.OSTCBPendTO` will be set to `TRUE` indicating that the task timed out.

Bug V2.75-002:

The following functions:

```
OSTaskChangePrio()
OSTaskDel()
OSTaskDelReq()
OSTaskNameSet()
OSTaskNameGet()
OSTaskResume()
OSTaskSuspend()
```

All needed to check for '`ptcb`' pointing to `(void *)1` in case the task was assigned to a Mutex PIP.

Bug V2.75-003:

`OSTaskDelReq()` had a local variable 'stat' which was declared as a `BOOLEAN` but was in fact used as an 8 bit integer. This local variable is now an `INT8U`.

NEW FEATURE

V2.76 adds a new semaphore function (`OSSemSet()`) that allows you to set the value (i.e. count) of the semaphore. This new feature is useful when you use semaphores as a signaling mechanism. You enable this function by setting `OS_SEM_SET_EN` to `1` in `OS_CFG.H` of your product. See details about this function in the reference manual.

V2.75

(2003/12/15)

This release corrects a number of issues that were reported by users of V2.70. This release also contains some changes. Probably the most significant improvement is that we made sure that µC/OS-II passes LINT without warnings and errors. PC Lint V8 by Gimpel Software was used to LINT µC/OS-II: http://www.gimpel.com/html/contact.htm.

FIXED BUGS IN V2.70

Bug V2.70-001:

`OSTaskSuspend()` and `OSTaskResume()` bug has been corrected. The problem and correction are described later.

Bug V2.70-002:

In `OSMemNameSet()`, a return statement was missing for the case when `pmem` is `NULL`. This bug has been corrected.

Bug V2.70-003:

In `OSQPostOpt()`, the test for `msg` being `NULL` must be deleted. This is because, as of V2.62, it's now possible to post `NULL` pointer messages to a message queue. This has been corrected.

Bug V2.70-004:

In `OSQDel()`, the first test should return `pevent` instead of a `NULL` pointer upon failure. This bug has been corrected.

Bug V2.70-005:

`OSTaskNameGet()` and `OSTaskNameSet()` were missing an `OS_EXIT_CRITICAL()` before the exit of the first test. This has been corrected.

Bug V2.70-006:

In `OSFlagPend()` the returned `flags_rdy` was not set correctly if you didn't specify `OS_FLAG_CONSUME`. This has been corrected.

Bug V2.70-007:

In `OSTaskQuery()` we needed to check to see if the TCB was assigned to a Mutex. An additional test has thus been added to correct the problem.

Bug V2.70-008:

In `OSMutexPend()` we removed a `&=` statement in an `if` statement for MISRA compliance.

Bug V2.70-009:

In `ucos_ii.h` we tested for `OS_MAX_EVENTS >= 256` when it should have been testing for `>= 65500`.

Bug V2.70-010:

Added test for `OS_ARG_CHK_EN` in `OSTimeDlyHMSM()`.

CHANGES IN V2.75

V2.75-001

We added a version number at the top of each file in the main comment header.

V2.75-002

`ucos_ii.h` now includes `os_cfg.h` and `os_cpu.h`. This allows you to compile µC/OS-II with only those three headers.

V2.75-003

Changed the data type for the variable `i` in `OS_InitRdyList()` from `INT16U` to `INT8U`.

V2.75-004

Added `cpu_sr = 0` in all the functions that need to use `OS_ENTER_CRITICAL()` and `OS_EXIT_CRITICAL()`. This is done because some compilers generate warnings when the variable is not directly referenced in the code because it's buried inside a macro. We could have used `cpu_sr = cpu_sr` but, LINT complained about the fact that `cpu_sr` is being assigned a value that has not been initialized.

V2.75-005

Removed the global variable `OSIntExitY` in `OSIntExit()` and replaced it with a local variable called `y`. Note that you will need to delete the line:

```
+ sizeof(OSIntExitY)
```

in the file `os_dbg.c` of your port file (if this file exits in your port).

IMPORTANT

If you use an OLD µC/OS-II port, you might need to adjust the constant to add to the `SP` (Stack Pointer) in `OSIntCtxSw()`. In other words, if you use a port that adjust the `SP` in `OSIntCtxSw()`, you might need to adjust the constant because your port will NOT WORK. If your port uses the new scheme outline in the hardcover edition of the µC/OS-II book, you will not have to do anything as your port will work just fine.

V2.75-006

Added a flag in `OS_TCB` (called `.OSTCBPendTO`) that indicates whether a 'pend' call has timed out or not. The addition of this flag was necessary to fix bug V2.70-001. Details about the changes are described on the next page.

V2.75-007

Added a test in `OSTaskCreate()` and `OSTaskCreateExt()` to prevent calling these functions from an ISR.

V2.75-008

Added a `(void)` in front of `OS_FlagTaskRdy()` in `OS_FLAG.C` and in front of `OS_EventTaskRdy()` in `OS_MBOX.C`, `OS_Q.C`, `OS_SEM.C` and `OS_MUTEX.C` because the return value is not being used. This was done to prevent LINT warnings.

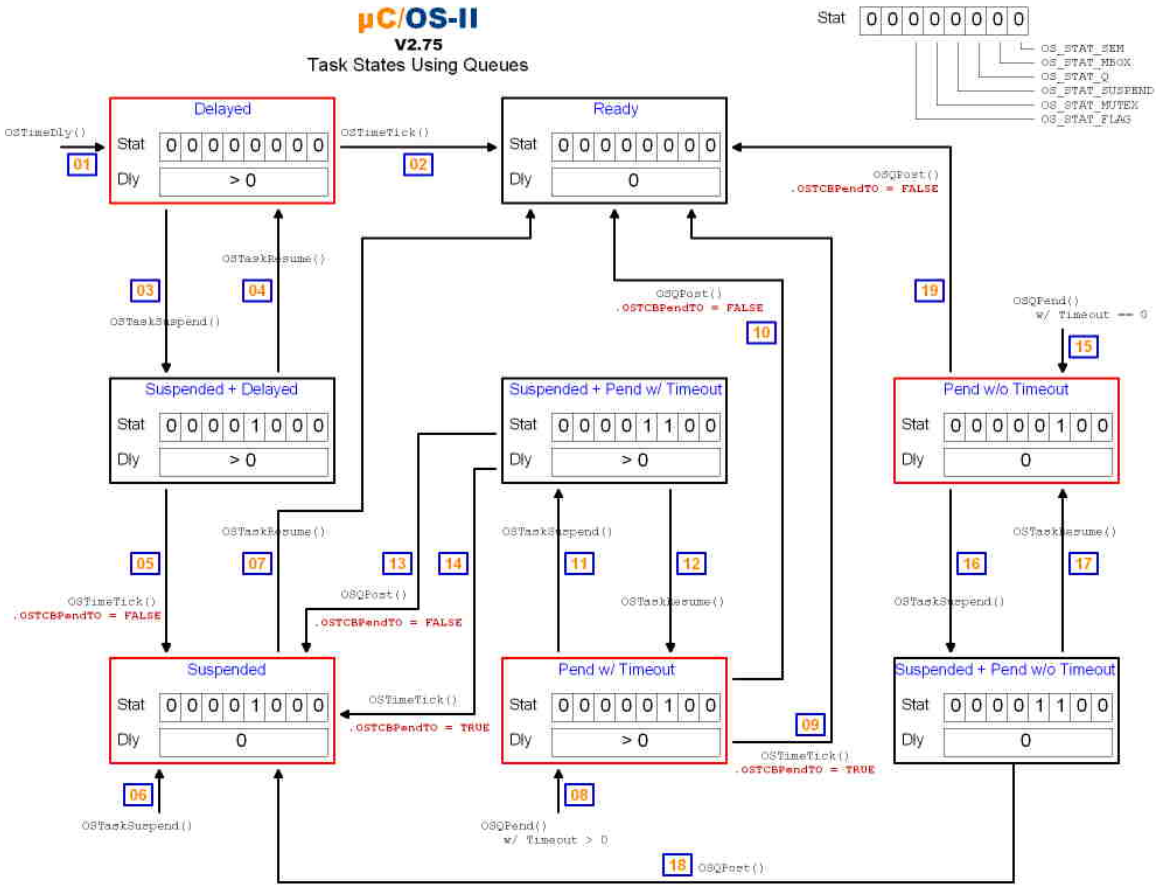
V2.75-009

Changed `#if OS_EVENT_EN > 0` with `#if OS_EVENT_EN` because LINT was complaining that the boolean value `OS_EVENT_EN` was being compared with an integer value.

Correction of Bug V2.70-001

Problem description:
If a task pends on an event with a timeout but `.OSTCBDly` gets decremented to `0` before the task gets suspended (using `OSTaskSuspend()`) by another task then, when the suspension is removed, the task 'appears' to be waiting forever on the message queue (when it was waiting with a timeout). Of course, if the queue is posted, the task would be made ready to run by the post.

Problem correction:
The problem was corrected by adding a variable called `.OSTCBPendTO` in the `OS_TCB`. This variable is set by `OSTimeTick()` when `OSTimeTick()` determines that the delayed task is in fact pending on either a semaphore, mailbox, queue, mutex or event flag. To find and correct the problem, we drew a state transition diagram of the different states a task can take as shown in the figure below.



Each large box represents a state a task can be in. A 'red' state can be entered directly by a task or from another task. The `Stat` byte contains the value of the `.OSTCBStat` field in the `OS_TCB` of the task. `Dly` represents the value of `.OSTCBDly` and can be either `0` or a non-zero value (i.e. `> 0`). We assumed message queues in this example but the

states apply to semaphores, mailboxes, mutexes and event flags. Below is a narration of the different states.

01 A running task calls `OSTimeDly()`. `.OSTCBStat` doesn't get changed and only `.OSTCBDly` is affected.

02 `OSTimeTick()` decrements `.OSTCBDly` to zero and the task is made ready-to-run.

03 A delayed task gets suspended by another task.

04 The task suspension is removed by another task.

05 `OSTimeTick()` decrements `.OSTCBDly` to zero but, since the task is still suspended, it doesn't get readied. Also, the flag `.OSTCBPendTO` gets set to `FALSE` since the task was not pending on anything.

06 A task gets suspended by itself or by another task. Of course, this task is removed from the ready list but is not waiting for any event.

07 The suspended task is readied by another task that calls `OSTaskResume()`.

08 A task calls `OSQPend()` and specifies a non-zero timeout value. This means that the task will be readied if a message is received within the timeout period or, if the timeout expires.

09 `OSTimeTick()` is called before a message is received. In this case, the flag `.OSTCBPendTO` is set and the `OS_STAT_Q` flag is cleared by `OSTimeTick()`. In previous versions, we didn't clear the `OS_STAT_Q` flag because we used it to indicate that the task timed out waiting for the event to occur. Since we now have the `.OSTCBPendTO` flag, we will use it for this purpose.

10 A task calls `OSQPost()` sending a message to the task via a message queue. In this case, the timeout is cancelled and the flag `.OSTCBPendTO` is set to `FALSE`.

11 A task calls `OSTaskSuspend()` to suspend a task that was already waiting on a message queue (with timeout).

12 A task calls `OSTaskResume()` to resume the task suspended. In this case, the task is still not ready-to-run because the message queue has not been posted and the timeout has not expired.

13 A task calls `OSQPost()` before the timeout expires. However, the task is still suspended. Note that the `OSQPost()` cancels the timeout (sets `.OSTCBDly` to `0`) and sets the flag `.OSTCBPendTO` to `FALSE` because we didn't get a timeout. Note also that the message is given to the task because it was the highest priority task waiting for the message, even though the task is still suspended.

14 `OSTimeTick()` occurs before the message gets posted to the queue. In this case, `OSTimeTick()` sets the `.OSTCBPendTO` flag to `TRUE` indicating that the message was not received within the specified timeout period. However, the task is still unconditionally suspended. If the message is posted before the task is resumed, the `.OSTCBPendTO` flag will be cleared.

15 A task calls `OSQPend()` and specifies a zero timeout value indicating that the task will wait forever to receive a message.

16 A task calls `OSTaskSuspend()` to suspend a task that was already waiting on a message queue (without timeout).

17 A task calls `OSTaskResume()` to resume the task suspended. In this case, the task is still not ready-to-run because it's waiting for an event that did not occur.

18 A task calls `OSQPost()`. However, the task is still suspended. The flag `.OSTCBPendTO` is set to `FALSE` because we didn't get a timeout. Note also that the message is given to the task because it was the highest priority task waiting for the message, even though the task is still suspended.

19 A task calls `OSQPost()`. The flag `.OSTCBPendTO` is set to `FALSE` because we didn't get a timeout.

V2.70

(2003/04/01)

V2.70 is a considered a major release for a number of reasons:

1) The directory structure for ports has been completely revised. This doesn't really affect the source code for µC/OS-II per-se but it does imply that port files have been moved around.
2) Include files are now surrounded by brackets instead of double quotes. This allows you to locate µC/OS-II and the port files anywhere on your computer system, and let your compile environment resolve include paths. In other words, you now need to tell your compiler which path to search for include files since µC/OS-II file no longer assume an absolute path.
3) All calls to standard library functions have been removed from µC/OS-II and have been replaced with internal `OS_???()` functions. This was done to simplify third party certification.
4) Item #3 above has an additional advantage - compilation of µC/OS-II now only depends on the following three files: `os_cpu.h`, `os_cfg.h` and `ucos_ii.h`. In other words, if you define the contents of `os_cpu.h` and `os_cfg.h` for your product, you will be able to compile µC/OS-II files standalone.
5) Port files for the 80x86 CPU running in a DOS environment are no longer included with the distribution. This has been done for two reasons. First, we don't want µC/OS-II to be thought of as 'only' an 80x86 RTOS since it's been ported to a large number of processors. Second, all the other processor ports are available on the web site as a free download and now, the 80x86 ports are no different.
6) The DOS utility `TO.EXE` is no longer part of the distribution.
7) We now include two new files: `os_cfg_r.h` and `os_dbg_r.c`. These are described later.
8) Initialization of the statistic task now takes 1/10 of a second instead of 1 second. This has been done to reduce the boot time of an embedded system target.
9) Changed the returned value from `OSFlagAccept()` and `OSFlagPend()` to now return the value of the flags that caused the task to become ready-to-run. This was done because a lot of users requested this 'preferred' behavior.

This release corrects just one minor issue that was reported by a user of V2.62. This release also contains some minor changes. No new features or functions were added.

Important

`ucos_ii.h` now includes `#include` statements to include `os_cpu.h` and `os_cfg.h`. This means that you **MUST** now **REMOVE** these `#include` statements from `includes.h`. In other words, `ucos_ii.h` now has:

```
#include  <os_cpu.h>
#include  <os_cfg.h>
```

and those statements **MUST** be **REMOVED** from `includes.h` otherwise the above two files would be multiply included

FIXED MINOR ISSUE WITH V2.62

Bug V2.62-001:

In `OSTaskDel()`, we had added a statement to clear the stack pointer of the task being deleted. This statement appears on line 428 and has been since removed. The code was added originally to show that the stack of a task that has been deleted is no longer valid. This was to support Kernel Awareness but was found to cause side effects. The line to delete is:

```
ptcb->OSTCBStkPtr = (OS_STK *)0;  /* Show that TCB is 'unused' */
```

CHANGES IN V2.70

V2.70-001

In V2.70, we changed the directory structure of where ports are placed. This change was necessary because of the growing confusion about where ports should be placed. The new directory structure is explained in AN-2002 which can be downloaded from the Micriμm web site.

V2.70-002

The file `OS_DEBUG.C` has been renamed to `OS_DBG.C`. `OS_DEBUG.C` was introduced in V2.62.

V2.70-003

Conditional compilation of object names is now checking for greater than 1 (i.e. > 1) instead of greater than zero (i.e. > 0). The reason is because of the following code example:

```
#if OS_EVENT_NAME_SIZE > 1
    pevent->OSEventName[0] = '?';
    pevent->OSEventName[1] = OS_ASCII_NUL;
#endif
```

If `OS_EVENT_NAME_SIZE` was set to `1` then there would not be sufficient room in the `.OSEventName` file to hold the '?' as well as the `NUL` character. This was really not a big problem in the past because it would be unlikely that you would have allocated only ONE character to the name of an object.

V2.70-004

Removed all calls to standard library functions and replaced them with local functions which are found in `OS_CORE.C` as follows:

Standard Library Function:	Has been replaced by:
`memcpy()`	`OS_MemCopy()`
`memset()`	`OS_MemClr()`
`strlen()`	`OS_StrLen()`
`strcpy()`	`OS_StrCopy()`

V2.70-005

Added call to function `OSDebugInit()` in `OSInit()`. `OSDebugInit()` is a function that has been added in V2.70 because some compilers will actually optimize out all the '`const`' variables in `os_dbg.c` if they are not referenced by any code. The '`const`' in `os_dbg.c` are used by a Kernel Aware debugger and all of the '`const`' are needed. `OSDebugInit()` is a function that really doesn't do anything except reference the '`const`' variables in `os_dbg.c` to prevent the compiler from optimizing them out. Of course, if `OS_DEBUG_EN` is set to `0` in `os_cfg.h` then `OSDebugInit()` is not called and is thus not needed.

V2.70-006

Changed the name of all variables called '`pdata`' and '`data`' to more appropriate variable names. The reason for this change is that some 8051 compilers reserve the words `pdata` and `data` for storage classes.

V2.70-007

`ucos_ii.h` now includes `#include` statements to include `os_cpu.h` and `os_cfg.h` and thus, you **MUST** now **REMOVE** these include statements from the project's master include file, `includes.h` to prevent double inclusion of `os_cpu.h` and `os_cfg.h` in your project. Because of this change, all of the µC/OS-II source files now include `ucos_ii.h` instead of `includes.h`.

V2.70-008

Include files are now surrounded by brackets instead of double quotes. This allows you to locate µC/OS-II and the port files anywhere on your computer system, and let your compile environment resolve include paths. In other words, you now need to tell your compiler which path to search for include files since µC/OS-II file no longer assume an absolute path.

V2.70-009

Changed `OSStatInit()` and `OS_TaskStat()` so that the statistic task only needs 1/10 of a second to determine the CPU capacity. This change was done to speed up the boot time of an embedded system.

V2.70-010

Changed the returned value from `OSFlagAccept()` and `OSFlagPend()` to now return the value of the flags that caused the task to become ready-to-run. This was done because a lot of users requested this 'preferred' behavior.

V2.62

(2003/01/15)

V2.62 is a simple maintenance release. The release corrects a few very minor issues that were reported by users and also, contains some changes to better support Kernel Aware debuggers. No new features or functions were added.

FIXED MINOR ISSUES WITH V2.61

Bug V2.61-001:

In `OS_FLAG.C`, the second `OS_ENTER_CRITICAL()` in `OSFlagPendGetFlagsRdy()` needed to be changed to `OS_EXIT_CRITICAL()`. This problem has been corrected. The BAD code was:

```
OS_FLAGS OSFlagPendGetFlagsRdy (void)
{
    #if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
    #endif
    OS_FLAGS flags;


    OS_ENTER_CRITICAL();
    flags = OSTCBCur->OSTCBFlagsRdy;
    OS_ENTER_CRITICAL();
    return (flags);
}
```

and should have been:

```
OS_FLAGS OSFlagPendGetFlagsRdy (void)
{
    #if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
    #endif
    OS_FLAGS flags;

    OS_ENTER_CRITICAL();
    flags = OSTCBCur->OSTCBFlagsRdy;
    OS_EXIT_CRITICAL();
    return (flags);
}
```

Bug V2.61-002:

In OS_MEM.C, the following code on line #242 was:

```
if (len > (OS_EVENT_NAME_SIZE - 1)) {
```

and should have been:

```
if (len > (OS_MEM_NAME_SIZE - 1)) {
```

Bug V2.61-003:

In OS_CORE.C, OS_TaskStatStkChk() didn't check for a task that was assigned to a MUTEX and thus attempted to compute the stack size of an invalid task. The correct code for this function is:

```
#if (OS_TASK_STAT_STK_CHK_EN > 0) && (OS_TASK_CREATE_EXT_EN > 0)
void  OS_TaskStatStkChk (void)
{
    OS_TCB      *ptcb;
    OS_STK_DATA  stk_data;
    INT8U        err;
    INT8U        prio;


    for (prio = 0; prio <= OS_IDLE_PRIO; prio++) {
        err = OSTaskStkChk(prio, &stk_data);
        if (err == OS_NO_ERR) {
            ptcb = OSTCBPrioTbl[prio];
            if (ptcb != (OS_TCB *)0) {                          /* Make sure task 'ptcb' is ...   */
                if (ptcb != (OS_TCB *)1) {                      /* ... still valid.               */
#if OS_TASK_PROFILE_EN > 0
                    #if OS_STK_GROWTH == 1
                    ptcb->OSTCBStkBase = ptcb->OSTCBStkBottom + ptcb->OSTCBStkSize;
                    #else
                    ptcb->OSTCBStkBase = ptcb->OSTCBStkBottom - ptcb->OSTCBStkSize;
                    #endif
                    ptcb->OSTCBStkUsed = (INT32U)stk_data.OSUsed;    /* Store the number of bytes used */
#endif
                }
            }
        }
    }
}
#endif
```

Bug V2.61-004:

In OS_CORE.C, OS_TaskStatStkChk() had an error computing the stack base. The code presented in V2.61-003 above corrects the issue.

CHANGES IN V2.62

V2.62-001

In V2.62, we removed the 'debug' code from `OS_CORE.C` and created a NEW file called `OS_DEBUG.C`. On other words, in V2.61, there were a number of '`const`' variables that were added to better support kernel aware debuggers. These `const`s have been moved to the new file `OS_DEBUG.C` for two reasons:

1) If you don't have a kernel aware debugger, `OS_CORE.C` would have added about 100 bytes of code that would serve no purpose. By moving the `const` to `OS_DEBUG.C`, if you don't have a kernel aware debugger, you don't need to compile and link `OS_DEBUG.C` with your µC/OS-II based application.

2) Some compilers (such as the IAR) compiles-out code or constants that don't appear to be used anywhere. In the case of the debug variables, the variables reside in ROM and are only there for the debugger. In other words, they serve no other purpose for µC/OS-II based applications and some compilers would be 'smart' enough to not include them. To prevent this from happening and thus make the variable available for the debugger, the debug variable were placed in `OS_DEBUG.C` so that you can use compiler specific directives to prevent this type of optimization. However, these directives are very compiler specific and could thus change from one compiler to another. This would cause compatibility problems if these directives were placed in `OS_CORE.C` because `OS_CORE.C` is supposed to be compiler and processor independent. Having a separate file (`OS_DEBUG.C`) solves this problem because the file can be associated with the PORT and not the processor independent code.

NOTE

This all means that a PORT should now contain `OS_DEBUG.C` if you use a Kernel Aware Debugger that requires the 'const' provided in `OS_DEBUG.C`. In fact, you might have to modify `OS_DEBUG.C` based on the compiler you are using.

V2.62-002

In `OS_MUTEX.C`, there were a couple of places where some of the MISRA C rules had not been followed (in `OSMutexPend()` and `OSMutexPost()`).

V2.62-003

Added 'tags' to structures.

V2.62-004

Added `OSEndianessTest` const in `OS_DEBUG.C` to allow the debugger to automatically determine whether the processor is little endian or big endian.

V2.61

(2002/10/20)

V2.61 is a simple maintenance release and no run-time bugs were found in µC/OS-II V2.60. The release only corrects a few very minor issues that mostly affected kernel awareness support, and adds a bit of internal code. No new features or functions were added.

FIXED MINOR ISSUES WITH V2.60

Bug V2.60-001:

In OS_CORE.C, the following ROM constant was set to:

```
INT16U  const  OSTaskStatStkChkEn = OS_TASK_STAT_EN;
```

And should have been:

```
INT16U  const  OSTaskStatStkChkEn = OS_TASK_STAT_STK_CHK_EN;
```

Bug V2.60-002:

In uCOS_II.H, the following code was:

```
#if OS_EVENT_NAME_SIZE > 0
INT8U         OSEventNameGet(OS_EVENT *pevent, char *pname, INT8U *err);
void          OSEventNameSet(OS_EVENT *pevent, char *pname, INT8U *err);
#endif
```

and should have been:

```
#if (OS_EVENT_EN > 0) && (OS_EVENT_NAME_SIZE > 0)
INT8U         OSEventNameGet(OS_EVENT *pevent, char *pname, INT8U *err);
void          OSEventNameSet(OS_EVENT *pevent, char *pname, INT8U *err);
#endif
```

Bug V2.60-003:

In uCOS_II.H, the following code was:

```
#if OS_FLAG_NAME_SIZE > 0
INT8U         OSFlagNameGet(OS_FLAG_GRP *pgrp, char *pname, INT8U *err);
void          OSFlagNameSet(OS_FLAG_GRP *pgrp, char *pname, INT8U *err);
#endif
```

and should have been:

```
#if (OS_FLAG_EN > 0) && (OS_FLAG_NAME_SIZE > 0)
INT8U         OSFlagNameGet(OS_FLAG_GRP *pgrp, char *pname, INT8U *err);
void          OSFlagNameSet(OS_FLAG_GRP *pgrp, char *pname, INT8U *err);
#endif
```

Bug V2.60-004:

In OS_MEM.C, the following code was missing in OS_MemInit(), for the last OS_MEM element:

```
#if OS_MEM_NAME_SIZE > 0
    (void)strcpy(pmem->OSMemName, "?");
#endif
```

Bug V2.60-005:

In OS_CORE.C, added conditional compilation for the following prototype:

```
#if OS_TASK_STAT_EN > 0
static  void  OS_InitTaskStat(void);
#endif
```

ADDED CODE

Added V2.61-001:

In OS_CORE.C, added the following ROM constant for kernel awareness support:

```
OSMemSize = sizeof(OS_MEM);
```

V2.60

(2002/09/28)

Changes were made to V2.52 for the following reasons:

a) To fix minor issues with V2.52.
b) To simplify FAA Level A certification by removing all MCDC (Modified Condition Decision Coverage).
c) To follow most of the guidelines of *The Motor Industry Software Reliability Association* "Guidelines for the use of the C language in vehicle based software".
d) To add support for kernel awareness.
e) To directly support µC/OS-View.
f) Added new features.
g) Made some changes to the code.

FIXED ISSUES WITH V2.52

Bug V2.52-001:

In `OS_CORE.C`, function `OS_InitMisc()`, there is no need to test `OS_TASK_CREATE_EXT_EN`:

```
#if (OS_TASK_STAT_EN > 0) && (OS_TASK_CREATE_EXT_EN > 0)
    OSIdleCtrRun  = 0L;
    OSIdleCtrMax  = 0L;
    OSStatRdy     = FALSE;
#endif
```

The correct code is thus:

```
#if OS_TASK_STAT_EN > 0
    OSIdleCtrRun  = 0L;
    OSIdleCtrMax  = 0L;
    OSStatRdy     = FALSE;
#endif
```

Bug V2.52-002:

In `OS_TASK.C`, function `OSTaskDel()`, the variable `self` was never used. The variable is now removed.

Bug V2.52-003:

In `OS_TASK.C`, function `OSTaskStkChk()` was missing a test. The incorrect code was:

```
ptcb = OSTCBPrioTbl[prio];
if (ptcb == (OS_TCB *)0) {
    OS_EXIT_CRITICAL();
    return (OS_TASK_NOT_EXIST);
}
```

The correct code is:

```
ptcb = OSTCBPrioTbl[prio];
if (ptcb == (OS_TCB *)0 || ptcb == (OS_TCB *)1) {
    OS_EXIT_CRITICAL();
    return (OS_TASK_NOT_EXIST);
}
```

Bug V2.52-004:

In OS_MUTEX.C, function OSMutexPost() the condition to check to see if the current task is the owner of the mutex has been changed from:

```
if (OSTCBCur->OSTCBPrio != pip &&
    OSTCBCur->OSTCBPrio != prio) {
    OS_EXIT_CRITICAL();
    return (OS_ERR_NOT_MUTEX_OWNER);
}
```

To:

```
if (OSTCBCur != (OS_TCB *)pevent->OSEventPtr) {
    OS_EXIT_CRITICAL();
    return (OS_ERR_NOT_MUTEX_OWNER);
}
```

This change allows a task to obtain multiple mutexes. A task could thus have the following code:

```
Acquire Mutex #1;
Acquire Mutex #2;
Release Mutex #2;
Release Mutex #1;
```

Mutexes MUST be released in the same order as they were acquired.

Bug V2.52-005:

In OS_MUTEX.C, the function OSMutexPend() was changed to allow a mutex owner to pend on another kernel object such as a semaphore. In other words, a task could have the following code:

```
Acquire Mutex;       /* Mutex is available, task now owns it      */
Acquire Semaphore;   /* Semaphore is NOT available, suspend task! */
.
.
```

Then, a high priority task that would call OSMutexPend() on the same mutex would notice that the mutex owner has a lower priority than the task that needs the mutex. OSMutexPend() would then raise the priority of the task that owns the mutex and will notice that the task is also waiting on a semaphore. OSMutexPend() would then change the priority of the mutex owner in the semaphore wait list.

SIMPLIFYING 'FAA LEVEL A' CERTIFICATION

Changes were made to V2.52 to remove all MCDC (Modified Condition Decision Coverage). MCDCs are basically conditionals with multiple possible outcomes. For example, in the following code, there are eight (8) possible outcomes based on the different values of `a`, `b`, `c`, `d`, `e` and `f`:

```
if (a == b && c == d && e == f) {
    /* Conditions met */
}
```

A better way to write the above code (from a certification perspective) is shown below:

```
if (a == b) {
    if (c == d) {
        if (e == f) {
            /* Conditions met */
        }
    }
}
```

I went through all the µC/OS-II code and removed the MCDCs. Of course, the code behaves exactly the same as before.

FOLLOWED MOST OF THE MISRA GUIDELINES

MISRA stands for "The Motor Industry Software Reliability Association" and this association published back in April 1998, a list of 127 guidelines for programming applications using the C programming language. You can obtain this document by visiting:

> http://www.misra.org.uk

The document is called:

> "Guidelines For The Use Of The C Language In Vehicle Based Software"
> ISBN 0 9524156 9 0

It so happens that µC/OS-II was written by following most of the MISRA guidelines even before the guidelines were ever published. At this time, µC/OS-II is not 'compliant' with the guidelines but simply follows most of them.

One of the most significant changes to µC/OS-II's code is the removal of assignments inside conditionals. For instance, the following code:

```
if ((pevent->OSEventTbl[y] &= ~bitx) == 0) {
    /* … */
}
```

Has been replaced by:

```
pevent->OSEventTbl[y] &= ~bitx;
if (pevent->OSEventTbl[y] == 0) {
    /* … */
}
```

SUPPORT OF KERNEL AWARE DEBUGGERS

Variables and constants have been added to help support kernel aware debuggers. Specifically, a number of variables can be queried by a debugger to find out about compiled-in options. For example, the debugger can find out the size of an `OS_TCB`, µC/OS-II's version number, the size of an event flag group (`OS_FLAG_GRP`) and much more. Those variables are enabled by `OS_DEBUG_EN` in `OS_CFG.H`.

SUPPORT OF µC/OS-View

Variables in `OS_TCB` have been added (see `OS_TASK_PROFILE_EN`) to support profiling tools such as µC/OS-View.

Also `OS_TaskStat()` can now check the stack of each of the active tasks (see `OS_TASK_STAT_STK_CHK_EN`).

An `OS_TCB` can also contain the name of each task which can then be displayed on the µC/OS-View Windows application.

µC/OS-View can 'step' tick interrupts one at a time. In other words, through a command sent by a user of µC/OS-View, µC/OS-II can process one tick at a time. Each tick requires a user to press a key from the µC/OS-View application.

ADDED NEW FEATURES

1) Find out which flag(s) caused task to wakeup.

Added the function `OSFlagPendGetFlagsRdy()` (file `OS_FLAG.C`) to allow to determine which flag(s) caused the current task to become ready. In other words, you will now be able to know what event flag(s) caused the pending task to wake up.

2) Posting NULL pointers to queues.

It is now possible to send `NULL` pointer message through queues. `OSQPost()` and `OSQPostFront()` no longer blocks `NULL` pointers from being deposited into queues. This means that `OSQPend()` will thus be able to receive `NULL` pointer messages. You should now check the status of the `err` argument to determine whether the return from the pend was caused by a timeout or the actual reception of a message.

Because of this change, I had to change the API for `OSQAccept()` so that it returns an error code indicating the outcome of the call.

3) Assigning names to Tasks and other kernel objects.

It is now possible to assign names to Tasks, Memory Partitions, Semaphores, Mutexes, Event Flags, Mailboxes and Queues. The names are useful when debugging applications. You assign names by calling one of the following functions:

```
OSEventNameSet()
OSFlagNameSet()
OSMemNameSet()
OSTaskNameSet()
```

You can obtain the name of a task or a kernel object by calling the following functions:

```
OSEventNameGet()
OSFlagNameGet()
OSMemNameGet()
OSTaskNameGet()
```

This version doesn't allow you to manipulate kernel objects using names. For example, you can't delete a task by specifying its name, you can't post to a queue by specifying the queue by its name, etc.

4) Disable calls to OSTaskSwHook() and OSTimeTickHook()

It is now possible to disable (at compile time) the need to have the functions `OSTaskSwHook()` and `OSTimeTickHook()`. This feature was requested because of the overhead involved in calling empty functions during a context switch and also every tick.

To disable `OSTaskSwHook()`, simply set `OS_TASK_SW_HOOK_EN` to `0` in `OS_CFG.H`. Of course, the port (`OS_CPU_A.ASM`) for the processor you are using must not call `OSTaskSwHook()`.

To disable `OSTimeTickHook()`, simply set `OS_TIME_TICK_HOOK_EN` to `0` in `OS_CFG.H`.

5) Added variables in OS_TCB to allow profiling

Variables have been added to `OS_TCB` to allow each task to be profiled. In other words, µC/OS-II contains variables that register the number of time a task is 'switched-in', how long a task takes to execute, how much stack space each task consumes and more. These variables have been added to better support µC/OS-View and other profiling tools.

6) Added tick stepping support for µC/OS-View

µC/OS-View can now 'halt' µC/OS-II's tick processing and allow you to issue 'step' commands from µC/OS-View. In other words, µC/OS-View can prevent µC/OS-II from calling `OSTimeTick()` so that timeouts and time delays are no longer processed. However, though a keystroke from µC/OS-View, you can execute a single tick at a time. If enabled, `OSTimeTickHook()` is still executed at the regular tick rate in case you have time critical items to take care of in your application.

7) Added new #defines in OS_CFG.H

Instead of edition your `OS_CFG.H`, I recommend that you copy one of the `OS_CFG.H` files provided with the V2.60 release and then modify it to satisfy your application requirements.

`OS_DEBUG_EN`

This `#define` adds ROM constants to help support kernel aware debuggers. Specifically, a number of named ROM variables can be queried by a debugger to find out about compiled-in options. For example, the debugger can find out the size of an `OS_TCB`, µC/OS-II's version number, the size of an event flag group (`OS_FLAG_GRP`) and much more.

`OS_EVENT_NAME_SIZE`

This `#define` determines the size of ASCII strings used to name either semaphores, mutexes, mailboxes and queues. If set to `0`, this feature will be disabled: no RAM will be allocated and the functions `OSEventNameGet()` and `OSEventNameSet()` will not be compiled. If set to a non-zero value, it determines the number of bytes allocated for the name. Please note that you need to accommodate for the `NUL` character and if you do use a non-zero value, you should have a minimum of `2` for this value.

`OS_FLAG_NAME_SIZE`

This `#define` determines the size of ASCII strings used to name event flag groups. If set to `0`, this feature will be disabled: no RAM will be allocated and the functions `OSFlagNameGet()` and `OSFlagNameSet()` will not be compiled. If set to a non-zero value, it determines the number of bytes allocated for the name. Please note that you need to accommodate for the `NUL` character and if you do use a non-zero value, you should have a minimum of `2` for this value.

`OS_MEM_NAME_SIZE`

This `#define` determines the size of ASCII strings used to name memory partitions. If set to `0`, this feature will be disabled: no RAM will be allocated and the functions `OSMemNameGet()` and `OSMemNameSet()` will not be compiled. If set to a non-zero value, it determines the number of bytes allocated for the name. Please note that you need to accommodate for the `NUL` character and if you do use a non-zero value, you should have a minimum of `2` for this value.

`OS_TASK_NAME_SIZE`

This `#define` determines the size of ASCII strings used to name tasks. If set to `0`, this feature will be disabled: no RAM will be allocated and the functions `OSTaskNameGet()` and `OSTaskNameSet()` will not be compiled. If set to a non-zero value, it determines the number of bytes allocated for the name. Please note that you need to accommodate for the `NUL` character and if you do use a non-zero value, you should have a minimum of `2` for this value.

OS_TASK_PROFILE_EN

This #define is used to allocate storage for variables used for run-time task profiling. These variables are used by µC/OS-View and some kernel aware debuggers.

OS_TASK_STAT_STK_CHK_EN

This #define allows the statistic task to do run-time checking of all the stacks of all the active tasks. In other words, when set to 1, OS_TaskStat() calls the function OS_TaskStatStkChk(). Of course, for this to happen, OS_TASK_STAT_EN must also be set to 1.

OS_TASK_SW_HOOK_EN

It is now possible to disable (at compile time) the need to have the functions OSTaskSwHook(). This feature was requested because of the overhead involved in calling empty functions during a context switch and also every tick. To disable OSTaskSwHook(), simply set OS_TASK_SW_HOOK_EN to 0 in OS_CFG.H. Of course, the port (OS_CPU_A.ASM) for the processor you are using must not call OSTaskSwHook().

OS_TICK_STEP_EN

µC/OS-View can now 'halt' µC/OS-II's tick processing and allow you to issue 'step' commands from µC/OS-View. In other words, µC/OS-View can prevent µC/OS-II from calling OSTimeTick() so that timeouts and time delays are no longer processed. However, though a keystroke from µC/OS-View, you can execute a single tick at a time. If OS_TIME_TICK_HOOK_EN (see below) is set to 1, OSTimeTickHook() is still executed at the regular tick rate in case you have time critical items to take care of in your application.

OS_TIME_TICK_HOOK_EN

It is now possible to disable (at compile time) the need to have the functions OSTimeTickHook(). This feature was requested because of the overhead involved in calling empty functions during a context switch and also every tick. To disable OSTimeTickHook(), simply set OS_TIME_TICK_HOOK_EN to 0 in OS_CFG.H.

CHANGES

1) Added 'extern C' in uCOS_II.H

An "extern C" statement has been added to allow you to compile µC/OS-II using a C++ compiler.

2) Renamed ALL files to lower case

All the µC/OS-II files have been renamed to lower case to make it easier to compile under UNIX environments.

3) Changed the structure of OSTaskChangePrio()

I changed the structure of the code for OSTaskChangePrio() to reduce the indentation, simplify and make the code cleaner. I also removed the re-enabling of

interrupts when computing `x`, `y`, `bitx` and `bity`. There is thus, there is no need to 'reserve' the `OSTCBPrioTbl[]` entry by setting it to `(OS_TCB *)1`.

4) Assigning a NULL pointer to OSTCBStkPtr

I now assign a `NULL` pointer to `OSTCBStkPtr` when the free list of TCBs is created and when a task is deleted.

5) Posting NULL pointers to queues.

Because it is now possible to post `NULL` pointers to queues, I had to change the API for `OSQAccept()` so that it returns an error code indicating the outcome of the call.

6) Removed assignments inside `if ()` statements.

Code like shown below:

```
if ((pevent->OSEventTbl[y] &= ~bitx) == 0) {
    /* … */
}
```

Has been replaced by:

```
pevent->OSEventTbl[y] &= ~bitx;
if (pevent->OSEventTbl[y] == 0) {
    /* … */
}
```

7) Removed MCDCs.

Code like shown below:

```
if (a == b && c == d && e == f) {
    /* Conditions met */
}
```

Has been replaced by the following code:

```
if (a == b) {
    if (c == d) {
        if (e == f) {
            /* Conditions met */
        }
    }
}
```

8) Added memset() to clear RAM

Added calls to `memset()` to clear (i.e. initialize) the `OSTCBPrioTbl[]`, `OSTCBTbl[]`, `OSMemTbl[]`, `OSFlagTbl[]` and `OSEventTbl[]`. The reason `memset()` was used was for speed and to reduce code size. These tables are cleared during initialization to prevent a kernel aware debugger to display un-initialized values.

In most cases, the initialization code for the different kernel objects has also been reduced.

V2.52

(2002/01/26)

This release is for the new edition of the book: *MicroC/OS-II, The Real-Time Kernel, 2nd Edition.*

V2.52 fixes minor bugs reported in V2.51.

Bug V2.51-003:

In `uCOS_II.H`, the following code was corrected as follows:

```
#ifndef OS_FLAG_QUERY_EN
 #error "OS_CFG.H, Missing OS_FLAG_DEL_EN: Include code for OSFlagQuery()"
```

needs to be:

```
#ifndef OS_FLAG_QUERY_EN
#error "OS_CFG.H, Missing OS_FLAG_QUERY_EN: Include code for OSFlagQuery()"
```

Bug V2.51-002:

In `OS_Q.C`, the following code was corrected as follows:

The function `OSQQuery()` contains a BUG in the following code which is towards the end of the function.

```
pq = (OS_Q *)pevent->OSEventPtr;
if (pq->OSQEntries > 0) {
    pdata->OSMsg = pq->OSQOut;  /* Get next message to return if available */
} else {
    pdata->OSMsg = (void *)0;
}
```

The **CORRECT** code is shown below. Note that **`pq->OSQOut`** was missing the **`*`**.

```
pq = (OS_Q *)pevent->OSEventPtr;
if (pq->OSQEntries > 0) {
    pdata->OSMsg = *pq->OSQOut; /* Get next message to return if available */
} else {
    pdata->OSMsg = (void *)0;
}
```

Bug V2.51-001:

In OS_CPU_A.ASM, the following code was corrected as follows:

The *NEW* ISRs MUST check to see `if OSIntNesting == 1` *BEFORE* you save the SP in the current task's OS_TCB. The incorrect 'pseudo' code is:

```
OSTCBCur->OSTCBStkPtr = SP          /* Save SP onto current task's stack */
```

and should be:

```
if (OSIntNesting == 1) {
  OSTCBCur->OSTCBStkPtr = SP       /* Save SP onto current task's stack */
}
```

The reason we need this change is that we don't want to save the current value of SP if the ISR is for a nested ISR!

V2.52 adds a few minor changes to V2.51.

OS_CORE.C:

I decided to split OSInit() into calls to multiple functions to make the code cleaner. The new functions should be self-explanatory:

```
static  void  OS_InitEventList(void);
static  void  OS_InitMisc(void);
static  void  OS_InitRdyList(void);
static  void  OS_InitTaskIdle(void);
static  void  OS_InitTaskStat(void);
static  void  OS_InitTCBList(void);
```

In OSIntEnter(), I removed the OS_ENTER_CRITICAL() and OS_EXIT_CRITICAL() macros because it is assumed that OSIntEnter() will be called with interrupts disabled. Also, I added a check to make sure OSRunning is set to TRUE.

In OSIntExit(), I added a check to make sure OSRunning is set to TRUE.

In OSTimeTick(), I added a check to make sure OSRunning is set to TRUE before going through the OS_TCBs.

In OS_TaskStat(), I changed the equation to prevent overflowing the calculation on very fast CPUs. The equation was written as:

```
CPU Usage (%) = 100 – 100 * OSIdleCtr / OSIdleCtrMax;
```

Because the compiler would first perform the `100 * OSIdleCtr` operation, an `OSIdleCtr` greater than `42,949,763` would overflow the calculation and thus report an incorrect result. The equation is now written as:

```
CPU Usage (%) = 100 – OSIdleCtr * (OSIdleCtrMax / 100);
```

This allows `OSIdleCtr` to reach `4,294,967,295` (i.e. 2^{32}-1) before the equation fails. I don't expect this to happen for a while since `OSIdleCtr` is incremented in a loop. The loop contains instructions that would consume a few CPU cycles each iteration.

OS_MBOX.C:

In `OSMboxPend()` (`OS_MBOX.C`), I moved the check for `OSIntNesting` at the beginning of the function because you should NEVER call `OSMboxPend()` from an ISR.

OS_Q.C:

In `OSQPend()` (`OS_Q.C`), I moved the check for `OSIntNesting` at the beginning of the function because you should NEVER call `OSQPend()` from an ISR.

OS_SEM.C:

In `OSSemPend()` (`OS_SEM.C`), I moved the check for `OSIntNesting` at the beginning of the function because you should NEVER call `OSSemPend()` from an ISR.

V2.51

(2001/06/09)

Two weeks ago, I released V2.05 and today, I found a bug in it (bug V205-001). I decided to slightly change the numbering system of releases. Complex releases (like V2.04 to V2.05) will now increase by 0.10 and minor (bug fixes or slight improvements) will now be increasing by 0.01. This means that V2.51 is now called V2.50 and with this bug fix, the release is V2.51. The reason this is done is to allow you to call `OSVersion()` and get the proper release number. If I didn't change the numbering system, I would have had to call the release with the bug correction V2.06. I was reserving such releases as major releases.

Bug V2.51-001:

In the NEW port file, an ISR MUST first check to see `if OSIntNesting == 1` before we save the `SP` in the current task `OS_TCB`. This bug only applies to the NEW algorithm for the port files and thus does NOT affect previous ports.

See **New Algorithm For Ports** at the end of the V2.51 notes.

V2.51 is a big upgrade for µC/OS-II for the following reasons:

1) In this release, I added Event Flags (see `OS_FLAG.C`). Event flags are described in AN-1007 which can be downloaded from www.Micrium.com.

2) I received numerous e-mails requesting to reduce the footprint of µC/OS-II to a minimum. To address this issue, I added a number of `#define` constants in `OS_CFG.H` which allow you to take out most of the features in µC/OS-II that you might not be using. Specifically, there are `#define`s to remove the code for `OS???Accept()`, `OS???Query()`, `OS???Post()`, `OSSchedLock()` and `OSSchedUnlock()` and more.

3) This release comes with NEW ports for the Intel 80x86. These ports have been revised to REMOVE the dependency on compilers. Specifically, you no longer need to change the function `OSIntCtxSw()` in order to adjust the value of the Stack Pointer (i.e. the `SP`) register based on compiler options. The modification to accomplish this feature can ALSO be added to most processor ports!

WARNING

If you use the NEW port files in your product you **WILL** need to change ALL your Interrupt Service Routines (ISRs) to handle the new way the port works.

See **New Algorithm For Ports** at the end of the V2.51 notes.

4) All µC/OS-II **internal** functions are now prefixed with `OS_` instead of `OS`. This allows you to immediately determine that these functions should NOT be called by your application. Also, these functions have been moved at the end of their respective file to get them 'out-of-the-way'.

5) `OS_TaskIdle()` now calls `OSTaskIdleHook()` to allow you to do such things as STOP the CPU to conserve power when running the idle task. You will need to add code in `OSTaskIdleHook()` to execute whatever is necessary for your CPU to enter it's power down mode.

6) I added `OSMboxPostOpt()` and `OSQPostOpt()`. The new calls allow you to 'broadcast' a message to **all** tasks waiting on either a message mailbox or a message queue. In addition, `OSQPostOpt()` can replace both `OSQPost()` AND `OSQPostFront()`. This was done to further reduce the amount of code space needed by µC/OS-II. In other words, you can start using `OSQPostOpt()` INSTEAD of `OSQPost()` and `OSQPostFront()` and thus save a significant amount of code space.

7) Added `#error` directives in `uCOS_II.H` to have the compiler complain whenever there are missing `#define`s in your application. This will be useful to ensure that you have not forgotten any of the NEW `#define`s added in V2.51.

8) Previous versions required that you declared a minimum of 2 event control blocks, 2 message queues, and 2 memory partitions. V2.51 now allows you to reduce the RAM footprint by allowing you to declare only ONE of each of the data structures mentioned (and well as only 1 event flag group). In other words, you can now specify in `OS_CFG.H`:

```
#define  OS_MAX_EVENTS    1
#define  OS_MAX_FLAGS     1
#define  OS_MAX_MEM_PART  1
#define  OS_MAX_QS        1
```

9) All conditional compilation is now done as follows:

```
#if condition_name > 0
```

instead of:

```
#if condition_name
```

The condition name is checked for a non-zero value to enable the code. This will allow the compiler to complain in case you forget to define `condition_name`.

10) V2.51 correct the four know bugs that were reported in V2.04.

V2.04-001:

The wrong argument was being passed to the call `OSTaskCreateHook()` in `OSTCBInit()`. The bad code was:

```
OSTaskCreateHook(OSTCBPrioTbl[prio]);
```

It is now:

```
OSTaskCreateHook(ptcb);
```

V2.04-002:

The test in `OSMutexPost()` to see if the posting task owns the MUTEX was incorrect. The correct test needed to have && instead of || as follows:

```
if (OSTCBCur->OSTCBPrio != pip &&
    OSTCBCur->OSTCBPrio != prio) {
    OS_EXIT_CRITICAL();
    return (OS_ERR_NOT_MUTEX_OWNER);
}
```

V2.04-003:

The function `OSMutexDel()` needed to release the priority of the PIP. The following line was added in `OSMutexDel()`:

```
OSTCBPrioTbl[pip] = (OS_TCB *)0;
```

V2.04-004:

The function prototype for `OSMutexDel()` needed to be added in `uCOS_II.H`.

OS_CFG.H:

Added a number of #define in OS_CFG.H to allow you to reduce the amount of code and data space. The reason this is done using #defines instead of simply using a librarian is to prevent having to support a large number of librarians and also to ensure that data space is also reduced when un-needed features (i.e. functions) also require data storage.

OS_MAX_FLAGS is used to determine how many event flags your application will support.

OS_FLAG_EN to Enable (1) or Disable (0) code generation for ALL event flag services and data storage. Also, OS_FLAG_WAIT_CLR_EN allows you to Enable (1) or Disable (0) code generation for code to wait for 'cleared' event flags.

The following table summarizes all the other #define constants ADDED in V2.51. The #defines are set to 1 by default, enabling the code.

#define name in OS_CFG.H	... to enable the function:
OS_FLAG_ACCEPT_EN	OSFlagAccept()
OS_FLAG_DEL_EN	OSFlagDel()
OS_FLAG_QUERY_EN	OSFlagQuery()
OS_MBOX_ACCEPT_EN	OSMboxAccept()
OS_MBOX_POST_EN	OSMboxPost()
OS_MBOX_POST_OPT_EN	OSMboxPostOpt()
OS_MBOX_QUERY_EN	OSMBoxQuery()
OS_MEM_QUERY_EN	OSMemQuery()
OS_MUTEX_ACCEPT_EN	OSMutexAccept()
OS_MUTEX_QUERY_EN	OSMutexQuery()
OS_Q_ACCEPT_EN	OSQAccept()
OS_Q_POST_EN	OSQPost()
OS_Q_POST_FRONT_EN	OSQPostFront()
OS_Q_POST_OPT_EN	OSQPostOpt()
OS_Q_QUERY_EN	OSQQuery()
OS_SEM_ACCEPT_EN	OSSemAccept()
OS_SEM_QUERY_EN	OSSemQuery()
OS_TASK_QUERY_EN	OSTaskQuery()
OS_TIME_DLY_HMSM_EN	OSTimeDlyHMSM()
OS_TIME_DLY_RESUME_EN	OSTimeDlyResume()
OS_TIME_GET_SET_EN	OSTimeGet() and OSTimeSet()
OS_SCHED_LOCK_EN	OSSchedLock()and OSSchedUnlock()

Added the typedef OS_FLAGS to allow you to specify the width of flags in an event flag group.

IMPORTANT

You **WILL** need to add **ALL** of the above `#define` in your `OS_CFG.H` files because `uCOS_II.H` contains error checks that will make your compiler complain if you don't include these `#define`s. The easiest way to accomplish this is to simply copy one of the `OS_CFG.H` files supplied in this release and paste it into your application and enable/disable the features you need.

OS_CORE.C:

Added call to `OS_FlagInit()` in `OSInit()` to support event flags.

Added call to `OSTaskIdleHook()` in `OS_TaskIdle()` to allow you to do such things as STOP the CPU to conserve power when running the idle task. You will need to add code in `OSTaskIdleHook()` to execute whatever is necessary for your CPU to enter it's power down mode.

Added conditional compilation so that when `OS_SCHED_LOCK_EN` is set to `1` in `OS_CFG.H`, the code for `OSSchedLock()` and `OSSchedUnlock()` will be produced.

Corrected a bug in `OS_TCBInit()`. `OSTaskCreateHook()` was being `OSTCBPrioTbl[prio]` passed INSTEAD of `ptcb`. `OSTCBPrioTbl[prio]` didn't contain a valid pointer when `OSTaskCreateHook()` was being called.

WARNING

If you use the NEW port files in your product you will need to change ALL your Interrupt Service Routines (ISRs) to handle the new way the port works.

See **New Algorithm For Ports** at the end of the V2.51 notes.

OS_FLAG.C:

Added event flags to µC/OS-II, see AN-1007.

OS_MBOX.C:

Added conditional compilation so that when `OS_MBOX_ACCEPT_EN` is set to `1` in `OS_CFG.H`, the code for `OSMboxAccept()` will be produced.

Added conditional compilation so that when `OS_MBOX_POST_EN` is set to `1` in `OS_CFG.H`, the code for `OSMboxPost()` will be produced. This allows you to reduce the amount of code space. The reason this conditional compilation has been added is because I added the more powerful function `OSMboxPostOpt()` which can emulate `OSMboxPost()` and also allows you to broadcast messages to all tasks waiting on the mailbox.

Added `OSMboxPostOpt()` which can emulate `OSMboxPost()` and also allows you to broadcast messages to all tasks waiting on the mailbox. The `#define` constant `OS_MBOX_POST_OPT_EN` found in `OS_CFG.H` allows you to enable (when `1`) or disable (when `0`) this feature.

Added conditional compilation so that when `OS_MBOX_QUERY_EN` is set to `1` in `OS_CFG.H`, the code for `OSMboxQuery()` will be produced. This allows you to reduce the amount of code space.

OS_MEM.C:

Added code to test the argument `addr` to make sure it's not a `NULL` pointer in `OSMemCreate()`.

Added code to test the argument `pmem` to make sure it's not a `NULL` pointer in `OSMemGet()`.

Added code to test the argument `pmem` and `pblk` to make sure they are not `NULL` pointers in `OSMemGet()`.

Added conditional compilation so that when `OS_MEM_QUERY_EN` is set to `1` in `OS_CFG.H`, the code for `OSMemQuery()` will be produced. This allows you to reduce the amount of code space.

Added code to test the argument `pmem` and `pdata` to make sure they are not `NULL` pointers in `OSMemQuery()`.

Added conditional compilation to allow you to declare storage for a single memory partition. In other words, you are now allowed to set OS_MAX_MEM_PART to 1 in OS_CFG.H.

OS_MUTEX.C:

Added conditional compilation so that when OS_MUTEX_ACCEPT_EN is set to 1 in OS_CFG.H, the code for OSMutexAccept() will be produced. This allows you to reduce the amount of code space.

Added conditional compilation so that when OS_MUTEX_QUERY_EN is set to 1 in OS_CFG.H, the code for OSMutexQuery() will be produced. This allows you to reduce the amount of code space.

Fixed a bug in OSMutexDel(). The entry in OSTCBPrioTbl[] was not being freed at the priority inheritance priority. This has been corrected.

Fixed a bug in OSMutexPost(). The current task priority was being tested for && instead of ||. This has been corrected.

OS_Q.C:

Added conditional compilation so that when OS_Q_ACCEPT_EN is set to 1 in OS_CFG.H, the code for OSQAccept() will be produced. This allows you to reduce the amount of code space.

Added conditional compilation so that when OS_Q_FLUSH_EN is set to 1 in OS_CFG.H, the code for OSFlushAccept() will be produced. This allows you to reduce the amount of code space.

Added conditional compilation so that when OS_Q_POST_EN is set to 1 in OS_CFG.H, the code for OSQPost() will be produced. This allows you to reduce the amount of code space. The reason this conditional compilation has been added is because I added the more powerful function OSQPostOpt() which can emulate both OSQPost() and OSQPostFront() also allows you to broadcast messages to all tasks waiting on the queue.

Added conditional compilation so that when OS_Q_POST_FRONT_EN is set to 1 in OS_CFG.H, the code for OSQPostFront() will be produced. This allows you to reduce the amount of code space. The reason this conditional compilation has been added is because I added the more powerful function OSQPostOpt().

Added OSQPostOpt() which can emulate both OSQPost() and OSQPostFront() and also allows you to broadcast messages to all tasks

waiting on the queue. The #define constant OS_Q_POST_OPT_EN found in OS_CFG.H allows you to enable (when 1) or disable (when 0) this feature.

Added conditional compilation so that when OS_Q_QUERY_EN is set to 1 in OS_CFG.H, the code for OSQQuery() will be produced. This allows you to reduce the amount of code space.

Added conditional compilation to allow you to declare storage for a single message queue. In other words, you are now allowed to set OS_MAX_QS to 1 in OS_CFG.H.

OS_SEM.C:

Added conditional compilation so that when OS_SEM_ACCEPT_EN is set to 1 in OS_CFG.H, the code for OSSemAccept() will be produced.

Added conditional compilation so that when OS_SEM_QUERY_EN is set to 1 in OS_CFG.H, the code for OSSemQuery() will be produced. This allows you to reduce the amount of code space.

OS_TASK.C:

Added call to OS_FlagUnlink() in OSTaskDel() to support event flags. Note that this code is conditionally compiled in when OS_FLAG_EN is set to 1.

Added conditional compilation so that when OS_TASK_QUERY_EN is set to 1 in OS_CFG.H, the code for OSTaskQuery() will be produced. This allows you to reduce the amount of code space.

OS_TIME.C:

Added conditional compilation so that when OS_TIME_DLY_HMSM_EN is set to 1 in OS_CFG.H, the code for OSTimeDlyHMSM() will be produced. This allows you to reduce the amount of code space in case you chose not to use this function.

Added conditional compilation so that when OS_TIME_DLY_RESUME_EN is set to 1 in OS_CFG.H, the code for OSTimeDlyResume() will be produced. This allows you to reduce the amount of code space in case you chose not to use this function.

Added conditional compilation so that when OS_TIME_GET_SET_EN is set to 1 in OS_CFG.H, the code for OSTimeGet() and OSTimeSet() will be produced. This allows you to reduce the amount of code space in case you chose not to use this function.

uCOS_II.C:

Added OS_FLAG.C.

uCOS_II.H:

Changed OS_VERSION to 205.

Added constants, data types and function prototypes to support Event Flags.

Added OS_POST_OPT_??? which are the options to specify in OSMboxPostOpt() and OSQPostOpt() calls.

The global variable OSTime is not allocated when OS_TIME_GET_SET_EN is set to 0. This reduces the RAM footprint by 4 bytes.

Added checks at the end of uCOS_II.H to ensure that you don't forget any #defines that are assumed to be declared in OS_CFG.H. If you do forget any of the required #defines in OS_CFG.H, the compiler will issue an error message. In other words, your compiler should complain about the fact that you didn't specify all the necessary #defines.

New Algorithm For Ports:

V2.51 comes with a new algorithm which prevents from having to adjust the stack pointer in `OSIntCtxSw()` and thus making the port independent of compilers and compiler options.

You should still be able to use your OLD (V2.04 and earlier) ports without change (except you'll need to add a few HOOK functions as described in the next section.

This new algorithm affects ALL your ISRs and thus you MUST play close attention to the following changes.

The OLD pseudo code for `OSIntCtxSw()` was:

```
OSIntCtxSw():                   /* OLD */
      Adjust the SP to remove call to OSIntExit(),
          locals in OSIntExit() and the call to OSIntCtxSw();
      Save the stack pointer to OSTCBCur->OSTCBStkPtr;
      Call OSTaskSwHook()
      OSTCBCur          = OSTCBHighRdy;
      OSPrioCur         = OSPrioHighRdy;
      CPU Stack Pointer = OSTCBHighRdy->OSTCBStkPtr;
      POP all the CPU registers from the new task's stack;
      Execute a return from interrupt instruction;
```

The **NEW** pseudo code for `OSIntCtxSw()` is now:

```
OSIntCtxSw():                   /* NEW */
      Call OSTaskSwHook()
      OSTCBCur          = OSTCBHighRdy;
      OSPrioCur         = OSPrioHighRdy;
      CPU Stack Pointer = OSTCBHighRdy->OSTCBStkPtr;
      POP all the CPU registers from the new task's stack;
      Execute a return from interrupt instruction;
```

You should notice that you NO LONGER need to adjust the SP. The reason this is possible is because, the SP of the task that can be switched out now NEEDS to be saved in ALL the ISRs as described below.

You MUST now change ALL your ISRs. The OLD pseudo code for your ISRs was:

```
YourISR():                  /* OLD */
      Save processor registers onto current task's stack;
      Call OSIntEnter() or increment OSIntNesting;
      .
      YOUR ISR Handler code;
      .
      Call OSIntExit();
      Restore processor registers from current task's stack;
      Execute a return from interrupt instruction;
```

The **NEW** pseudo code for `OSIntCtxSw()` is now:

```
YourISR():                  /* NEW */
      Save processor registers onto current task's stack;
      Call OSIntEnter() or increment OSIntNesting;
      if (OSIntNesting == 1) {
          Save the CPU's Stack Pointer onto current task's stack;
      }
      .
      YOUR ISR Handler code;
      .
      Call OSIntExit();
      Restore processor registers from current task's stack;
      Execute a return from interrupt instruction;
```

Upgrading from V2.04 (or earlier) to V2.51:

You should be able to use processor ports made for V2.04 or earlier. Because I added new features, you will most likely need to change the following files:

1) OS_CFG.H:
   You will need to ADD all the new #define constants and also, declare the data type OS_FLAGS. As I mentioned previously, you can simply copy one of the OS_CFG.H files supplied with this release and paste it into your own and make the appropriate selection of features you need in your product.

2) OS_CPU_C.C:
   You will need to ADD an empty function for OSTaskIdleHook() as follows unless you actually want to add your own code to the function:

```
void OSTaskIdleHook (void)
{
}
```

3) OS_CPU_A.ASM:
   If you want to use the new ALGORITHM described in the previous section, you will need to change OSIntCtxSw(), OSTickISR() **AND** all your ISRs. You should be able to use your OLD ports without change if you don't want to use the new algorithm.

4) OS_CPU.H:
   No change.

5) Your ISRs:
   If you want to use the new ALGORITHM described in the previous section, you will need to change ALL your ISRs. You should be able to use your OLD ports without change if you don't want to use the new algorithm.

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.

This page is intentionally blank.

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.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.