/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* uC/OS-II

\* The Real-Time Kernel

\* MUTUAL EXCLUSION SEMAPHORE MANAGEMENT

\*

\* (c) Copyright 1992-2009, Micrium, Weston, FL

\* All Rights Reserved

\*

\* File : OS\_MUTEX.C

\* By : Jean J. Labrosse

\* Version : V2.91

\*

\* LICENSING TERMS:

\* ---------------

\* uC/OS-II is provided in source form for FREE evaluation, for educational use or for peaceful research.

\* If you plan on using uC/OS-II in a commercial product you need to contact Micri祄 to properly license

\* its use in your product. We provide ALL the source code for your convenience and to help you experience

\* uC/OS-II. The fact that the source is provided does NOT mean that you can use it without paying a

\* licensing fee.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#ifndef OS\_MASTER\_FILE

#include <ucos\_ii.h>

#endif

#if OS\_MUTEX\_EN > 0u

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LOCAL CONSTANTS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#define OS\_MUTEX\_KEEP\_LOWER\_8 ((INT16U)0x00FFu)

#define OS\_MUTEX\_KEEP\_UPPER\_8 ((INT16U)0xFF00u)

#define OS\_MUTEX\_AVAILABLE ((INT16U)0x00FFu)

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LOCAL CONSTANTS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

static void OSMutex\_RdyAtPrio(OS\_TCB \*ptcb, INT8U prio);

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ACCEPT MUTUAL EXCLUSION SEMAPHORE

\*

\* Description: This function checks the mutual exclusion semaphore to see if a resource is available.

\* Unlike OSMutexPend(), OSMutexAccept() does not suspend the calling task if the resource is

\* not available or the event did not occur.

\*

\* Arguments : pevent is a pointer to the event control block

\*

\* perr is a pointer to an error code which will be returned to your application:

\* OS\_ERR\_NONE if the call was successful.

\* OS\_ERR\_EVENT\_TYPE if 'pevent' is not a pointer to a mutex

\* OS\_ERR\_PEVENT\_NULL 'pevent' is a NULL pointer

\* OS\_ERR\_PEND\_ISR if you called this function from an ISR

\* OS\_ERR\_PIP\_LOWER If the priority of the task that owns the Mutex is

\* HIGHER (i.e. a lower number) than the PIP. This error

\* indicates that you did not set the PIP higher (lower

\* number) than ALL the tasks that compete for the Mutex.

\* Unfortunately, this is something that could not be

\* detected when the Mutex is created because we don't know

\* what tasks will be using the Mutex.

\*

\* Returns : == OS\_TRUE if the resource is available, the mutual exclusion semaphore is acquired

\* == OS\_FALSE a) if the resource is not available

\* b) you didn't pass a pointer to a mutual exclusion semaphore

\* c) you called this function from an ISR

\*

\* Warning(s) : This function CANNOT be called from an ISR because mutual exclusion semaphores are

\* intended to be used by tasks only.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_MUTEX\_ACCEPT\_EN > 0u

BOOLEAN OSMutexAccept (OS\_EVENT \*pevent,

INT8U \*perr)

{

INT8U pip; /\* Priority Inheritance Priority (PIP) \*/

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#ifdef OS\_SAFETY\_CRITICAL

if (perr == (INT8U \*)0) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

\*perr = OS\_ERR\_PEVENT\_NULL;

return (OS\_FALSE);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_MUTEX) { /\* Validate event block type \*/

\*perr = OS\_ERR\_EVENT\_TYPE;

return (OS\_FALSE);

}

if (OSIntNesting > 0u) { /\* Make sure it's not called from an ISR \*/

\*perr = OS\_ERR\_PEND\_ISR;

return (OS\_FALSE);

}

OS\_ENTER\_CRITICAL(); /\* Get value (0 or 1) of Mutex \*/

pip = (INT8U)(pevent->OSEventCnt >> 8u); /\* Get PIP from mutex \*/

if ((pevent->OSEventCnt & OS\_MUTEX\_KEEP\_LOWER\_8) == OS\_MUTEX\_AVAILABLE) {

pevent->OSEventCnt &= OS\_MUTEX\_KEEP\_UPPER\_8; /\* Mask off LSByte (Acquire Mutex) \*/

pevent->OSEventCnt |= OSTCBCur->OSTCBPrio; /\* Save current task priority in LSByte \*/

pevent->OSEventPtr = (void \*)OSTCBCur; /\* Link TCB of task owning Mutex \*/

if (OSTCBCur->OSTCBPrio <= pip) { /\* PIP 'must' have a SMALLER prio ... \*/

OS\_EXIT\_CRITICAL(); /\* ... than current task! \*/

\*perr = OS\_ERR\_PIP\_LOWER;

} else {

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

}

return (OS\_TRUE);

}

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

return (OS\_FALSE);

}

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* CREATE A MUTUAL EXCLUSION SEMAPHORE

\*

\* Description: This function creates a mutual exclusion semaphore.

\*

\* Arguments : prio is the priority to use when accessing the mutual exclusion semaphore. In

\* other words, when the semaphore is acquired and a higher priority task

\* attempts to obtain the semaphore then the priority of the task owning the

\* semaphore is raised to this priority. It is assumed that you will specify

\* a priority that is LOWER in value than ANY of the tasks competing for the

\* mutex.

\*

\* perr is a pointer to an error code which will be returned to your application:

\* OS\_ERR\_NONE if the call was successful.

\* OS\_ERR\_CREATE\_ISR if you attempted to create a MUTEX from an ISR

\* OS\_ERR\_PRIO\_EXIST if a task at the priority inheritance priority

\* already exist.

\* OS\_ERR\_PEVENT\_NULL No more event control blocks available.

\* OS\_ERR\_PRIO\_INVALID if the priority you specify is higher that the

\* maximum allowed (i.e. > OS\_LOWEST\_PRIO)

\*

\* Returns : != (void \*)0 is a pointer to the event control clock (OS\_EVENT) associated with the

\* created mutex.

\* == (void \*)0 if an error is detected.

\*

\* Note(s) : 1) The LEAST significant 8 bits of '.OSEventCnt' are used to hold the priority number

\* of the task owning the mutex or 0xFF if no task owns the mutex.

\*

\* 2) The MOST significant 8 bits of '.OSEventCnt' are used to hold the priority number

\* to use to reduce priority inversion.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

OS\_EVENT \*OSMutexCreate (INT8U prio,

INT8U \*perr)

{

OS\_EVENT \*pevent;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#ifdef OS\_SAFETY\_CRITICAL

if (perr == (INT8U \*)0) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

#ifdef OS\_SAFETY\_CRITICAL\_IEC61508

if (OSSafetyCriticalStartFlag == OS\_TRUE) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (prio >= OS\_LOWEST\_PRIO) { /\* Validate PIP \*/

\*perr = OS\_ERR\_PRIO\_INVALID;

return ((OS\_EVENT \*)0);

}

#endif

if (OSIntNesting > 0u) { /\* See if called from ISR ... \*/

\*perr = OS\_ERR\_CREATE\_ISR; /\* ... can't CREATE mutex from an ISR \*/

return ((OS\_EVENT \*)0);

}

OS\_ENTER\_CRITICAL();

if (OSTCBPrioTbl[prio] != (OS\_TCB \*)0) { /\* Mutex priority must not already exist \*/

OS\_EXIT\_CRITICAL(); /\* Task already exist at priority ... \*/

\*perr = OS\_ERR\_PRIO\_EXIST; /\* ... inheritance priority \*/

return ((OS\_EVENT \*)0);

}

OSTCBPrioTbl[prio] = OS\_TCB\_RESERVED; /\* Reserve the table entry \*/

pevent = OSEventFreeList; /\* Get next free event control block \*/

if (pevent == (OS\_EVENT \*)0) { /\* See if an ECB was available \*/

OSTCBPrioTbl[prio] = (OS\_TCB \*)0; /\* No, Release the table entry \*/

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_PEVENT\_NULL; /\* No more event control blocks \*/

return (pevent);

}

OSEventFreeList = (OS\_EVENT \*)OSEventFreeList->OSEventPtr; /\* Adjust the free list \*/

OS\_EXIT\_CRITICAL();

pevent->OSEventType = OS\_EVENT\_TYPE\_MUTEX;

pevent->OSEventCnt = (INT16U)((INT16U)prio << 8u) | OS\_MUTEX\_AVAILABLE; /\* Resource is avail. \*/

pevent->OSEventPtr = (void \*)0; /\* No task owning the mutex \*/

#if OS\_EVENT\_NAME\_EN > 0u

pevent->OSEventName = (INT8U \*)(void \*)"?";

#endif

OS\_EventWaitListInit(pevent);

\*perr = OS\_ERR\_NONE;

return (pevent);

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DELETE A MUTEX

\*

\* Description: This function deletes a mutual exclusion semaphore and readies all tasks pending on the it.

\*

\* Arguments : pevent is a pointer to the event control block associated with the desired mutex.

\*

\* opt determines delete options as follows:

\* opt == OS\_DEL\_NO\_PEND Delete mutex ONLY if no task pending

\* opt == OS\_DEL\_ALWAYS Deletes the mutex even if tasks are waiting.

\* In this case, all the tasks pending will be readied.

\*

\* perr is a pointer to an error code that can contain one of the following values:

\* OS\_ERR\_NONE The call was successful and the mutex was deleted

\* OS\_ERR\_DEL\_ISR If you attempted to delete the MUTEX from an ISR

\* OS\_ERR\_INVALID\_OPT An invalid option was specified

\* OS\_ERR\_TASK\_WAITING One or more tasks were waiting on the mutex

\* OS\_ERR\_EVENT\_TYPE If you didn't pass a pointer to a mutex

\* OS\_ERR\_PEVENT\_NULL If 'pevent' is a NULL pointer.

\*

\* Returns : pevent upon error

\* (OS\_EVENT \*)0 if the mutex was successfully deleted.

\*

\* Note(s) : 1) This function must be used with care. Tasks that would normally expect the presence of

\* the mutex MUST check the return code of OSMutexPend().

\*

\* 2) This call can potentially disable interrupts for a long time. The interrupt disable

\* time is directly proportional to the number of tasks waiting on the mutex.

\*

\* 3) Because ALL tasks pending on the mutex will be readied, you MUST be careful because the

\* resource(s) will no longer be guarded by the mutex.

\*

\* 4) IMPORTANT: In the 'OS\_DEL\_ALWAYS' case, we assume that the owner of the Mutex (if there

\* is one) is ready-to-run and is thus NOT pending on another kernel object or

\* has delayed itself. In other words, if a task owns the mutex being deleted,

\* that task will be made ready-to-run at its original priority.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_MUTEX\_DEL\_EN > 0u

OS\_EVENT \*OSMutexDel (OS\_EVENT \*pevent,

INT8U opt,

INT8U \*perr)

{

BOOLEAN tasks\_waiting;

OS\_EVENT \*pevent\_return;

INT8U pip; /\* Priority inheritance priority \*/

INT8U prio;

OS\_TCB \*ptcb;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#ifdef OS\_SAFETY\_CRITICAL

if (perr == (INT8U \*)0) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

\*perr = OS\_ERR\_PEVENT\_NULL;

return (pevent);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_MUTEX) { /\* Validate event block type \*/

\*perr = OS\_ERR\_EVENT\_TYPE;

return (pevent);

}

if (OSIntNesting > 0u) { /\* See if called from ISR ... \*/

\*perr = OS\_ERR\_DEL\_ISR; /\* ... can't DELETE from an ISR \*/

return (pevent);

}

OS\_ENTER\_CRITICAL();

if (pevent->OSEventGrp != 0u) { /\* See if any tasks waiting on mutex \*/

tasks\_waiting = OS\_TRUE; /\* Yes \*/

} else {

tasks\_waiting = OS\_FALSE; /\* No \*/

}

switch (opt) {

case OS\_DEL\_NO\_PEND: /\* DELETE MUTEX ONLY IF NO TASK WAITING --- \*/

if (tasks\_waiting == OS\_FALSE) {

#if OS\_EVENT\_NAME\_EN > 0u

pevent->OSEventName = (INT8U \*)(void \*)"?";

#endif

pip = (INT8U)(pevent->OSEventCnt >> 8u);

OSTCBPrioTbl[pip] = (OS\_TCB \*)0; /\* Free up the PIP \*/

pevent->OSEventType = OS\_EVENT\_TYPE\_UNUSED;

pevent->OSEventPtr = OSEventFreeList; /\* Return Event Control Block to free list \*/

pevent->OSEventCnt = 0u;

OSEventFreeList = pevent;

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

pevent\_return = (OS\_EVENT \*)0; /\* Mutex has been deleted \*/

} else {

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_TASK\_WAITING;

pevent\_return = pevent;

}

break;

case OS\_DEL\_ALWAYS: /\* ALWAYS DELETE THE MUTEX ---------------- \*/

pip = (INT8U)(pevent->OSEventCnt >> 8u); /\* Get PIP of mutex \*/

prio = (INT8U)(pevent->OSEventCnt & OS\_MUTEX\_KEEP\_LOWER\_8); /\* Get owner's original prio \*/

ptcb = (OS\_TCB \*)pevent->OSEventPtr;

if (ptcb != (OS\_TCB \*)0) { /\* See if any task owns the mutex \*/

if (ptcb->OSTCBPrio == pip) { /\* See if original prio was changed \*/

OSMutex\_RdyAtPrio(ptcb, prio); /\* Yes, Restore the task's original prio \*/

}

}

while (pevent->OSEventGrp != 0u) { /\* Ready ALL tasks waiting for mutex \*/

(void)OS\_EventTaskRdy(pevent, (void \*)0, OS\_STAT\_MUTEX, OS\_STAT\_PEND\_OK);

}

#if OS\_EVENT\_NAME\_EN > 0u

pevent->OSEventName = (INT8U \*)(void \*)"?";

#endif

pip = (INT8U)(pevent->OSEventCnt >> 8u);

OSTCBPrioTbl[pip] = (OS\_TCB \*)0; /\* Free up the PIP \*/

pevent->OSEventType = OS\_EVENT\_TYPE\_UNUSED;

pevent->OSEventPtr = OSEventFreeList; /\* Return Event Control Block to free list \*/

pevent->OSEventCnt = 0u;

OSEventFreeList = pevent; /\* Get next free event control block \*/

OS\_EXIT\_CRITICAL();

if (tasks\_waiting == OS\_TRUE) { /\* Reschedule only if task(s) were waiting \*/

OS\_Sched(); /\* Find highest priority task ready to run \*/

}

\*perr = OS\_ERR\_NONE;

pevent\_return = (OS\_EVENT \*)0; /\* Mutex has been deleted \*/

break;

default:

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_INVALID\_OPT;

pevent\_return = pevent;

break;

}

return (pevent\_return);

}

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PEND ON MUTUAL EXCLUSION SEMAPHORE

\*

\* Description: This function waits for a mutual exclusion semaphore.

\*

\* Arguments : pevent is a pointer to the event control block associated with the desired

\* mutex.

\*

\* timeout is an optional timeout period (in clock ticks). If non-zero, your task will

\* wait for the resource up to the amount of time specified by this argument.

\* If you specify 0, however, your task will wait forever at the specified

\* mutex or, until the resource becomes available.

\*

\* perr is a pointer to where an error message will be deposited. Possible error

\* messages are:

\* OS\_ERR\_NONE The call was successful and your task owns the mutex

\* OS\_ERR\_TIMEOUT The mutex was not available within the specified 'timeout'.

\* OS\_ERR\_PEND\_ABORT The wait on the mutex was aborted.

\* OS\_ERR\_EVENT\_TYPE If you didn't pass a pointer to a mutex

\* OS\_ERR\_PEVENT\_NULL 'pevent' is a NULL pointer

\* OS\_ERR\_PEND\_ISR If you called this function from an ISR and the result

\* would lead to a suspension.

\* OS\_ERR\_PIP\_LOWER If the priority of the task that owns the Mutex is

\* HIGHER (i.e. a lower number) than the PIP. This error

\* indicates that you did not set the PIP higher (lower

\* number) than ALL the tasks that compete for the Mutex.

\* Unfortunately, this is something that could not be

\* detected when the Mutex is created because we don't know

\* what tasks will be using the Mutex.

\* OS\_ERR\_PEND\_LOCKED If you called this function when the scheduler is locked

\*

\* Returns : none

\*

\* Note(s) : 1) The task that owns the Mutex MUST NOT pend on any other event while it owns the mutex.

\*

\* 2) You MUST NOT change the priority of the task that owns the mutex

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void OSMutexPend (OS\_EVENT \*pevent,

INT32U timeout,

INT8U \*perr)

{

INT8U pip; /\* Priority Inheritance Priority (PIP) \*/

INT8U mprio; /\* Mutex owner priority \*/

BOOLEAN rdy; /\* Flag indicating task was ready \*/

OS\_TCB \*ptcb;

OS\_EVENT \*pevent2;

INT8U y;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#ifdef OS\_SAFETY\_CRITICAL

if (perr == (INT8U \*)0) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

\*perr = OS\_ERR\_PEVENT\_NULL;

return;

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_MUTEX) { /\* Validate event block type \*/

\*perr = OS\_ERR\_EVENT\_TYPE;

return;

}

if (OSIntNesting > 0u) { /\* See if called from ISR ... \*/

\*perr = OS\_ERR\_PEND\_ISR; /\* ... can't PEND from an ISR \*/

return;

}

if (OSLockNesting > 0u) { /\* See if called with scheduler locked ... \*/

\*perr = OS\_ERR\_PEND\_LOCKED; /\* ... can't PEND when locked \*/

return;

}

OS\_ENTER\_CRITICAL();

pip = (INT8U)(pevent->OSEventCnt >> 8u); /\* Get PIP from mutex \*/

/\* Is Mutex available? \*/

if ((INT8U)(pevent->OSEventCnt & OS\_MUTEX\_KEEP\_LOWER\_8) == OS\_MUTEX\_AVAILABLE) {

pevent->OSEventCnt &= OS\_MUTEX\_KEEP\_UPPER\_8; /\* Yes, Acquire the resource \*/

pevent->OSEventCnt |= OSTCBCur->OSTCBPrio; /\* Save priority of owning task \*/

pevent->OSEventPtr = (void \*)OSTCBCur; /\* Point to owning task's OS\_TCB \*/

if (OSTCBCur->OSTCBPrio <= pip) { /\* PIP 'must' have a SMALLER prio ... \*/

OS\_EXIT\_CRITICAL(); /\* ... than current task! \*/

\*perr = OS\_ERR\_PIP\_LOWER;

} else {

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

}

return;

}

mprio = (INT8U)(pevent->OSEventCnt & OS\_MUTEX\_KEEP\_LOWER\_8); /\* No, Get priority of mutex owner \*/

ptcb = (OS\_TCB \*)(pevent->OSEventPtr); /\* Point to TCB of mutex owner \*/

if (ptcb->OSTCBPrio > pip) { /\* Need to promote prio of owner?\*/

if (mprio > OSTCBCur->OSTCBPrio) {

y = ptcb->OSTCBY;

if ((OSRdyTbl[y] & ptcb->OSTCBBitX) != 0u) { /\* See if mutex owner is ready \*/

OSRdyTbl[y] &= (OS\_PRIO)~ptcb->OSTCBBitX; /\* Yes, Remove owner from Rdy ...\*/

if (OSRdyTbl[y] == 0u) { /\* ... list at current prio \*/

OSRdyGrp &= (OS\_PRIO)~ptcb->OSTCBBitY;

}

rdy = OS\_TRUE;

} else {

pevent2 = ptcb->OSTCBEventPtr;

if (pevent2 != (OS\_EVENT \*)0) { /\* Remove from event wait list \*/

y = ptcb->OSTCBY;

pevent2->OSEventTbl[y] &= (OS\_PRIO)~ptcb->OSTCBBitX;

if (pevent2->OSEventTbl[y] == 0u) {

pevent2->OSEventGrp &= (OS\_PRIO)~ptcb->OSTCBBitY;

}

}

rdy = OS\_FALSE; /\* No \*/

}

ptcb->OSTCBPrio = pip; /\* Change owner task prio to PIP \*/

#if OS\_LOWEST\_PRIO <= 63u

ptcb->OSTCBY = (INT8U)( ptcb->OSTCBPrio >> 3u);

ptcb->OSTCBX = (INT8U)( ptcb->OSTCBPrio & 0x07u);

#else

ptcb->OSTCBY = (INT8U)((INT8U)(ptcb->OSTCBPrio >> 4u) & 0xFFu);

ptcb->OSTCBX = (INT8U)( ptcb->OSTCBPrio & 0x0Fu);

#endif

ptcb->OSTCBBitY = (OS\_PRIO)(1uL << ptcb->OSTCBY);

ptcb->OSTCBBitX = (OS\_PRIO)(1uL << ptcb->OSTCBX);

if (rdy == OS\_TRUE) { /\* If task was ready at owner's priority ...\*/

OSRdyGrp |= ptcb->OSTCBBitY; /\* ... make it ready at new priority. \*/

OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;

} else {

pevent2 = ptcb->OSTCBEventPtr;

if (pevent2 != (OS\_EVENT \*)0) { /\* Add to event wait list \*/

pevent2->OSEventGrp |= ptcb->OSTCBBitY;

pevent2->OSEventTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;

}

}

OSTCBPrioTbl[pip] = ptcb;

}

}

OSTCBCur->OSTCBStat |= OS\_STAT\_MUTEX; /\* Mutex not available, pend current task \*/

OSTCBCur->OSTCBStatPend = OS\_STAT\_PEND\_OK;

OSTCBCur->OSTCBDly = timeout; /\* Store timeout in current task's TCB \*/

OS\_EventTaskWait(pevent); /\* Suspend task until event or timeout occurs \*/

OS\_EXIT\_CRITICAL();

OS\_Sched(); /\* Find next highest priority task ready \*/

OS\_ENTER\_CRITICAL();

switch (OSTCBCur->OSTCBStatPend) { /\* See if we timed-out or aborted \*/

case OS\_STAT\_PEND\_OK:

\*perr = OS\_ERR\_NONE;

break;

case OS\_STAT\_PEND\_ABORT:

\*perr = OS\_ERR\_PEND\_ABORT; /\* Indicate that we aborted getting mutex \*/

break;

case OS\_STAT\_PEND\_TO:

default:

OS\_EventTaskRemove(OSTCBCur, pevent);

\*perr = OS\_ERR\_TIMEOUT; /\* Indicate that we didn't get mutex within TO \*/

break;

}

OSTCBCur->OSTCBStat = OS\_STAT\_RDY; /\* Set task status to ready \*/

OSTCBCur->OSTCBStatPend = OS\_STAT\_PEND\_OK; /\* Clear pend status \*/

OSTCBCur->OSTCBEventPtr = (OS\_EVENT \*)0; /\* Clear event pointers \*/

#if (OS\_EVENT\_MULTI\_EN > 0u)

OSTCBCur->OSTCBEventMultiPtr = (OS\_EVENT \*\*)0;

#endif

OS\_EXIT\_CRITICAL();

}

/\*$PAGE\*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* POST TO A MUTUAL EXCLUSION SEMAPHORE

\*

\* Description: This function signals a mutual exclusion semaphore

\*

\* Arguments : pevent is a pointer to the event control block associated with the desired

\* mutex.

\*

\* Returns : OS\_ERR\_NONE The call was successful and the mutex was signaled.

\* OS\_ERR\_EVENT\_TYPE If you didn't pass a pointer to a mutex

\* OS\_ERR\_PEVENT\_NULL 'pevent' is a NULL pointer

\* OS\_ERR\_POST\_ISR Attempted to post from an ISR (not valid for MUTEXes)

\* OS\_ERR\_NOT\_MUTEX\_OWNER The task that did the post is NOT the owner of the MUTEX.

\* OS\_ERR\_PIP\_LOWER If the priority of the new task that owns the Mutex is

\* HIGHER (i.e. a lower number) than the PIP. This error

\* indicates that you did not set the PIP higher (lower

\* number) than ALL the tasks that compete for the Mutex.

\* Unfortunately, this is something that could not be

\* detected when the Mutex is created because we don't know

\* what tasks will be using the Mutex.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

INT8U OSMutexPost (OS\_EVENT \*pevent)

{

INT8U pip; /\* Priority inheritance priority \*/

INT8U prio;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

if (OSIntNesting > 0u) { /\* See if called from ISR ... \*/

return (OS\_ERR\_POST\_ISR); /\* ... can't POST mutex from an ISR \*/

}

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

return (OS\_ERR\_PEVENT\_NULL);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_MUTEX) { /\* Validate event block type \*/

return (OS\_ERR\_EVENT\_TYPE);

}

OS\_ENTER\_CRITICAL();

pip = (INT8U)(pevent->OSEventCnt >> 8u); /\* Get priority inheritance priority of mutex \*/

prio = (INT8U)(pevent->OSEventCnt & OS\_MUTEX\_KEEP\_LOWER\_8); /\* Get owner's original priority \*/

if (OSTCBCur != (OS\_TCB \*)pevent->OSEventPtr) { /\* See if posting task owns the MUTEX \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NOT\_MUTEX\_OWNER);

}

if (OSTCBCur->OSTCBPrio == pip) { /\* Did we have to raise current task's priority? \*/

OSMutex\_RdyAtPrio(OSTCBCur, prio); /\* Restore the task's original priority \*/

}

OSTCBPrioTbl[pip] = OS\_TCB\_RESERVED; /\* Reserve table entry \*/

if (pevent->OSEventGrp != 0u) { /\* Any task waiting for the mutex? \*/

/\* Yes, Make HPT waiting for mutex ready \*/

prio = OS\_EventTaskRdy(pevent, (void \*)0, OS\_STAT\_MUTEX, OS\_STAT\_PEND\_OK);

pevent->OSEventCnt &= OS\_MUTEX\_KEEP\_UPPER\_8; /\* Save priority of mutex's new owner \*/

pevent->OSEventCnt |= prio;

pevent->OSEventPtr = OSTCBPrioTbl[prio]; /\* Link to new mutex owner's OS\_TCB \*/

if (prio <= pip) { /\* PIP 'must' have a SMALLER prio ... \*/

OS\_EXIT\_CRITICAL(); /\* ... than current task! \*/

OS\_Sched(); /\* Find highest priority task ready to run \*/

return (OS\_ERR\_PIP\_LOWER);

} else {

OS\_EXIT\_CRITICAL();

OS\_Sched(); /\* Find highest priority task ready to run \*/

return (OS\_ERR\_NONE);

}

}

pevent->OSEventCnt |= OS\_MUTEX\_AVAILABLE; /\* No, Mutex is now available \*/

pevent->OSEventPtr = (void \*)0;

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* QUERY A MUTUAL EXCLUSION SEMAPHORE

\*

\* Description: This function obtains information about a mutex

\*

\* Arguments : pevent is a pointer to the event control block associated with the desired mutex

\*

\* p\_mutex\_data is a pointer to a structure that will contain information about the mutex

\*

\* Returns : OS\_ERR\_NONE The call was successful and the message was sent

\* OS\_ERR\_QUERY\_ISR If you called this function from an ISR

\* OS\_ERR\_PEVENT\_NULL If 'pevent' is a NULL pointer

\* OS\_ERR\_PDATA\_NULL If 'p\_mutex\_data' is a NULL pointer

\* OS\_ERR\_EVENT\_TYPE If you are attempting to obtain data from a non mutex.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_MUTEX\_QUERY\_EN > 0u

INT8U OSMutexQuery (OS\_EVENT \*pevent,

OS\_MUTEX\_DATA \*p\_mutex\_data)

{

INT8U i;

OS\_PRIO \*psrc;

OS\_PRIO \*pdest;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

if (OSIntNesting > 0u) { /\* See if called from ISR ... \*/

return (OS\_ERR\_QUERY\_ISR); /\* ... can't QUERY mutex from an ISR \*/

}

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

return (OS\_ERR\_PEVENT\_NULL);

}

if (p\_mutex\_data == (OS\_MUTEX\_DATA \*)0) { /\* Validate 'p\_mutex\_data' \*/

return (OS\_ERR\_PDATA\_NULL);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_MUTEX) { /\* Validate event block type \*/

return (OS\_ERR\_EVENT\_TYPE);

}

OS\_ENTER\_CRITICAL();

p\_mutex\_data->OSMutexPIP = (INT8U)(pevent->OSEventCnt >> 8u);

p\_mutex\_data->OSOwnerPrio = (INT8U)(pevent->OSEventCnt & OS\_MUTEX\_KEEP\_LOWER\_8);

if (p\_mutex\_data->OSOwnerPrio == 0xFFu) {

p\_mutex\_data->OSValue = OS\_TRUE;

} else {

p\_mutex\_data->OSValue = OS\_FALSE;

}

p\_mutex\_data->OSEventGrp = pevent->OSEventGrp; /\* Copy wait list \*/

psrc = &pevent->OSEventTbl[0];

pdest = &p\_mutex\_data->OSEventTbl[0];

for (i = 0u; i < OS\_EVENT\_TBL\_SIZE; i++) {

\*pdest++ = \*psrc++;

}

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

#endif /\* OS\_MUTEX\_QUERY\_EN \*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* RESTORE A TASK BACK TO ITS ORIGINAL PRIORITY

\*

\* Description: This function makes a task ready at the specified priority

\*

\* Arguments : ptcb is a pointer to OS\_TCB of the task to make ready

\*

\* prio is the desired priority

\*

\* Returns : none

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

static void OSMutex\_RdyAtPrio (OS\_TCB \*ptcb,

INT8U prio)

{

INT8U y;

y = ptcb->OSTCBY; /\* Remove owner from ready list at 'pip' \*/

OSRdyTbl[y] &= (OS\_PRIO)~ptcb->OSTCBBitX;

if (OSRdyTbl[y] == 0u) {

OSRdyGrp &= (OS\_PRIO)~ptcb->OSTCBBitY;

}

ptcb->OSTCBPrio = prio;

OSPrioCur = prio; /\* The current task is now at this priority \*/

#if OS\_LOWEST\_PRIO <= 63u

ptcb->OSTCBY = (INT8U)((INT8U)(prio >> 3u) & 0x07u);

ptcb->OSTCBX = (INT8U)(prio & 0x07u);

#else

ptcb->OSTCBY = (INT8U)((INT8U)(prio >> 4u) & 0x0Fu);

ptcb->OSTCBX = (INT8U) (prio & 0x0Fu);

#endif

ptcb->OSTCBBitY = (OS\_PRIO)(1uL << ptcb->OSTCBY);

ptcb->OSTCBBitX = (OS\_PRIO)(1uL << ptcb->OSTCBX);

OSRdyGrp |= ptcb->OSTCBBitY; /\* Make task ready at original priority \*/

OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;

OSTCBPrioTbl[prio] = ptcb;

}

#endif /\* OS\_MUTEX\_EN \*/