/\*

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

\* uC/OS-II

\* The Real-Time Kernel

\* MESSAGE MAILBOX MANAGEMENT

\*

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

\* All Rights Reserved

\*

\* File : OS\_MBOX.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\_MBOX\_EN > 0u

/\*

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

\* ACCEPT MESSAGE FROM MAILBOX

\*

\* Description: This function checks the mailbox to see if a message is available. Unlike OSMboxPend(),

\* OSMboxAccept() does not suspend the calling task if a message is not available.

\*

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

\*

\* Returns : != (void \*)0 is the message in the mailbox if one is available. The mailbox is cleared

\* so the next time OSMboxAccept() is called, the mailbox will be empty.

\* == (void \*)0 if the mailbox is empty or,

\* if 'pevent' is a NULL pointer or,

\* if you didn't pass the proper event pointer.

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

\*/

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

void \*OSMboxAccept (OS\_EVENT \*pevent)

{

void \*pmsg;

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

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

#endif

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

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

return ((void \*)0);

}

#endif

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

return ((void \*)0);

}

OS\_ENTER\_CRITICAL();

pmsg = pevent->OSEventPtr;

pevent->OSEventPtr = (void \*)0; /\* Clear the mailbox \*/

OS\_EXIT\_CRITICAL();

return (pmsg); /\* Return the message received (or NULL) \*/

}

#endif

/\*

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

\* CREATE A MESSAGE MAILBOX

\*

\* Description: This function creates a message mailbox if free event control blocks are available.

\*

\* Arguments : pmsg is a pointer to a message that you wish to deposit in the mailbox. If

\* you set this value to the NULL pointer (i.e. (void \*)0) then the mailbox

\* will be considered empty.

\*

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

\* created mailbox

\* == (OS\_EVENT \*)0 if no event control blocks were available

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

\*/

OS\_EVENT \*OSMboxCreate (void \*pmsg)

{

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\_IEC61508

if (OSSafetyCriticalStartFlag == OS\_TRUE) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

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

return ((OS\_EVENT \*)0); /\* ... can't CREATE from an ISR \*/

}

OS\_ENTER\_CRITICAL();

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

if (OSEventFreeList != (OS\_EVENT \*)0) { /\* See if pool of free ECB pool was empty \*/

OSEventFreeList = (OS\_EVENT \*)OSEventFreeList->OSEventPtr;

}

OS\_EXIT\_CRITICAL();

if (pevent != (OS\_EVENT \*)0) {

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

pevent->OSEventCnt = 0u;

pevent->OSEventPtr = pmsg; /\* Deposit message in event control block \*/

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

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

#endif

OS\_EventWaitListInit(pevent);

}

return (pevent); /\* Return pointer to event control block \*/

}

/\*

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

\* DELETE A MAIBOX

\*

\* Description: This function deletes a mailbox and readies all tasks pending on the mailbox.

\*

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

\* mailbox.

\*

\* opt determines delete options as follows:

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

\* opt == OS\_DEL\_ALWAYS Deletes the mailbox 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 mailbox was deleted

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

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

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

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

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

\*

\* Returns : pevent upon error

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

\*

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

\* the mailbox MUST check the return code of OSMboxPend().

\* 2) OSMboxAccept() callers will not know that the intended mailbox has been deleted!

\* 3) 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 mailbox.

\* 4) Because ALL tasks pending on the mailbox will be readied, you MUST be careful in

\* applications where the mailbox is used for mutual exclusion because the resource(s)

\* will no longer be guarded by the mailbox.

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

\*/

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

OS\_EVENT \*OSMboxDel (OS\_EVENT \*pevent,

INT8U opt,

INT8U \*perr)

{

BOOLEAN tasks\_waiting;

OS\_EVENT \*pevent\_return;

#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\_MBOX) { /\* 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 mailbox \*/

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

} else {

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

}

switch (opt) {

case OS\_DEL\_NO\_PEND: /\* Delete mailbox only if no task waiting \*/

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

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

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

#endif

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();

\*perr = OS\_ERR\_NONE;

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

} else {

OS\_EXIT\_CRITICAL();

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

pevent\_return = pevent;

}

break;

case OS\_DEL\_ALWAYS: /\* Always delete the mailbox \*/

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

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

}

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

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

#endif

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; /\* Mailbox has been deleted \*/

break;

default:

OS\_EXIT\_CRITICAL();

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

pevent\_return = pevent;

break;

}

return (pevent\_return);

}

#endif

/\*

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

\* PEND ON MAILBOX FOR A MESSAGE

\*

\* Description: This function waits for a message to be sent to a mailbox

\*

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

\*

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

\* wait for a message to arrive at the mailbox up to the amount of time

\* specified by this argument. If you specify 0, however, your task will wait

\* forever at the specified mailbox or, until a message arrives.

\*

\* 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 received a

\* message.

\* OS\_ERR\_TIMEOUT A message was not received within the specified 'timeout'.

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

\* OS\_ERR\_EVENT\_TYPE Invalid event type

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

\* would lead to a suspension.

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

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

\*

\* Returns : != (void \*)0 is a pointer to the message received

\* == (void \*)0 if no message was received or,

\* if 'pevent' is a NULL pointer or,

\* if you didn't pass the proper pointer to the event control block.

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

\*/

void \*OSMboxPend (OS\_EVENT \*pevent,

INT32U timeout,

INT8U \*perr)

{

void \*pmsg;

#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 ((void \*)0);

}

#endif

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

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

return ((void \*)0);

}

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

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

return ((void \*)0);

}

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

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

return ((void \*)0);

}

OS\_ENTER\_CRITICAL();

pmsg = pevent->OSEventPtr;

if (pmsg != (void \*)0) { /\* See if there is already a message \*/

pevent->OSEventPtr = (void \*)0; /\* Clear the mailbox \*/

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

return (pmsg); /\* Return the message received (or NULL) \*/

}

OSTCBCur->OSTCBStat |= OS\_STAT\_MBOX; /\* Message not available, task will pend \*/

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

OSTCBCur->OSTCBDly = timeout; /\* Load timeout in TCB \*/

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

OS\_EXIT\_CRITICAL();

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

OS\_ENTER\_CRITICAL();

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

case OS\_STAT\_PEND\_OK:

pmsg = OSTCBCur->OSTCBMsg;

\*perr = OS\_ERR\_NONE;

break;

case OS\_STAT\_PEND\_ABORT:

pmsg = (void \*)0;

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

break;

case OS\_STAT\_PEND\_TO:

default:

OS\_EventTaskRemove(OSTCBCur, pevent);

pmsg = (void \*)0;

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

OSTCBCur->OSTCBMsg = (void \*)0; /\* Clear received message \*/

OS\_EXIT\_CRITICAL();

return (pmsg); /\* Return received message \*/

}

/\*

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

\* ABORT WAITING ON A MESSAGE MAILBOX

\*

\* Description: This function aborts & readies any tasks currently waiting on a mailbox. This function

\* should be used to fault-abort the wait on the mailbox, rather than to normally signal

\* the mailbox via OSMboxPost() or OSMboxPostOpt().

\*

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

\*

\* opt determines the type of ABORT performed:

\* OS\_PEND\_OPT\_NONE ABORT wait for a single task (HPT) waiting on the

\* mailbox

\* OS\_PEND\_OPT\_BROADCAST ABORT wait for ALL tasks that are waiting on the

\* mailbox

\*

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

\* messages are:

\*

\* OS\_ERR\_NONE No tasks were waiting on the mailbox.

\* OS\_ERR\_PEND\_ABORT At least one task waiting on the mailbox was readied

\* and informed of the aborted wait; check return value

\* for the number of tasks whose wait on the mailbox

\* was aborted.

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

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

\*

\* Returns : == 0 if no tasks were waiting on the mailbox, or upon error.

\* > 0 if one or more tasks waiting on the mailbox are now readied and informed.

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

\*/

#if OS\_MBOX\_PEND\_ABORT\_EN > 0u

INT8U OSMboxPendAbort (OS\_EVENT \*pevent,

INT8U opt,

INT8U \*perr)

{

INT8U nbr\_tasks;

#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 (0u);

}

#endif

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

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

return (0u);

}

OS\_ENTER\_CRITICAL();

if (pevent->OSEventGrp != 0u) { /\* See if any task waiting on mailbox? \*/

nbr\_tasks = 0u;

switch (opt) {

case OS\_PEND\_OPT\_BROADCAST: /\* Do we need to abort ALL waiting tasks? \*/

while (pevent->OSEventGrp != 0u) { /\* Yes, ready ALL tasks waiting on mailbox \*/

(void)OS\_EventTaskRdy(pevent, (void \*)0, OS\_STAT\_MBOX, OS\_STAT\_PEND\_ABORT);

nbr\_tasks++;

}

break;

case OS\_PEND\_OPT\_NONE:

default: /\* No, ready HPT waiting on mailbox \*/

(void)OS\_EventTaskRdy(pevent, (void \*)0, OS\_STAT\_MBOX, OS\_STAT\_PEND\_ABORT);

nbr\_tasks++;

break;

}

OS\_EXIT\_CRITICAL();

OS\_Sched(); /\* Find HPT ready to run \*/

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

return (nbr\_tasks);

}

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

return (0u); /\* No tasks waiting on mailbox \*/

}

#endif

/\*

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

\* POST MESSAGE TO A MAILBOX

\*

\* Description: This function sends a message to a mailbox

\*

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

\*

\* pmsg is a pointer to the message to send. You MUST NOT send a NULL pointer.

\*

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

\* OS\_ERR\_MBOX\_FULL If the mailbox already contains a message. You can can only send one

\* message at a time and thus, the message MUST be consumed before you

\* are allowed to send another one.

\* OS\_ERR\_EVENT\_TYPE If you are attempting to post to a non mailbox.

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

\* OS\_ERR\_POST\_NULL\_PTR If you are attempting to post a NULL pointer

\*

\* Note(s) : 1) HPT means Highest Priority Task

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

\*/

#if OS\_MBOX\_POST\_EN > 0u

INT8U OSMboxPost (OS\_EVENT \*pevent,

void \*pmsg)

{

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

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

#endif

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

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

return (OS\_ERR\_PEVENT\_NULL);

}

if (pmsg == (void \*)0) { /\* Make sure we are not posting a NULL pointer \*/

return (OS\_ERR\_POST\_NULL\_PTR);

}

#endif

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

return (OS\_ERR\_EVENT\_TYPE);

}

OS\_ENTER\_CRITICAL();

if (pevent->OSEventGrp != 0u) { /\* See if any task pending on mailbox \*/

/\* Ready HPT waiting on event \*/

(void)OS\_EventTaskRdy(pevent, pmsg, OS\_STAT\_MBOX, OS\_STAT\_PEND\_OK);

OS\_EXIT\_CRITICAL();

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

return (OS\_ERR\_NONE);

}

if (pevent->OSEventPtr != (void \*)0) { /\* Make sure mailbox doesn't already have a msg \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_MBOX\_FULL);

}

pevent->OSEventPtr = pmsg; /\* Place message in mailbox \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

#endif

/\*

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

\* POST MESSAGE TO A MAILBOX

\*

\* Description: This function sends a message to a mailbox

\*

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

\*

\* pmsg is a pointer to the message to send. You MUST NOT send a NULL pointer.

\*

\* opt determines the type of POST performed:

\* OS\_POST\_OPT\_NONE POST to a single waiting task

\* (Identical to OSMboxPost())

\* OS\_POST\_OPT\_BROADCAST POST to ALL tasks that are waiting on the mailbox

\*

\* OS\_POST\_OPT\_NO\_SCHED Indicates that the scheduler will NOT be invoked

\*

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

\* OS\_ERR\_MBOX\_FULL If the mailbox already contains a message. You can can only send one

\* message at a time and thus, the message MUST be consumed before you

\* are allowed to send another one.

\* OS\_ERR\_EVENT\_TYPE If you are attempting to post to a non mailbox.

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

\* OS\_ERR\_POST\_NULL\_PTR If you are attempting to post a NULL pointer

\*

\* Note(s) : 1) HPT means Highest Priority Task

\*

\* Warning : Interrupts can be disabled for a long time if you do a 'broadcast'. In fact, the

\* interrupt disable time is proportional to the number of tasks waiting on the mailbox.

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

\*/

#if OS\_MBOX\_POST\_OPT\_EN > 0u

INT8U OSMboxPostOpt (OS\_EVENT \*pevent,

void \*pmsg,

INT8U opt)

{

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

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

#endif

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

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

return (OS\_ERR\_PEVENT\_NULL);

}

if (pmsg == (void \*)0) { /\* Make sure we are not posting a NULL pointer \*/

return (OS\_ERR\_POST\_NULL\_PTR);

}

#endif

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

return (OS\_ERR\_EVENT\_TYPE);

}

OS\_ENTER\_CRITICAL();

if (pevent->OSEventGrp != 0u) { /\* See if any task pending on mailbox \*/

if ((opt & OS\_POST\_OPT\_BROADCAST) != 0x00u) { /\* Do we need to post msg to ALL waiting tasks ? \*/

while (pevent->OSEventGrp != 0u) { /\* Yes, Post to ALL tasks waiting on mailbox \*/

(void)OS\_EventTaskRdy(pevent, pmsg, OS\_STAT\_MBOX, OS\_STAT\_PEND\_OK);

}

} else { /\* No, Post to HPT waiting on mbox \*/

(void)OS\_EventTaskRdy(pevent, pmsg, OS\_STAT\_MBOX, OS\_STAT\_PEND\_OK);

}

OS\_EXIT\_CRITICAL();

if ((opt & OS\_POST\_OPT\_NO\_SCHED) == 0u) { /\* See if scheduler needs to be invoked \*/

OS\_Sched(); /\* Find HPT ready to run \*/

}

return (OS\_ERR\_NONE);

}

if (pevent->OSEventPtr != (void \*)0) { /\* Make sure mailbox doesn't already have a msg \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_MBOX\_FULL);

}

pevent->OSEventPtr = pmsg; /\* Place message in mailbox \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

#endif

/\*

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

\* QUERY A MESSAGE MAILBOX

\*

\* Description: This function obtains information about a message mailbox.

\*

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

\*

\* p\_mbox\_data is a pointer to a structure that will contain information about the message

\* mailbox.

\*

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

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

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

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

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

\*/

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

INT8U OSMboxQuery (OS\_EVENT \*pevent,

OS\_MBOX\_DATA \*p\_mbox\_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 OS\_ARG\_CHK\_EN > 0u

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

return (OS\_ERR\_PEVENT\_NULL);

}

if (p\_mbox\_data == (OS\_MBOX\_DATA \*)0) { /\* Validate 'p\_mbox\_data' \*/

return (OS\_ERR\_PDATA\_NULL);

}

#endif

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

return (OS\_ERR\_EVENT\_TYPE);

}

OS\_ENTER\_CRITICAL();

p\_mbox\_data->OSEventGrp = pevent->OSEventGrp; /\* Copy message mailbox wait list \*/

psrc = &pevent->OSEventTbl[0];

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

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

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

}

p\_mbox\_data->OSMsg = pevent->OSEventPtr; /\* Get message from mailbox \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

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

#endif /\* OS\_MBOX\_EN \*/