/\*

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

\* uC/OS-II

\* The Real-Time Kernel

\* SEMAPHORE MANAGEMENT

\*

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

\* All Rights Reserved

\*

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

/\*

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

\* ACCEPT SEMAPHORE

\*

\* Description: This function checks the semaphore to see if a resource is available or, if an event

\* occurred. Unlike OSSemPend(), OSSemAccept() 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

\*

\* Returns : > 0 if the resource is available or the event did not occur the semaphore is

\* decremented to obtain the resource.

\* == 0 if the resource is not available or the event did not occur or,

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

\* if you didn't pass a pointer to a semaphore

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

\*/

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

INT16U OSSemAccept (OS\_EVENT \*pevent)

{

INT16U cnt;

#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 (0u);

}

#endif

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

return (0u);

}

OS\_ENTER\_CRITICAL();

cnt = pevent->OSEventCnt;

if (cnt > 0u) { /\* See if resource is available \*/

pevent->OSEventCnt--; /\* Yes, decrement semaphore and notify caller \*/

}

OS\_EXIT\_CRITICAL();

return (cnt); /\* Return semaphore count \*/

}

#endif

/\*

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

\* CREATE A SEMAPHORE

\*

\* Description: This function creates a semaphore.

\*

\* Arguments : cnt is the initial value for the semaphore. If the value is 0, no resource is

\* available (or no event has occurred). You initialize the semaphore to a

\* non-zero value to specify how many resources are available (e.g. if you have

\* 10 resources, you would initialize the semaphore to 10).

\*

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

\* created semaphore

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

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

\*/

OS\_EVENT \*OSSemCreate (INT16U cnt)

{

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) { /\* Get an event control block \*/

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

pevent->OSEventCnt = cnt; /\* Set semaphore value \*/

pevent->OSEventPtr = (void \*)0; /\* Unlink from ECB free list \*/

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

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

#endif

OS\_EventWaitListInit(pevent); /\* Initialize to 'nobody waiting' on sem. \*/

}

return (pevent);

}

/\*

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

\* DELETE A SEMAPHORE

\*

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

\*

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

\* semaphore.

\*

\* opt determines delete options as follows:

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

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

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

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

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

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

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

\*

\* Returns : pevent upon error

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

\*

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

\* the semaphore MUST check the return code of OSSemPend().

\* 2) OSSemAccept() callers will not know that the intended semaphore has been deleted unless

\* they check 'pevent' to see that it's a NULL pointer.

\* 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 semaphore.

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

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

\* will no longer be guarded by the semaphore.

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

\*/

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

OS\_EVENT \*OSSemDel (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\_SEM) { /\* 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 semaphore \*/

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

} else {

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

}

switch (opt) {

case OS\_DEL\_NO\_PEND: /\* Delete semaphore 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; /\* Semaphore has been deleted \*/

} else {

OS\_EXIT\_CRITICAL();

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

pevent\_return = pevent;

}

break;

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

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

(void)OS\_EventTaskRdy(pevent, (void \*)0, OS\_STAT\_SEM, 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; /\* Semaphore has been deleted \*/

break;

default:

OS\_EXIT\_CRITICAL();

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

pevent\_return = pevent;

break;

}

return (pevent\_return);

}

#endif

/\*

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

\* PEND ON SEMAPHORE

\*

\* Description: This function waits for a semaphore.

\*

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

\* semaphore.

\*

\* 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

\* semaphore or, until the resource becomes available (or the event occurs).

\*

\* 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 resource

\* or, the event you are waiting for occurred.

\* OS\_ERR\_TIMEOUT The semaphore was not received within the specified

\* 'timeout'.

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

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

\* 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 : none

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

\*/

void OSSemPend (OS\_EVENT \*pevent,

INT32U timeout,

INT8U \*perr)

{

#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\_SEM) { /\* 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();

if (pevent->OSEventCnt > 0u) { /\* If sem. is positive, resource available ... \*/

pevent->OSEventCnt--; /\* ... decrement semaphore only if positive. \*/

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

return;

}

/\* Otherwise, must wait until event occurs \*/

OSTCBCur->OSTCBStat |= OS\_STAT\_SEM; /\* Resource not available, pend on semaphore \*/

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

OSTCBCur->OSTCBDly = timeout; /\* Store pend timeout in 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 \*/

break;

case OS\_STAT\_PEND\_TO:

default:

OS\_EventTaskRemove(OSTCBCur, pevent);

\*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

OS\_EXIT\_CRITICAL();

}

/\*

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

\* ABORT WAITING ON A SEMAPHORE

\*

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

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

\* the semaphore via OSSemPost().

\*

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

\* semaphore.

\*

\* opt determines the type of ABORT performed:

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

\* semaphore

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

\* semaphore

\*

\* 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 semaphore.

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

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

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

\* was aborted.

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

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

\*

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

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

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

\*/

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

INT8U OSSemPendAbort (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\_SEM) { /\* Validate event block type \*/

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

return (0u);

}

OS\_ENTER\_CRITICAL();

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

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 semaphore \*/

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

nbr\_tasks++;

}

break;

case OS\_PEND\_OPT\_NONE:

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

(void)OS\_EventTaskRdy(pevent, (void \*)0, OS\_STAT\_SEM, 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 semaphore \*/

}

#endif

/\*

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

\* POST TO A SEMAPHORE

\*

\* Description: This function signals a semaphore

\*

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

\* semaphore.

\*

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

\* OS\_ERR\_SEM\_OVF If the semaphore count exceeded its limit. In other words, you have

\* signalled the semaphore more often than you waited on it with either

\* OSSemAccept() or OSSemPend().

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

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

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

\*/

INT8U OSSemPost (OS\_EVENT \*pevent)

{

#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);

}

#endif

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

return (OS\_ERR\_EVENT\_TYPE);

}

OS\_ENTER\_CRITICAL();

if (pevent->OSEventGrp != 0u) { /\* See if any task waiting for semaphore \*/

/\* Ready HPT waiting on event \*/

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

OS\_EXIT\_CRITICAL();

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

return (OS\_ERR\_NONE);

}

if (pevent->OSEventCnt < 65535u) { /\* Make sure semaphore will not overflow \*/

pevent->OSEventCnt++; /\* Increment semaphore count to register event \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

OS\_EXIT\_CRITICAL(); /\* Semaphore value has reached its maximum \*/

return (OS\_ERR\_SEM\_OVF);

}

/\*

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

\* QUERY A SEMAPHORE

\*

\* Description: This function obtains information about a semaphore

\*

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

\* semaphore

\*

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

\* semaphore.

\*

\* 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 semaphore.

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

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

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

\*/

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

INT8U OSSemQuery (OS\_EVENT \*pevent,

OS\_SEM\_DATA \*p\_sem\_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\_sem\_data == (OS\_SEM\_DATA \*)0) { /\* Validate 'p\_sem\_data' \*/

return (OS\_ERR\_PDATA\_NULL);

}

#endif

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

return (OS\_ERR\_EVENT\_TYPE);

}

OS\_ENTER\_CRITICAL();

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

psrc = &pevent->OSEventTbl[0];

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

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

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

}

p\_sem\_data->OSCnt = pevent->OSEventCnt; /\* Get semaphore count \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

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

/\*

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

\* SET SEMAPHORE

\*

\* Description: This function sets the semaphore count to the value specified as an argument. Typically,

\* this value would be 0.

\*

\* You would typically use this function when a semaphore is used as a signaling mechanism

\* and, you want to reset the count value.

\*

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

\*

\* cnt is the new value for the semaphore count. You would pass 0 to reset the

\* semaphore count.

\*

\* perr is a pointer to an error code returned by the function as follows:

\*

\* OS\_ERR\_NONE The call was successful and the semaphore value was set.

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

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

\* OS\_ERR\_TASK\_WAITING If tasks are waiting on the semaphore.

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

\*/

#if OS\_SEM\_SET\_EN > 0u

void OSSemSet (OS\_EVENT \*pevent,

INT16U cnt,

INT8U \*perr)

{

#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\_SEM) { /\* Validate event block type \*/

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

return;

}

OS\_ENTER\_CRITICAL();

\*perr = OS\_ERR\_NONE;

if (pevent->OSEventCnt > 0u) { /\* See if semaphore already has a count \*/

pevent->OSEventCnt = cnt; /\* Yes, set it to the new value specified. \*/

} else { /\* No \*/

if (pevent->OSEventGrp == 0u) { /\* See if task(s) waiting? \*/

pevent->OSEventCnt = cnt; /\* No, OK to set the value \*/

} else {

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

}

}

OS\_EXIT\_CRITICAL();

}

#endif

#endif /\* OS\_SEM\_EN \*/