/\*

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

\* uC/OS-II

\* The Real-Time Kernel

\* CORE FUNCTIONS

\*

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

\* All Rights Reserved

\*

\* File : OS\_CORE.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

#define OS\_GLOBALS

#include "ucos\_ii.h"

#endif

/\*

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

\* PRIORITY RESOLUTION TABLE

\*

\* Note: Index into table is bit pattern to resolve highest priority

\* Indexed value corresponds to highest priority bit position (i.e. 0..7)

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

\*/

INT8U const OSUnMapTbl[256] = {

0u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x00 to 0x0F \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x10 to 0x1F \*/

5u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x20 to 0x2F \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x30 to 0x3F \*/

6u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x40 to 0x4F \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x50 to 0x5F \*/

5u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x60 to 0x6F \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x70 to 0x7F \*/

7u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x80 to 0x8F \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x90 to 0x9F \*/

5u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0xA0 to 0xAF \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0xB0 to 0xBF \*/

6u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0xC0 to 0xCF \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0xD0 to 0xDF \*/

5u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0xE0 to 0xEF \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u /\* 0xF0 to 0xFF \*/

};

/\*

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

\* FUNCTION PROTOTYPES

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

\*/

static void OS\_InitEventList(void);

static void OS\_InitMisc(void);

static void OS\_InitRdyList(void);

static void OS\_InitTaskIdle(void);

#if OS\_TASK\_STAT\_EN > 0u

static void OS\_InitTaskStat(void);

#endif

static void OS\_InitTCBList(void);

static void OS\_SchedNew(void);

/\*

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

\* GET THE NAME OF A SEMAPHORE, MUTEX, MAILBOX or QUEUE

\*

\* Description: This function is used to obtain the name assigned to a semaphore, mutex, mailbox or queue.

\*

\* Arguments : pevent is a pointer to the event group. 'pevent' can point either to a semaphore,

\* a mutex, a mailbox or a queue. Where this function is concerned, the actual

\* type is irrelevant.

\*

\* pname is a pointer to a pointer to an ASCII string that will receive the name of the semaphore,

\* mutex, mailbox or queue.

\*

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

\*

\* OS\_ERR\_NONE if the name was copied to 'pname'

\* OS\_ERR\_EVENT\_TYPE if 'pevent' is not pointing to the proper event

\* control block type.

\* OS\_ERR\_PNAME\_NULL You passed a NULL pointer for 'pname'

\* OS\_ERR\_PEVENT\_NULL if you passed a NULL pointer for 'pevent'

\* OS\_ERR\_NAME\_GET\_ISR if you are trying to call this function from an ISR

\*

\* Returns : The length of the string or 0 if the 'pevent' is a NULL pointer.

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

\*/

#if (OS\_EVENT\_EN) && (OS\_EVENT\_NAME\_EN > 0u)

INT8U OSEventNameGet (OS\_EVENT \*pevent,

INT8U \*\*pname,

INT8U \*perr)

{

INT8U len;

#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) { /\* Is 'pevent' a NULL pointer? \*/

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

return (0u);

}

if (pname == (INT8U \*\*)0) { /\* Is 'pname' a NULL pointer? \*/

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

return (0u);

}

#endif

if (OSIntNesting > 0u) { /\* See if trying to call from an ISR \*/

\*perr = OS\_ERR\_NAME\_GET\_ISR;

return (0u);

}

switch (pevent->OSEventType) {

case OS\_EVENT\_TYPE\_SEM:

case OS\_EVENT\_TYPE\_MUTEX:

case OS\_EVENT\_TYPE\_MBOX:

case OS\_EVENT\_TYPE\_Q:

break;

default:

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

return (0u);

}

OS\_ENTER\_CRITICAL();

\*pname = pevent->OSEventName;

len = OS\_StrLen(\*pname);

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

return (len);

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ASSIGN A NAME TO A SEMAPHORE, MUTEX, MAILBOX or QUEUE

\*

\* Description: This function assigns a name to a semaphore, mutex, mailbox or queue.

\*

\* Arguments : pevent is a pointer to the event group. 'pevent' can point either to a semaphore,

\* a mutex, a mailbox or a queue. Where this function is concerned, it doesn't

\* matter the actual type.

\*

\* pname is a pointer to an ASCII string that will be used as the name of the semaphore,

\* mutex, mailbox or queue.

\*

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

\*

\* OS\_ERR\_NONE if the requested task is resumed

\* OS\_ERR\_EVENT\_TYPE if 'pevent' is not pointing to the proper event

\* control block type.

\* OS\_ERR\_PNAME\_NULL You passed a NULL pointer for 'pname'

\* OS\_ERR\_PEVENT\_NULL if you passed a NULL pointer for 'pevent'

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

\*

\* Returns : None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#if (OS\_EVENT\_EN) && (OS\_EVENT\_NAME\_EN > 0u)

void OSEventNameSet (OS\_EVENT \*pevent,

INT8U \*pname,

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) { /\* Is 'pevent' a NULL pointer? \*/

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

return;

}

if (pname == (INT8U \*)0) { /\* Is 'pname' a NULL pointer? \*/

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

return;

}

#endif

if (OSIntNesting > 0u) { /\* See if trying to call from an ISR \*/

\*perr = OS\_ERR\_NAME\_SET\_ISR;

return;

}

switch (pevent->OSEventType) {

case OS\_EVENT\_TYPE\_SEM:

case OS\_EVENT\_TYPE\_MUTEX:

case OS\_EVENT\_TYPE\_MBOX:

case OS\_EVENT\_TYPE\_Q:

break;

default:

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

return;

}

OS\_ENTER\_CRITICAL();

pevent->OSEventName = pname;

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

}

#endif

/\*$PAGE\*/

/\*

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

\* PEND ON MULTIPLE EVENTS

\*

\* Description: This function waits for multiple events. If multiple events are ready at the start of the

\* pend call, then all available events are returned as ready. If the task must pend on the

\* multiple events, then only the first posted or aborted event is returned as ready.

\*

\* Arguments : pevents\_pend is a pointer to a NULL-terminated array of event control blocks to wait for.

\*

\* pevents\_rdy is a pointer to an array to return which event control blocks are available

\* or ready. The size of the array MUST be greater than or equal to the size

\* of the 'pevents\_pend' array, including terminating NULL.

\*

\* pmsgs\_rdy is a pointer to an array to return messages from any available message-type

\* events. The size of the array MUST be greater than or equal to the size of

\* the 'pevents\_pend' array, excluding the terminating NULL. Since NULL

\* messages are valid messages, this array cannot be NULL-terminated. Instead,

\* every available message-type event returns its messages in the 'pmsgs\_rdy'

\* array at the same index as the event is returned in the 'pevents\_rdy' array.

\* All other 'pmsgs\_rdy' array indices are filled with NULL messages.

\*

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

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

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

\* events or, until the resources becomes available (or the events occur).

\*

\* 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 resources

\* or, the events you are waiting for occurred; check the

\* 'pevents\_rdy' array for which events are available.

\* OS\_ERR\_PEND\_ABORT The wait on the events was aborted; check the

\* 'pevents\_rdy' array for which events were aborted.

\* OS\_ERR\_TIMEOUT The events were not received within the specified

\* 'timeout'.

\* OS\_ERR\_PEVENT\_NULL If 'pevents\_pend', 'pevents\_rdy', or 'pmsgs\_rdy' is a

\* NULL pointer.

\* OS\_ERR\_EVENT\_TYPE If you didn't pass a pointer to an array of semaphores,

\* mailboxes, and/or queues.

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

\* would lead to a suspension.

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

\*

\* Returns : > 0 the number of events returned as ready or aborted.

\* == 0 if no events are returned as ready because of timeout or upon error.

\*

\* Notes : 1) a. Validate 'pevents\_pend' array as valid OS\_EVENTs :

\*

\* semaphores, mailboxes, queues

\*

\* b. Return ALL available events and messages, if any

\*

\* c. Add current task priority as pending to each events's wait list

\* Performed in OS\_EventTaskWaitMulti()

\*

\* d. Wait on any of multiple events

\*

\* e. Remove current task priority as pending from each events's wait list

\* Performed in OS\_EventTaskRdy(), if events posted or aborted

\*

\* f. Return any event posted or aborted, if any

\* else

\* Return timeout

\*

\* 2) 'pevents\_rdy' initialized to NULL PRIOR to all other validation or function handling in

\* case of any error(s).

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

\*/

/\*$PAGE\*/

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

INT16U OSEventPendMulti (OS\_EVENT \*\*pevents\_pend,

OS\_EVENT \*\*pevents\_rdy,

void \*\*pmsgs\_rdy,

INT32U timeout,

INT8U \*perr)

{

OS\_EVENT \*\*pevents;

OS\_EVENT \*pevent;

#if ((OS\_Q\_EN > 0u) && (OS\_MAX\_QS > 0u))

OS\_Q \*pq;

#endif

BOOLEAN events\_rdy;

INT16U events\_rdy\_nbr;

INT8U events\_stat;

#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 (pevents\_pend == (OS\_EVENT \*\*)0) { /\* Validate 'pevents\_pend' \*/

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

return (0u);

}

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

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

return (0u);

}

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

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

return (0u);

}

if (pmsgs\_rdy == (void \*\*)0) { /\* Validate 'pmsgs\_rdy' \*/

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

return (0u);

}

#endif

\*pevents\_rdy = (OS\_EVENT \*)0; /\* Init array to NULL in case of errors \*/

pevents = pevents\_pend;

pevent = \*pevents;

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

switch (pevent->OSEventType) { /\* Validate event block types \*/

#if (OS\_SEM\_EN > 0u)

case OS\_EVENT\_TYPE\_SEM:

break;

#endif

#if (OS\_MBOX\_EN > 0u)

case OS\_EVENT\_TYPE\_MBOX:

break;

#endif

#if ((OS\_Q\_EN > 0u) && (OS\_MAX\_QS > 0u))

case OS\_EVENT\_TYPE\_Q:

break;

#endif

case OS\_EVENT\_TYPE\_MUTEX:

case OS\_EVENT\_TYPE\_FLAG:

default:

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

return (0u);

}

pevents++;

pevent = \*pevents;

}

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

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

return (0u);

}

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

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

return (0u);

}

OS\_ENTER\_CRITICAL();

events\_rdy = OS\_FALSE;

events\_rdy\_nbr = 0u;

events\_stat = OS\_STAT\_RDY;

pevents = pevents\_pend;

pevent = \*pevents;

while (pevent != (OS\_EVENT \*)0) { /\* See if any events already available \*/

switch (pevent->OSEventType) {

#if (OS\_SEM\_EN > 0u)

case OS\_EVENT\_TYPE\_SEM:

if (pevent->OSEventCnt > 0u) { /\* If semaphore count > 0, resource available; \*/

pevent->OSEventCnt--; /\* ... decrement semaphore, ... \*/

\*pevents\_rdy++ = pevent; /\* ... and return available semaphore event \*/

events\_rdy = OS\_TRUE;

\*pmsgs\_rdy++ = (void \*)0; /\* NO message returned for semaphores \*/

events\_rdy\_nbr++;

} else {

events\_stat |= OS\_STAT\_SEM; /\* Configure multi-pend for semaphore events \*/

}

break;

#endif

#if (OS\_MBOX\_EN > 0u)

case OS\_EVENT\_TYPE\_MBOX:

if (pevent->OSEventPtr != (void \*)0) { /\* If mailbox NOT empty; ... \*/

/\* ... return available message, ... \*/

\*pmsgs\_rdy++ = (void \*)pevent->OSEventPtr;

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

\*pevents\_rdy++ = pevent; /\* ... and return available mailbox event \*/

events\_rdy = OS\_TRUE;

events\_rdy\_nbr++;

} else {

events\_stat |= OS\_STAT\_MBOX; /\* Configure multi-pend for mailbox events \*/

}

break;

#endif

#if ((OS\_Q\_EN > 0u) && (OS\_MAX\_QS > 0u))

case OS\_EVENT\_TYPE\_Q:

pq = (OS\_Q \*)pevent->OSEventPtr;

if (pq->OSQEntries > 0u) { /\* If queue NOT empty; ... \*/

/\* ... return available message, ... \*/

\*pmsgs\_rdy++ = (void \*)\*pq->OSQOut++;

if (pq->OSQOut == pq->OSQEnd) { /\* If OUT ptr at queue end, ... \*/

pq->OSQOut = pq->OSQStart; /\* ... wrap to queue start \*/

}

pq->OSQEntries--; /\* Update number of queue entries \*/

\*pevents\_rdy++ = pevent; /\* ... and return available queue event \*/

events\_rdy = OS\_TRUE;

events\_rdy\_nbr++;

} else {

events\_stat |= OS\_STAT\_Q; /\* Configure multi-pend for queue events \*/

}

break;

#endif

case OS\_EVENT\_TYPE\_MUTEX:

case OS\_EVENT\_TYPE\_FLAG:

default:

OS\_EXIT\_CRITICAL();

\*pevents\_rdy = (OS\_EVENT \*)0; /\* NULL terminate return event array \*/

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

return (events\_rdy\_nbr);

}

pevents++;

pevent = \*pevents;

}

if ( events\_rdy == OS\_TRUE) { /\* Return any events already available \*/

\*pevents\_rdy = (OS\_EVENT \*)0; /\* NULL terminate return event array \*/

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

return (events\_rdy\_nbr);

}

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

OSTCBCur->OSTCBStat |= events\_stat | /\* Resource not available, ... \*/

OS\_STAT\_MULTI; /\* ... pend on multiple events \*/

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

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

OS\_EventTaskWaitMulti(pevents\_pend); /\* Suspend task until events or timeout occurs \*/

OS\_EXIT\_CRITICAL();

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

OS\_ENTER\_CRITICAL();

switch (OSTCBCur->OSTCBStatPend) { /\* Handle event posted, aborted, or timed-out \*/

case OS\_STAT\_PEND\_OK:

case OS\_STAT\_PEND\_ABORT:

pevent = OSTCBCur->OSTCBEventPtr;

if (pevent != (OS\_EVENT \*)0) { /\* If task event ptr != NULL, ... \*/

\*pevents\_rdy++ = pevent; /\* ... return available event ... \*/

\*pevents\_rdy = (OS\_EVENT \*)0; /\* ... & NULL terminate return event array \*/

events\_rdy\_nbr++;

} else { /\* Else NO event available, handle as timeout \*/

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

OS\_EventTaskRemoveMulti(OSTCBCur, pevents\_pend);

}

break;

case OS\_STAT\_PEND\_TO: /\* If events timed out, ... \*/

default: /\* ... remove task from events' wait lists \*/

OS\_EventTaskRemoveMulti(OSTCBCur, pevents\_pend);

break;

}

switch (OSTCBCur->OSTCBStatPend) {

case OS\_STAT\_PEND\_OK:

switch (pevent->OSEventType) { /\* Return event's message \*/

#if (OS\_SEM\_EN > 0u)

case OS\_EVENT\_TYPE\_SEM:

\*pmsgs\_rdy++ = (void \*)0; /\* NO message returned for semaphores \*/

break;

#endif

#if ((OS\_MBOX\_EN > 0u) || \

((OS\_Q\_EN > 0u) && (OS\_MAX\_QS > 0u)))

case OS\_EVENT\_TYPE\_MBOX:

case OS\_EVENT\_TYPE\_Q:

\*pmsgs\_rdy++ = (void \*)OSTCBCur->OSTCBMsg; /\* Return received message \*/

break;

#endif

case OS\_EVENT\_TYPE\_MUTEX:

case OS\_EVENT\_TYPE\_FLAG:

default:

OS\_EXIT\_CRITICAL();

\*pevents\_rdy = (OS\_EVENT \*)0; /\* NULL terminate return event array \*/

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

return (events\_rdy\_nbr);

}

\*perr = OS\_ERR\_NONE;

break;

case OS\_STAT\_PEND\_ABORT:

\*pmsgs\_rdy++ = (void \*)0; /\* NO message returned for abort \*/

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

break;

case OS\_STAT\_PEND\_TO:

default:

\*pmsgs\_rdy++ = (void \*)0; /\* NO message returned for timeout \*/

\*perr = OS\_ERR\_TIMEOUT; /\* Indicate that events timed out \*/

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 \*/

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

#if ((OS\_MBOX\_EN > 0u) || \

((OS\_Q\_EN > 0u) && (OS\_MAX\_QS > 0u)))

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

#endif

OS\_EXIT\_CRITICAL();

return (events\_rdy\_nbr);

}

#endif

/\*

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

\* INITIALIZATION

\*

\* Description: This function is used to initialize the internals of uC/OS-II and MUST be called prior to

\* creating any uC/OS-II object and, prior to calling OSStart().

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

void OSInit (void)

{

OSInitHookBegin(); /\* Call port specific initialization code \*/

OS\_InitMisc(); /\* Initialize miscellaneous variables \*/

OS\_InitRdyList(); /\* Initialize the Ready List \*/

OS\_InitTCBList(); /\* Initialize the free list of OS\_TCBs \*/

OS\_InitEventList(); /\* Initialize the free list of OS\_EVENTs \*/

#if (OS\_FLAG\_EN > 0u) && (OS\_MAX\_FLAGS > 0u)

OS\_FlagInit(); /\* Initialize the event flag structures \*/

#endif

#if (OS\_MEM\_EN > 0u) && (OS\_MAX\_MEM\_PART > 0u)

OS\_MemInit(); /\* Initialize the memory manager \*/

#endif

#if (OS\_Q\_EN > 0u) && (OS\_MAX\_QS > 0u)

OS\_QInit(); /\* Initialize the message queue structures \*/

#endif

OS\_InitTaskIdle(); /\* Create the Idle Task \*/

#if OS\_TASK\_STAT\_EN > 0u

OS\_InitTaskStat(); /\* Create the Statistic Task \*/

#endif

#if OS\_TMR\_EN > 0u

OSTmr\_Init(); /\* Initialize the Timer Manager \*/

#endif

OSInitHookEnd(); /\* Call port specific init. code \*/

#if OS\_DEBUG\_EN > 0u

OSDebugInit();

#endif

}

/\*

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

\* ENTER ISR

\*

\* Description: This function is used to notify uC/OS-II that you are about to service an interrupt

\* service routine (ISR). This allows uC/OS-II to keep track of interrupt nesting and thus

\* only perform rescheduling at the last nested ISR.

\*

\* Arguments : none

\*

\* Returns : none

\*

\* Notes : 1) This function should be called ith interrupts already disabled

\* 2) Your ISR can directly increment OSIntNesting without calling this function because

\* OSIntNesting has been declared 'global'.

\* 3) You MUST still call OSIntExit() even though you increment OSIntNesting directly.

\* 4) You MUST invoke OSIntEnter() and OSIntExit() in pair. In other words, for every call

\* to OSIntEnter() at the beginning of the ISR you MUST have a call to OSIntExit() at the

\* end of the ISR.

\* 5) You are allowed to nest interrupts up to 255 levels deep.

\* 6) I removed the OS\_ENTER\_CRITICAL() and OS\_EXIT\_CRITICAL() around the increment because

\* OSIntEnter() is always called with interrupts disabled.

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

\*/

void OSIntEnter (void)

{

if (OSRunning == OS\_TRUE) {

if (OSIntNesting < 255u) {

OSIntNesting++; /\* Increment ISR nesting level \*/

}

}

}

/\*

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

\* EXIT ISR

\*

\* Description: This function is used to notify uC/OS-II that you have completed serviving an ISR. When

\* the last nested ISR has completed, uC/OS-II will call the scheduler to determine whether

\* a new, high-priority task, is ready to run.

\*

\* Arguments : none

\*

\* Returns : none

\*

\* Notes : 1) You MUST invoke OSIntEnter() and OSIntExit() in pair. In other words, for every call

\* to OSIntEnter() at the beginning of the ISR you MUST have a call to OSIntExit() at the

\* end of the ISR.

\* 2) Rescheduling is prevented when the scheduler is locked (see OS\_SchedLock())

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

\*/

void OSIntExit (void)

{

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

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

#endif

if (OSRunning == OS\_TRUE) {

OS\_ENTER\_CRITICAL();

if (OSIntNesting > 0u) { /\* Prevent OSIntNesting from wrapping \*/

OSIntNesting--;

}

if (OSIntNesting == 0u) { /\* Reschedule only if all ISRs complete ... \*/

if (OSLockNesting == 0u) { /\* ... and not locked. \*/

OS\_SchedNew();

OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];

if (OSPrioHighRdy != OSPrioCur) { /\* No Ctx Sw if current task is highest rdy \*/

#if OS\_TASK\_PROFILE\_EN > 0u

OSTCBHighRdy->OSTCBCtxSwCtr++; /\* Inc. # of context switches to this task \*/

#endif

OSCtxSwCtr++; /\* Keep track of the number of ctx switches \*/

OSIntCtxSw(); /\* Perform interrupt level ctx switch \*/

}

}

}

OS\_EXIT\_CRITICAL();

}

}

/\*

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

\* INDICATE THAT IT'S NO LONGER SAFE TO CREATE OBJECTS

\*

\* Description: This function is called by the application code to indicate that all initialization has

\* been completed and that kernel objects are no longer allowed to be created.

\*

\* Arguments : none

\*

\* Returns : none

\*

\* Note(s) : 1) You should call this function when you no longer want to allow application code to

\* create kernel objects.

\* 2) You need to define the macro 'OS\_SAFETY\_CRITICAL\_IEC61508'

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

\*/

#ifdef OS\_SAFETY\_CRITICAL\_IEC61508

void OSSafetyCriticalStart (void)

{

OSSafetyCriticalStartFlag = OS\_TRUE;

}

#endif

/\*

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

\* PREVENT SCHEDULING

\*

\* Description: This function is used to prevent rescheduling to take place. This allows your application

\* to prevent context switches until you are ready to permit context switching.

\*

\* Arguments : none

\*

\* Returns : none

\*

\* Notes : 1) You MUST invoke OSSchedLock() and OSSchedUnlock() in pair. In other words, for every

\* call to OSSchedLock() you MUST have a call to OSSchedUnlock().

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

\*/

#if OS\_SCHED\_LOCK\_EN > 0u

void OSSchedLock (void)

{

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

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

#endif

if (OSRunning == OS\_TRUE) { /\* Make sure multitasking is running \*/

OS\_ENTER\_CRITICAL();

if (OSIntNesting == 0u) { /\* Can't call from an ISR \*/

if (OSLockNesting < 255u) { /\* Prevent OSLockNesting from wrapping back to 0 \*/

OSLockNesting++; /\* Increment lock nesting level \*/

}

}

OS\_EXIT\_CRITICAL();

}

}

#endif

/\*

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

\* ENABLE SCHEDULING

\*

\* Description: This function is used to re-allow rescheduling.

\*

\* Arguments : none

\*

\* Returns : none

\*

\* Notes : 1) You MUST invoke OSSchedLock() and OSSchedUnlock() in pair. In other words, for every

\* call to OSSchedLock() you MUST have a call to OSSchedUnlock().

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

\*/

#if OS\_SCHED\_LOCK\_EN > 0u

void OSSchedUnlock (void)

{

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

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

#endif

if (OSRunning == OS\_TRUE) { /\* Make sure multitasking is running \*/

OS\_ENTER\_CRITICAL();

if (OSLockNesting > 0u) { /\* Do not decrement if already 0 \*/

OSLockNesting--; /\* Decrement lock nesting level \*/

if (OSLockNesting == 0u) { /\* See if scheduler is enabled and ... \*/

if (OSIntNesting == 0u) { /\* ... not in an ISR \*/

OS\_EXIT\_CRITICAL();

OS\_Sched(); /\* See if a HPT is ready \*/

} else {

OS\_EXIT\_CRITICAL();

}

} else {

OS\_EXIT\_CRITICAL();

}

} else {

OS\_EXIT\_CRITICAL();

}

}

}

#endif

/\*

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

\* START MULTITASKING

\*

\* Description: This function is used to start the multitasking process which lets uC/OS-II manages the

\* task that you have created. Before you can call OSStart(), you MUST have called OSInit()

\* and you MUST have created at least one task.

\*

\* Arguments : none

\*

\* Returns : none

\*

\* Note : OSStartHighRdy() MUST:

\* a) Call OSTaskSwHook() then,

\* b) Set OSRunning to OS\_TRUE.

\* c) Load the context of the task pointed to by OSTCBHighRdy.

\* d\_ Execute the task.

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

\*/

void OSStart (void)

{

if (OSRunning == OS\_FALSE) {

OS\_SchedNew(); /\* Find highest priority's task priority number \*/

OSPrioCur = OSPrioHighRdy;

OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy]; /\* Point to highest priority task ready to run \*/

OSTCBCur = OSTCBHighRdy;

OSStartHighRdy(); /\* Execute target specific code to start task \*/

}

}

/\*

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

\* STATISTICS INITIALIZATION

\*

\* Description: This function is called by your application to establish CPU usage by first determining

\* how high a 32-bit counter would count to in 1 second if no other tasks were to execute

\* during that time. CPU usage is then determined by a low priority task which keeps track

\* of this 32-bit counter every second but this time, with other tasks running. CPU usage is

\* determined by:

\*

\* OSIdleCtr

\* CPU Usage (%) = 100 \* (1 - ------------)

\* OSIdleCtrMax

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

#if OS\_TASK\_STAT\_EN > 0u

void OSStatInit (void)

{

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

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

#endif

OSTimeDly(2u); /\* Synchronize with clock tick \*/

OS\_ENTER\_CRITICAL();

OSIdleCtr = 0uL; /\* Clear idle counter \*/

OS\_EXIT\_CRITICAL();

OSTimeDly(OS\_TICKS\_PER\_SEC / 10u); /\* Determine MAX. idle counter value for 1/10 second \*/

OS\_ENTER\_CRITICAL();

OSIdleCtrMax = OSIdleCtr; /\* Store maximum idle counter count in 1/10 second \*/

OSStatRdy = OS\_TRUE;

OS\_EXIT\_CRITICAL();

}

#endif

/\*

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

\* PROCESS SYSTEM TICK

\*

\* Description: This function is used to signal to uC/OS-II the occurrence of a 'system tick' (also known

\* as a 'clock tick'). This function should be called by the ticker ISR but, can also be

\* called by a high priority task.

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

void OSTimeTick (void)

{

OS\_TCB \*ptcb;

#if OS\_TICK\_STEP\_EN > 0u

BOOLEAN step;

#endif

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

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

#endif

#if OS\_TIME\_TICK\_HOOK\_EN > 0u

OSTimeTickHook(); /\* Call user definable hook \*/

#endif

#if OS\_TIME\_GET\_SET\_EN > 0u

OS\_ENTER\_CRITICAL(); /\* Update the 32-bit tick counter \*/

OSTime++;

OS\_EXIT\_CRITICAL();

#endif

if (OSRunning == OS\_TRUE) {

#if OS\_TICK\_STEP\_EN > 0u

switch (OSTickStepState) { /\* Determine whether we need to process a tick \*/

case OS\_TICK\_STEP\_DIS: /\* Yes, stepping is disabled \*/

step = OS\_TRUE;

break;

case OS\_TICK\_STEP\_WAIT: /\* No, waiting for uC/OS-View to set ... \*/

step = OS\_FALSE; /\* .. OSTickStepState to OS\_TICK\_STEP\_ONCE \*/

break;

case OS\_TICK\_STEP\_ONCE: /\* Yes, process tick once and wait for next ... \*/

step = OS\_TRUE; /\* ... step command from uC/OS-View \*/

OSTickStepState = OS\_TICK\_STEP\_WAIT;

break;

default: /\* Invalid case, correct situation \*/

step = OS\_TRUE;

OSTickStepState = OS\_TICK\_STEP\_DIS;

break;

}

if (step == OS\_FALSE) { /\* Return if waiting for step command \*/

return;

}

#endif

ptcb = OSTCBList; /\* Point at first TCB in TCB list \*/

while (ptcb->OSTCBPrio != OS\_TASK\_IDLE\_PRIO) { /\* Go through all TCBs in TCB list \*/

OS\_ENTER\_CRITICAL();

if (ptcb->OSTCBDly != 0u) { /\* No, Delayed or waiting for event with TO \*/

ptcb->OSTCBDly--; /\* Decrement nbr of ticks to end of delay \*/

if (ptcb->OSTCBDly == 0u) { /\* Check for timeout \*/

if ((ptcb->OSTCBStat & OS\_STAT\_PEND\_ANY) != OS\_STAT\_RDY) {

ptcb->OSTCBStat &= (INT8U)~(INT8U)OS\_STAT\_PEND\_ANY; /\* Yes, Clear status flag \*/

ptcb->OSTCBStatPend = OS\_STAT\_PEND\_TO; /\* Indicate PEND timeout \*/

} else {

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

}

if ((ptcb->OSTCBStat & OS\_STAT\_SUSPEND) == OS\_STAT\_RDY) { /\* Is task suspended? \*/

OSRdyGrp |= ptcb->OSTCBBitY; /\* No, Make ready \*/

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

}

}

}

ptcb = ptcb->OSTCBNext; /\* Point at next TCB in TCB list \*/

OS\_EXIT\_CRITICAL();

}

}

}

/\*

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

\* GET VERSION

\*

\* Description: This function is used to return the version number of uC/OS-II. The returned value

\* corresponds to uC/OS-II's version number multiplied by 100. In other words, version 2.00

\* would be returned as 200.

\*

\* Arguments : none

\*

\* Returns : the version number of uC/OS-II multiplied by 100.

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

\*/

INT16U OSVersion (void)

{

return (OS\_VERSION);

}

/\*

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

\* DUMMY FUNCTION

\*

\* Description: This function doesn't do anything. It is called by OSTaskDel().

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

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

void OS\_Dummy (void)

{

}

#endif

/\*

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

\* MAKE TASK READY TO RUN BASED ON EVENT OCCURING

\*

\* Description: This function is called by other uC/OS-II services and is used to ready a task that was

\* waiting for an event to occur.

\*

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

\*

\* pmsg is a pointer to a message. This pointer is used by message oriented services

\* such as MAILBOXEs and QUEUEs. The pointer is not used when called by other

\* service functions.

\*

\* msk is a mask that is used to clear the status byte of the TCB. For example,

\* OSSemPost() will pass OS\_STAT\_SEM, OSMboxPost() will pass OS\_STAT\_MBOX etc.

\*

\* pend\_stat is used to indicate the readied task's pending status:

\*

\* OS\_STAT\_PEND\_OK Task ready due to a post (or delete), not a timeout or

\* an abort.

\* OS\_STAT\_PEND\_ABORT Task ready due to an abort.

\*

\* Returns : none

\*

\* Note : This function is INTERNAL to uC/OS-II and your application should not call it.

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

\*/

#if (OS\_EVENT\_EN)

INT8U OS\_EventTaskRdy (OS\_EVENT \*pevent,

void \*pmsg,

INT8U msk,

INT8U pend\_stat)

{

OS\_TCB \*ptcb;

INT8U y;

INT8U x;

INT8U prio;

#if OS\_LOWEST\_PRIO > 63u

OS\_PRIO \*ptbl;

#endif

#if OS\_LOWEST\_PRIO <= 63u

y = OSUnMapTbl[pevent->OSEventGrp]; /\* Find HPT waiting for message \*/

x = OSUnMapTbl[pevent->OSEventTbl[y]];

prio = (INT8U)((y << 3u) + x); /\* Find priority of task getting the msg \*/

#else

if ((pevent->OSEventGrp & 0xFFu) != 0u) { /\* Find HPT waiting for message \*/

y = OSUnMapTbl[ pevent->OSEventGrp & 0xFFu];

} else {

y = OSUnMapTbl[(OS\_PRIO)(pevent->OSEventGrp >> 8u) & 0xFFu] + 8u;

}

ptbl = &pevent->OSEventTbl[y];

if ((\*ptbl & 0xFFu) != 0u) {

x = OSUnMapTbl[\*ptbl & 0xFFu];

} else {

x = OSUnMapTbl[(OS\_PRIO)(\*ptbl >> 8u) & 0xFFu] + 8u;

}

prio = (INT8U)((y << 4u) + x); /\* Find priority of task getting the msg \*/

#endif

ptcb = OSTCBPrioTbl[prio]; /\* Point to this task's OS\_TCB \*/

ptcb->OSTCBDly = 0u; /\* Prevent OSTimeTick() from readying task \*/

#if ((OS\_Q\_EN > 0u) && (OS\_MAX\_QS > 0u)) || (OS\_MBOX\_EN > 0u)

ptcb->OSTCBMsg = pmsg; /\* Send message directly to waiting task \*/

#else

pmsg = pmsg; /\* Prevent compiler warning if not used \*/

#endif

ptcb->OSTCBStat &= (INT8U)~msk; /\* Clear bit associated with event type \*/

ptcb->OSTCBStatPend = pend\_stat; /\* Set pend status of post or abort \*/

/\* See if task is ready (could be susp'd) \*/

if ((ptcb->OSTCBStat & OS\_STAT\_SUSPEND) == OS\_STAT\_RDY) {

OSRdyGrp |= ptcb->OSTCBBitY; /\* Put task in the ready to run list \*/

OSRdyTbl[y] |= ptcb->OSTCBBitX;

}

OS\_EventTaskRemove(ptcb, pevent); /\* Remove this task from event wait list \*/

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

if (ptcb->OSTCBEventMultiPtr != (OS\_EVENT \*\*)0) { /\* Remove this task from events' wait lists \*/

OS\_EventTaskRemoveMulti(ptcb, ptcb->OSTCBEventMultiPtr);

ptcb->OSTCBEventPtr = (OS\_EVENT \*)pevent;/\* Return event as first multi-pend event ready\*/

}

#endif

return (prio);

}

#endif

/\*

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

\* MAKE TASK WAIT FOR EVENT TO OCCUR

\*

\* Description: This function is called by other uC/OS-II services to suspend a task because an event has

\* not occurred.

\*

\* Arguments : pevent is a pointer to the event control block for which the task will be waiting for.

\*

\* Returns : none

\*

\* Note : This function is INTERNAL to uC/OS-II and your application should not call it.

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

\*/

#if (OS\_EVENT\_EN)

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

{

INT8U y;

OSTCBCur->OSTCBEventPtr = pevent; /\* Store ptr to ECB in TCB \*/

pevent->OSEventTbl[OSTCBCur->OSTCBY] |= OSTCBCur->OSTCBBitX; /\* Put task in waiting list \*/

pevent->OSEventGrp |= OSTCBCur->OSTCBBitY;

y = OSTCBCur->OSTCBY; /\* Task no longer ready \*/

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

if (OSRdyTbl[y] == 0u) { /\* Clear event grp bit if this was only task pending \*/

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

}

}

#endif

/\*

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

\* MAKE TASK WAIT FOR ANY OF MULTIPLE EVENTS TO OCCUR

\*

\* Description: This function is called by other uC/OS-II services to suspend a task because any one of

\* multiple events has not occurred.

\*

\* Arguments : pevents\_wait is a pointer to an array of event control blocks, NULL-terminated, for

\* which the task will be waiting for.

\*

\* Returns : none.

\*

\* Note : This function is INTERNAL to uC/OS-II and your application should not call it.

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

\*/

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

void OS\_EventTaskWaitMulti (OS\_EVENT \*\*pevents\_wait)

{

OS\_EVENT \*\*pevents;

OS\_EVENT \*pevent;

INT8U y;

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

OSTCBCur->OSTCBEventMultiPtr = (OS\_EVENT \*\*)pevents\_wait; /\* Store ptr to ECBs in TCB \*/

pevents = pevents\_wait;

pevent = \*pevents;

while (pevent != (OS\_EVENT \*)0) { /\* Put task in waiting lists \*/

pevent->OSEventTbl[OSTCBCur->OSTCBY] |= OSTCBCur->OSTCBBitX;

pevent->OSEventGrp |= OSTCBCur->OSTCBBitY;

pevents++;

pevent = \*pevents;

}

y = OSTCBCur->OSTCBY; /\* Task no longer ready \*/

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

if (OSRdyTbl[y] == 0u) { /\* Clear event grp bit if this was only task pending \*/

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

}

}

#endif

/\*

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

\* REMOVE TASK FROM EVENT WAIT LIST

\*

\* Description: Remove a task from an event's wait list.

\*

\* Arguments : ptcb is a pointer to the task to remove.

\*

\* pevent is a pointer to the event control block.

\*

\* Returns : none

\*

\* Note : This function is INTERNAL to uC/OS-II and your application should not call it.

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

\*/

#if (OS\_EVENT\_EN)

void OS\_EventTaskRemove (OS\_TCB \*ptcb,

OS\_EVENT \*pevent)

{

INT8U y;

y = ptcb->OSTCBY;

pevent->OSEventTbl[y] &= (OS\_PRIO)~ptcb->OSTCBBitX; /\* Remove task from wait list \*/

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

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

}

}

#endif

/\*

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

\* REMOVE TASK FROM MULTIPLE EVENTS WAIT LISTS

\*

\* Description: Remove a task from multiple events' wait lists.

\*

\* Arguments : ptcb is a pointer to the task to remove.

\*

\* pevents\_multi is a pointer to the array of event control blocks, NULL-terminated.

\*

\* Returns : none

\*

\* Note : This function is INTERNAL to uC/OS-II and your application should not call it.

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

\*/

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

void OS\_EventTaskRemoveMulti (OS\_TCB \*ptcb,

OS\_EVENT \*\*pevents\_multi)

{

OS\_EVENT \*\*pevents;

OS\_EVENT \*pevent;

INT8U y;

OS\_PRIO bity;

OS\_PRIO bitx;

y = ptcb->OSTCBY;

bity = ptcb->OSTCBBitY;

bitx = ptcb->OSTCBBitX;

pevents = pevents\_multi;

pevent = \*pevents;

while (pevent != (OS\_EVENT \*)0) { /\* Remove task from all events' wait lists \*/

pevent->OSEventTbl[y] &= (OS\_PRIO)~bitx;

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

pevent->OSEventGrp &= (OS\_PRIO)~bity;

}

pevents++;

pevent = \*pevents;

}

}

#endif

/\*

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

\* INITIALIZE EVENT CONTROL BLOCK'S WAIT LIST

\*

\* Description: This function is called by other uC/OS-II services to initialize the event wait list.

\*

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

\*

\* Returns : none

\*

\* Note : This function is INTERNAL to uC/OS-II and your application should not call it.

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

\*/

#if (OS\_EVENT\_EN)

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

{

INT8U i;

pevent->OSEventGrp = 0u; /\* No task waiting on event \*/

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

pevent->OSEventTbl[i] = 0u;

}

}

#endif

/\*

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

\* INITIALIZATION

\* INITIALIZE THE FREE LIST OF EVENT CONTROL BLOCKS

\*

\* Description: This function is called by OSInit() to initialize the free list of event control blocks.

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

static void OS\_InitEventList (void)

{

#if (OS\_EVENT\_EN) && (OS\_MAX\_EVENTS > 0u)

#if (OS\_MAX\_EVENTS > 1u)

INT16U ix;

INT16U ix\_next;

OS\_EVENT \*pevent1;

OS\_EVENT \*pevent2;

OS\_MemClr((INT8U \*)&OSEventTbl[0], sizeof(OSEventTbl)); /\* Clear the event table \*/

for (ix = 0u; ix < (OS\_MAX\_EVENTS - 1u); ix++) { /\* Init. list of free EVENT control blocks \*/

ix\_next = ix + 1u;

pevent1 = &OSEventTbl[ix];

pevent2 = &OSEventTbl[ix\_next];

pevent1->OSEventType = OS\_EVENT\_TYPE\_UNUSED;

pevent1->OSEventPtr = pevent2;

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

pevent1->OSEventName = (INT8U \*)(void \*)"?"; /\* Unknown name \*/

#endif

}

pevent1 = &OSEventTbl[ix];

pevent1->OSEventType = OS\_EVENT\_TYPE\_UNUSED;

pevent1->OSEventPtr = (OS\_EVENT \*)0;

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

pevent1->OSEventName = (INT8U \*)(void \*)"?"; /\* Unknown name \*/

#endif

OSEventFreeList = &OSEventTbl[0];

#else

OSEventFreeList = &OSEventTbl[0]; /\* Only have ONE event control block \*/

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

OSEventFreeList->OSEventPtr = (OS\_EVENT \*)0;

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

OSEventFreeList->OSEventName = (INT8U \*)"?"; /\* Unknown name \*/

#endif

#endif

#endif

}

/\*

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

\* INITIALIZATION

\* INITIALIZE MISCELLANEOUS VARIABLES

\*

\* Description: This function is called by OSInit() to initialize miscellaneous variables.

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

static void OS\_InitMisc (void)

{

#if OS\_TIME\_GET\_SET\_EN > 0u

OSTime = 0uL; /\* Clear the 32-bit system clock \*/

#endif

OSIntNesting = 0u; /\* Clear the interrupt nesting counter \*/

OSLockNesting = 0u; /\* Clear the scheduling lock counter \*/

OSTaskCtr = 0u; /\* Clear the number of tasks \*/

OSRunning = OS\_FALSE; /\* Indicate that multitasking not started \*/

OSCtxSwCtr = 0u; /\* Clear the context switch counter \*/

OSIdleCtr = 0uL; /\* Clear the 32-bit idle counter \*/

#if OS\_TASK\_STAT\_EN > 0u

OSIdleCtrRun = 0uL;

OSIdleCtrMax = 0uL;

OSStatRdy = OS\_FALSE; /\* Statistic task is not ready \*/

#endif

#ifdef OS\_SAFETY\_CRITICAL\_IEC61508

OSSafetyCriticalStartFlag = OS\_FALSE; /\* Still allow creation of objects \*/

#endif

}

/\*

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

\* INITIALIZATION

\* INITIALIZE THE READY LIST

\*

\* Description: This function is called by OSInit() to initialize the Ready List.

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

static void OS\_InitRdyList (void)

{

INT8U i;

OSRdyGrp = 0u; /\* Clear the ready list \*/

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

OSRdyTbl[i] = 0u;

}

OSPrioCur = 0u;

OSPrioHighRdy = 0u;

OSTCBHighRdy = (OS\_TCB \*)0;

OSTCBCur = (OS\_TCB \*)0;

}

/\*

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

\* INITIALIZATION

\* CREATING THE IDLE TASK

\*

\* Description: This function creates the Idle Task.

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

static void OS\_InitTaskIdle (void)

{

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

INT8U err;

#endif

#if OS\_TASK\_CREATE\_EXT\_EN > 0u

#if OS\_STK\_GROWTH == 1u

(void)OSTaskCreateExt(OS\_TaskIdle,

(void \*)0, /\* No arguments passed to OS\_TaskIdle() \*/

&OSTaskIdleStk[OS\_TASK\_IDLE\_STK\_SIZE - 1u],/\* Set Top-Of-Stack \*/

OS\_TASK\_IDLE\_PRIO, /\* Lowest priority level \*/

OS\_TASK\_IDLE\_ID,

&OSTaskIdleStk[0], /\* Set Bottom-Of-Stack \*/

OS\_TASK\_IDLE\_STK\_SIZE,

(void \*)0, /\* No TCB extension \*/

OS\_TASK\_OPT\_STK\_CHK | OS\_TASK\_OPT\_STK\_CLR);/\* Enable stack checking + clear stack \*/

#else

(void)OSTaskCreateExt(OS\_TaskIdle,

(void \*)0, /\* No arguments passed to OS\_TaskIdle() \*/

&OSTaskIdleStk[0], /\* Set Top-Of-Stack \*/

OS\_TASK\_IDLE\_PRIO, /\* Lowest priority level \*/

OS\_TASK\_IDLE\_ID,

&OSTaskIdleStk[OS\_TASK\_IDLE\_STK\_SIZE - 1u],/\* Set Bottom-Of-Stack \*/

OS\_TASK\_IDLE\_STK\_SIZE,

(void \*)0, /\* No TCB extension \*/

OS\_TASK\_OPT\_STK\_CHK | OS\_TASK\_OPT\_STK\_CLR);/\* Enable stack checking + clear stack \*/

#endif

#else

#if OS\_STK\_GROWTH == 1u

(void)OSTaskCreate(OS\_TaskIdle,

(void \*)0,

&OSTaskIdleStk[OS\_TASK\_IDLE\_STK\_SIZE - 1u],

OS\_TASK\_IDLE\_PRIO);

#else

(void)OSTaskCreate(OS\_TaskIdle,

(void \*)0,

&OSTaskIdleStk[0],

OS\_TASK\_IDLE\_PRIO);

#endif

#endif

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

OSTaskNameSet(OS\_TASK\_IDLE\_PRIO, (INT8U \*)(void \*)"uC/OS-II Idle", &err);

#endif

}

/\*

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

\* INITIALIZATION

\* CREATING THE STATISTIC TASK

\*

\* Description: This function creates the Statistic Task.

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

#if OS\_TASK\_STAT\_EN > 0u

static void OS\_InitTaskStat (void)

{

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

INT8U err;

#endif

#if OS\_TASK\_CREATE\_EXT\_EN > 0u

#if OS\_STK\_GROWTH == 1u

(void)OSTaskCreateExt(OS\_TaskStat,

(void \*)0, /\* No args passed to OS\_TaskStat()\*/

&OSTaskStatStk[OS\_TASK\_STAT\_STK\_SIZE - 1u], /\* Set Top-Of-Stack \*/

OS\_TASK\_STAT\_PRIO, /\* One higher than the idle task \*/

OS\_TASK\_STAT\_ID,

&OSTaskStatStk[0], /\* Set Bottom-Of-Stack \*/

OS\_TASK\_STAT\_STK\_SIZE,

(void \*)0, /\* No TCB extension \*/

OS\_TASK\_OPT\_STK\_CHK | OS\_TASK\_OPT\_STK\_CLR); /\* Enable stack checking + clear \*/

#else

(void)OSTaskCreateExt(OS\_TaskStat,

(void \*)0, /\* No args passed to OS\_TaskStat()\*/

&OSTaskStatStk[0], /\* Set Top-Of-Stack \*/

OS\_TASK\_STAT\_PRIO, /\* One higher than the idle task \*/

OS\_TASK\_STAT\_ID,

&OSTaskStatStk[OS\_TASK\_STAT\_STK\_SIZE - 1u], /\* Set Bottom-Of-Stack \*/

OS\_TASK\_STAT\_STK\_SIZE,

(void \*)0, /\* No TCB extension \*/

OS\_TASK\_OPT\_STK\_CHK | OS\_TASK\_OPT\_STK\_CLR); /\* Enable stack checking + clear \*/

#endif

#else

#if OS\_STK\_GROWTH == 1u

(void)OSTaskCreate(OS\_TaskStat,

(void \*)0, /\* No args passed to OS\_TaskStat()\*/

&OSTaskStatStk[OS\_TASK\_STAT\_STK\_SIZE - 1u], /\* Set Top-Of-Stack \*/

OS\_TASK\_STAT\_PRIO); /\* One higher than the idle task \*/

#else

(void)OSTaskCreate(OS\_TaskStat,

(void \*)0, /\* No args passed to OS\_TaskStat()\*/

&OSTaskStatStk[0], /\* Set Top-Of-Stack \*/

OS\_TASK\_STAT\_PRIO); /\* One higher than the idle task \*/

#endif

#endif

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

OSTaskNameSet(OS\_TASK\_STAT\_PRIO, (INT8U \*)(void \*)"uC/OS-II Stat", &err);

#endif

}

#endif

/\*

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

\* INITIALIZATION

\* INITIALIZE THE FREE LIST OF TASK CONTROL BLOCKS

\*

\* Description: This function is called by OSInit() to initialize the free list of OS\_TCBs.

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

static void OS\_InitTCBList (void)

{

INT8U ix;

INT8U ix\_next;

OS\_TCB \*ptcb1;

OS\_TCB \*ptcb2;

OS\_MemClr((INT8U \*)&OSTCBTbl[0], sizeof(OSTCBTbl)); /\* Clear all the TCBs \*/

OS\_MemClr((INT8U \*)&OSTCBPrioTbl[0], sizeof(OSTCBPrioTbl)); /\* Clear the priority table \*/

for (ix = 0u; ix < (OS\_MAX\_TASKS + OS\_N\_SYS\_TASKS - 1u); ix++) { /\* Init. list of free TCBs \*/

ix\_next = ix + 1u;

ptcb1 = &OSTCBTbl[ix];

ptcb2 = &OSTCBTbl[ix\_next];

ptcb1->OSTCBNext = ptcb2;

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

ptcb1->OSTCBTaskName = (INT8U \*)(void \*)"?"; /\* Unknown name \*/

#endif

}

ptcb1 = &OSTCBTbl[ix];

ptcb1->OSTCBNext = (OS\_TCB \*)0; /\* Last OS\_TCB \*/

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

ptcb1->OSTCBTaskName = (INT8U \*)(void \*)"?"; /\* Unknown name \*/

#endif

OSTCBList = (OS\_TCB \*)0; /\* TCB lists initializations \*/

OSTCBFreeList = &OSTCBTbl[0];

}/\*

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

\* CLEAR A SECTION OF MEMORY

\*

\* Description: This function is called by other uC/OS-II services to clear a contiguous block of RAM.

\*

\* Arguments : pdest is the start of the RAM to clear (i.e. write 0x00 to)

\*

\* size is the number of bytes to clear.

\*

\* Returns : none

\*

\* Notes : 1) This function is INTERNAL to uC/OS-II and your application should not call it.

\* 2) Note that we can only clear up to 64K bytes of RAM. This is not an issue because none

\* of the uses of this function gets close to this limit.

\* 3) The clear is done one byte at a time since this will work on any processor irrespective

\* of the alignment of the destination.

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

\*/

void OS\_MemClr (INT8U \*pdest,

INT16U size)

{

while (size > 0u) {

\*pdest++ = (INT8U)0;

size--;

}

}/\*

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

\* COPY A BLOCK OF MEMORY

\*

\* Description: This function is called by other uC/OS-II services to copy a block of memory from one

\* location to another.

\*

\* Arguments : pdest is a pointer to the 'destination' memory block

\*

\* psrc is a pointer to the 'source' memory block

\*

\* size is the number of bytes to copy.

\*

\* Returns : none

\*

\* Notes : 1) This function is INTERNAL to uC/OS-II and your application should not call it. There is

\* no provision to handle overlapping memory copy. However, that's not a problem since this

\* is not a situation that will happen.

\* 2) Note that we can only copy up to 64K bytes of RAM

\* 3) The copy is done one byte at a time since this will work on any processor irrespective

\* of the alignment of the source and destination.

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

\*/

void OS\_MemCopy (INT8U \*pdest,

INT8U \*psrc,

INT16U size)

{

while (size > 0u) {

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

size--;

}

}

/\*

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

\* SCHEDULER

\*

\* Description: This function is called by other uC/OS-II services to determine whether a new, high

\* priority task has been made ready to run. This function is invoked by TASK level code

\* and is not used to reschedule tasks from ISRs (see OSIntExit() for ISR rescheduling).

\*

\* Arguments : none

\*

\* Returns : none

\*

\* Notes : 1) This function is INTERNAL to uC/OS-II and your application should not call it.

\* 2) Rescheduling is prevented when the scheduler is locked (see OS\_SchedLock())

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

\*/

void OS\_Sched (void)

{

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

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

#endif

OS\_ENTER\_CRITICAL();

if (OSIntNesting == 0u) { /\* Schedule only if all ISRs done and ... \*/

if (OSLockNesting == 0u) { /\* ... scheduler is not locked \*/

OS\_SchedNew();

OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];

if (OSPrioHighRdy != OSPrioCur) { /\* No Ctx Sw if current task is highest rdy \*/

#if OS\_TASK\_PROFILE\_EN > 0u

OSTCBHighRdy->OSTCBCtxSwCtr++; /\* Inc. # of context switches to this task \*/

#endif

OSCtxSwCtr++; /\* Increment context switch counter \*/

OS\_TASK\_SW(); /\* Perform a context switch \*/

}

}

}

OS\_EXIT\_CRITICAL();

}

/\*

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

\* FIND HIGHEST PRIORITY TASK READY TO RUN

\*

\* Description: This function is called by other uC/OS-II services to determine the highest priority task

\* that is ready to run. The global variable 'OSPrioHighRdy' is changed accordingly.

\*

\* Arguments : none

\*

\* Returns : none

\*

\* Notes : 1) This function is INTERNAL to uC/OS-II and your application should not call it.

\* 2) Interrupts are assumed to be disabled when this function is called.

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

\*/

static void OS\_SchedNew (void)

{

#if OS\_LOWEST\_PRIO <= 63u /\* See if we support up to 64 tasks \*/

INT8U y;

y = OSUnMapTbl[OSRdyGrp];

OSPrioHighRdy = (INT8U)((y << 3u) + OSUnMapTbl[OSRdyTbl[y]]);

#else /\* We support up to 256 tasks \*/

INT8U y;

OS\_PRIO \*ptbl;

if ((OSRdyGrp & 0xFFu) != 0u) {

y = OSUnMapTbl[OSRdyGrp & 0xFFu];

} else {

y = OSUnMapTbl[(OS\_PRIO)(OSRdyGrp >> 8u) & 0xFFu] + 8u;

}

ptbl = &OSRdyTbl[y];

if ((\*ptbl & 0xFFu) != 0u) {

OSPrioHighRdy = (INT8U)((y << 4u) + OSUnMapTbl[(\*ptbl & 0xFFu)]);

} else {

OSPrioHighRdy = (INT8U)((y << 4u) + OSUnMapTbl[(OS\_PRIO)(\*ptbl >> 8u) & 0xFFu] + 8u);

}

#endif

}

/\*

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

\* DETERMINE THE LENGTH OF AN ASCII STRING

\*

\* Description: This function is called by other uC/OS-II services to determine the size of an ASCII string

\* (excluding the NUL character).

\*

\* Arguments : psrc is a pointer to the string for which we need to know the size.

\*

\* Returns : The size of the string (excluding the NUL terminating character)

\*

\* Notes : 1) This function is INTERNAL to uC/OS-II and your application should not call it.

\* 2) The string to check must be less than 255 characters long.

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

\*/

#if (OS\_EVENT\_NAME\_EN > 0u) || (OS\_FLAG\_NAME\_EN > 0u) || (OS\_MEM\_NAME\_EN > 0u) || (OS\_TASK\_NAME\_EN > 0u) || (OS\_TMR\_CFG\_NAME\_EN > 0u)

INT8U OS\_StrLen (INT8U \*psrc)

{

INT8U len;

len = 0u;

while (\*psrc != OS\_ASCII\_NUL) {

psrc++;

len++;

}

return (len);

}

#endif/\*

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

\* IDLE TASK

\*

\* Description: This task is internal to uC/OS-II and executes whenever no other higher priority tasks

\* executes because they are ALL waiting for event(s) to occur.

\*

\* Arguments : none

\*

\* Returns : none

\*

\* Note(s) : 1) OSTaskIdleHook() is called after the critical section to ensure that interrupts will be

\* enabled for at least a few instructions. On some processors (ex. Philips XA), enabling

\* and then disabling interrupts didn't allow the processor enough time to have interrupts

\* enabled before they were disabled again. uC/OS-II would thus never recognize

\* interrupts.

\* 2) This hook has been added to allow you to do such things as STOP the CPU to conserve

\* power.

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

\*/

void OS\_TaskIdle (void \*p\_arg)

{

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

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

#endif

p\_arg = p\_arg; /\* Prevent compiler warning for not using 'p\_arg' \*/

for (;;) {

OS\_ENTER\_CRITICAL();

OSIdleCtr++;

OS\_EXIT\_CRITICAL();

OSTaskIdleHook(); /\* Call user definable HOOK \*/

}

}/\*

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

\* STATISTICS TASK

\*

\* Description: This task is internal to uC/OS-II and is used to compute some statistics about the

\* multitasking environment. Specifically, OS\_TaskStat() computes the CPU usage.

\* CPU usage is determined by:

\*

\* OSIdleCtr

\* OSCPUUsage = 100 \* (1 - ------------) (units are in %)

\* OSIdleCtrMax

\*

\* Arguments : parg this pointer is not used at this time.

\*

\* Returns : none

\*

\* Notes : 1) This task runs at a priority level higher than the idle task. In fact, it runs at the

\* next higher priority, OS\_TASK\_IDLE\_PRIO-1.

\* 2) You can disable this task by setting the configuration #define OS\_TASK\_STAT\_EN to 0.

\* 3) You MUST have at least a delay of 2/10 seconds to allow for the system to establish the

\* maximum value for the idle counter.

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

\*/

#if OS\_TASK\_STAT\_EN > 0u

void OS\_TaskStat (void \*p\_arg)

{

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

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

#endif

p\_arg = p\_arg; /\* Prevent compiler warning for not using 'p\_arg' \*/

while (OSStatRdy == OS\_FALSE) {

OSTimeDly(2u \* OS\_TICKS\_PER\_SEC / 10u); /\* Wait until statistic task is ready \*/

}

OSIdleCtrMax /= 100uL;

if (OSIdleCtrMax == 0uL) {

OSCPUUsage = 0u;

#if OS\_TASK\_SUSPEND\_EN > 0u

(void)OSTaskSuspend(OS\_PRIO\_SELF);

#else

for (;;) {

OSTimeDly(OS\_TICKS\_PER\_SEC);

}

#endif

}

for (;;) {

OS\_ENTER\_CRITICAL();

OSIdleCtrRun = OSIdleCtr; /\* Obtain the of the idle counter for the past second \*/

OSIdleCtr = 0uL; /\* Reset the idle counter for the next second \*/

OS\_EXIT\_CRITICAL();

OSCPUUsage = (INT8U)(100uL - OSIdleCtrRun / OSIdleCtrMax);

OSTaskStatHook(); /\* Invoke user definable hook \*/

#if (OS\_TASK\_STAT\_STK\_CHK\_EN > 0u) && (OS\_TASK\_CREATE\_EXT\_EN > 0u)

OS\_TaskStatStkChk(); /\* Check the stacks for each task \*/

#endif

OSTimeDly(OS\_TICKS\_PER\_SEC / 10u); /\* Accumulate OSIdleCtr for the next 1/10 second \*/

}

}

#endif/\*

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

\* CHECK ALL TASK STACKS

\*

\* Description: This function is called by OS\_TaskStat() to check the stacks of each active task.

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

#if (OS\_TASK\_STAT\_STK\_CHK\_EN > 0u) && (OS\_TASK\_CREATE\_EXT\_EN > 0u)

void OS\_TaskStatStkChk (void)

{

OS\_TCB \*ptcb;

OS\_STK\_DATA stk\_data;

INT8U err;

INT8U prio;

for (prio = 0u; prio <= OS\_TASK\_IDLE\_PRIO; prio++) {

err = OSTaskStkChk(prio, &stk\_data);

if (err == OS\_ERR\_NONE) {

ptcb = OSTCBPrioTbl[prio];

if (ptcb != (OS\_TCB \*)0) { /\* Make sure task 'ptcb' is ... \*/

if (ptcb != OS\_TCB\_RESERVED) { /\* ... still valid. \*/

#if OS\_TASK\_PROFILE\_EN > 0u

#if OS\_STK\_GROWTH == 1u

ptcb->OSTCBStkBase = ptcb->OSTCBStkBottom + ptcb->OSTCBStkSize;

#else

ptcb->OSTCBStkBase = ptcb->OSTCBStkBottom - ptcb->OSTCBStkSize;

#endif

ptcb->OSTCBStkUsed = stk\_data.OSUsed; /\* Store the number of bytes used \*/

#endif

}

}

}

}

}

#endif/\*

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

\* INITIALIZE TCB

\*

\* Description: This function is internal to uC/OS-II and is used to initialize a Task Control Block when

\* a task is created (see OSTaskCreate() and OSTaskCreateExt()).

\*

\* Arguments : prio is the priority of the task being created

\*

\* ptos is a pointer to the task's top-of-stack assuming that the CPU registers

\* have been placed on the stack. Note that the top-of-stack corresponds to a

\* 'high' memory location is OS\_STK\_GROWTH is set to 1 and a 'low' memory

\* location if OS\_STK\_GROWTH is set to 0. Note that stack growth is CPU

\* specific.

\*

\* pbos is a pointer to the bottom of stack. A NULL pointer is passed if called by

\* 'OSTaskCreate()'.

\*

\* id is the task's ID (0..65535)

\*

\* stk\_size is the size of the stack (in 'stack units'). If the stack units are INT8Us

\* then, 'stk\_size' contains the number of bytes for the stack. If the stack

\* units are INT32Us then, the stack contains '4 \* stk\_size' bytes. The stack

\* units are established by the #define constant OS\_STK which is CPU

\* specific. 'stk\_size' is 0 if called by 'OSTaskCreate()'.

\*

\* pext is a pointer to a user supplied memory area that is used to extend the task

\* control block. This allows you to store the contents of floating-point

\* registers, MMU registers or anything else you could find useful during a

\* context switch. You can even assign a name to each task and store this name

\* in this TCB extension. A NULL pointer is passed if called by OSTaskCreate().

\*

\* opt options as passed to 'OSTaskCreateExt()' or,

\* 0 if called from 'OSTaskCreate()'.

\*

\* Returns : OS\_ERR\_NONE if the call was successful

\* OS\_ERR\_TASK\_NO\_MORE\_TCB if there are no more free TCBs to be allocated and thus, the task cannot

\* be created.

\*

\* Note : This function is INTERNAL to uC/OS-II and your application should not call it.

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

\*/

INT8U OS\_TCBInit (INT8U prio,

OS\_STK \*ptos,

OS\_STK \*pbos,

INT16U id,

INT32U stk\_size,

void \*pext,

INT16U opt)

{

OS\_TCB \*ptcb;

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

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

#endif

#if OS\_TASK\_REG\_TBL\_SIZE > 0u

INT8U i;

#endif

OS\_ENTER\_CRITICAL();

ptcb = OSTCBFreeList; /\* Get a free TCB from the free TCB list \*/

if (ptcb != (OS\_TCB \*)0) {

OSTCBFreeList = ptcb->OSTCBNext; /\* Update pointer to free TCB list \*/

OS\_EXIT\_CRITICAL();

ptcb->OSTCBStkPtr = ptos; /\* Load Stack pointer in TCB \*/

ptcb->OSTCBPrio = prio; /\* Load task priority into TCB \*/

ptcb->OSTCBStat = OS\_STAT\_RDY; /\* Task is ready to run \*/

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

ptcb->OSTCBDly = 0u; /\* Task is not delayed \*/

#if OS\_TASK\_CREATE\_EXT\_EN > 0u

ptcb->OSTCBExtPtr = pext; /\* Store pointer to TCB extension \*/

ptcb->OSTCBStkSize = stk\_size; /\* Store stack size \*/

ptcb->OSTCBStkBottom = pbos; /\* Store pointer to bottom of stack \*/

ptcb->OSTCBOpt = opt; /\* Store task options \*/

ptcb->OSTCBId = id; /\* Store task ID \*/

#else

pext = pext; /\* Prevent compiler warning if not used \*/

stk\_size = stk\_size;

pbos = pbos;

opt = opt;

id = id;

#endif

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

ptcb->OSTCBDelReq = OS\_ERR\_NONE;

#endif

#if OS\_LOWEST\_PRIO <= 63u /\* Pre-compute X, Y \*/

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

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

#else /\* Pre-compute X, Y \*/

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

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

#endif

/\* Pre-compute BitX and BitY \*/

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

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

#if (OS\_EVENT\_EN)

ptcb->OSTCBEventPtr = (OS\_EVENT \*)0; /\* Task is not pending on an event \*/

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

ptcb->OSTCBEventMultiPtr = (OS\_EVENT \*\*)0; /\* Task is not pending on any events \*/

#endif

#endif

#if (OS\_FLAG\_EN > 0u) && (OS\_MAX\_FLAGS > 0u) && (OS\_TASK\_DEL\_EN > 0u)

ptcb->OSTCBFlagNode = (OS\_FLAG\_NODE \*)0; /\* Task is not pending on an event flag \*/

#endif

#if (OS\_MBOX\_EN > 0u) || ((OS\_Q\_EN > 0u) && (OS\_MAX\_QS > 0u))

ptcb->OSTCBMsg = (void \*)0; /\* No message received \*/

#endif

#if OS\_TASK\_PROFILE\_EN > 0u

ptcb->OSTCBCtxSwCtr = 0uL; /\* Initialize profiling variables \*/

ptcb->OSTCBCyclesStart = 0uL;

ptcb->OSTCBCyclesTot = 0uL;

ptcb->OSTCBStkBase = (OS\_STK \*)0;

ptcb->OSTCBStkUsed = 0uL;

#endif

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

ptcb->OSTCBTaskName = (INT8U \*)(void \*)"?";

#endif

#if OS\_TASK\_REG\_TBL\_SIZE > 0u /\* Initialize the task variables \*/

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

ptcb->OSTCBRegTbl[i] = 0u;

}

#endif

OSTCBInitHook(ptcb);

OSTaskCreateHook(ptcb); /\* Call user defined hook \*/

OS\_ENTER\_CRITICAL();

OSTCBPrioTbl[prio] = ptcb;

ptcb->OSTCBNext = OSTCBList; /\* Link into TCB chain \*/

ptcb->OSTCBPrev = (OS\_TCB \*)0;

if (OSTCBList != (OS\_TCB \*)0) {

OSTCBList->OSTCBPrev = ptcb;

}

OSTCBList = ptcb;

OSRdyGrp |= ptcb->OSTCBBitY; /\* Make task ready to run \*/

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

OSTaskCtr++; /\* Increment the #tasks counter \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_TASK\_NO\_MORE\_TCB);

}