uC/OS RTOS Library

Revision 1.6
March 26, 2010

Table of Contents

1. Introduction

The NetBurner uC/OS RTOS is a preemptive multitasking real-time operating system designed to be very efficient and full featured, providing rapid real-time response and a small footprint. You can easily create and manage multiple tasks and communicate between tasks. The RTOS is integrated with the I/O system to make communication with the other system components, such as the TCP/IP Stack (**not** applicable for the **non-network** platforms such as the Mod5213), quick and easy.

Required Header Files

```
#include <ucos.h>
```

Location depends on your NetBurner platform: \nburn\include, \nburn\incluse_nn or \nburn\include_sc.

2. Function Summary

Task Functions

- OSTaskCreate -- Creates a new task
- OSTaskCreateName --- Creates a new task and assigns it a name (test string)
- OSSimpleTaskCreate --- A macro that sets up the stack and starts the task at the proper priority
- OSSimpleTaskCreateName --- Same as OSSimpleTaskCreate, but adds a name (text string)
- OSTaskDelete --- Deletes a task
- OSChangePrio --- Changes a tasks priority

Time Delay Functions

- OSTimeDly --- Delay or sleep for a fixed interval
- OSChangeTaskDly --- Changes the interval for a waiting task

Task Locking Functions

- OSLock --- Locks the OS and prevents task switches
- OSUnlock --- Unlocks the OS
- OSLockObj --- A C++ class to make task locking easy

Semaphore Functions

- OSSemInit --- Initializes an OS_SEM structure
- OSSemPost --- Post to a semaphore
- OSSemPend --- Pend on a semaphore
- OSSemPendNoWait --- Pend on a semaphore without waiting

Mail Box Functions

- OSMboxInit --- Initializes an OS_MBOX structure
- OSMboxPost --- Post to a mailbox
- OSMboxPend --- Pend on a mailbox
- OSMboxPendNoWait --- Pend on a mailbox without waiting

Queue Functions

- OSQInit --- Initializes an OS_QUEUE structure
- OSQPost --- Post to a queue
- OSQPend --- Pend on a queue
- OSQPendNoWait --- Pend on a queue without waiting

FIFO Functions

- OSFifoInit --- Initializes an OS_FIFO structure
- OSFifoPost --- Post to a fifo
- OSFifoPostFirst --- Post to the head of a fifo
- OSFifoPend --- Pend on a fifo
- OSFifoPendNoWait --- Pend on a fifo without waiting

OS Critical Functions

The OS_CRIT and related functions implement an OS function referred to as a mutex or counted critical section. Their purpose is to provide a mechanism to protect critical data with a structure or resource. Some examples of its use would be to protect the data in a linked list, or to control an external command interface. You will want to use this kind of critical section when you need to keep one task from interrupting another task when doing manipulations in a set.

- OSCritInit --- Initializes the critical section
- OSCritEnter --- Tries to enters or claim the critical section
- OSCritEnterNoWait --- Tries to enter or claim the critical section without waiting
- OSCritLeave --- Releases the critical section
- Two Examples

Interrupt Functions

OSIntEnter and OSIntExit are taken care of in the **INTERRUPT Macro** for **all** NetBurner Platforms. For more information, please read the Interrupts section in NetBurner Runtime Libraries User's Manual (in **C:\Nburn\docs\NetBurnerRuntimeLibrary**).

- OSIntEnter --- Must be called when a user interrupt is entered
- OSIntExit --- Must be called when a user interrupt is exited

User Critical Functions

These function like a level 7 interrupt. **Important:** You will have full processor time once you enter the section, but **all** uC/OS functions and features **will** be disabled **until** you exit the section. All hardware peripherals interrupts will also be disabled.

- USER_ENTER_CRITICAL --- Sets a level 7 interrupt mask when entered
- USER_EXIT_CRITICAL --- Sets the interrupt mask to the value before critical section was entered

Debugging Functions

The debugging routines are only **valid** when **UCOS_STACK_CHECK** is **defined**.

- OSDumpTCBStacks --- Dumps all of the task stack information to stdout
- OSDumpTasks --- Dumps all of the task info to stdout

Flag Functions

- OSFlagCreate ---Creates and initializes an OS_FLAGS object
- OSFlagSet --- Sets the bits asserted bits_to_set
- OSFlagState ---Returns the current value of flags
- OSFlagClear --- Clears the bits asserted in bits_to_clr
- OSFlagPendAll ---- Waits until all of the flags indicated by mask are set
- OSFlagPendNoWait --- Checks (but does not wait) if all of the flags indicated by the mask are set
- OSFlagPendAny --- Waits until any of the flags indicated by the mask are set
- OSFlagPendAnyNoWait ---Checks (but does not wait) if any of the flags indicated by the mask are set

2.1. OSTaskCreate

Synopsis:

```
BYTE OSTaskCreate( void ( * task )( void * taskfunc ), void * data,
                   void * pstacktop, void * pstackbot, BYTE priority );
```

Description:

This function creates a new task. You must allocate storage for the stack that this new task will use and it must be 4 byte aligned. Task priorities can range from 1 to 63, where 63 is the lowest priority level and 1 is highest priority level. The recommended user priority levels for your application are in the range of 46 to 62. This avoids any conflicts with network communications.

Warning: The uC/OS can only have one task at each priority.

Parameters:

Type	Description
taskfunc	The address of the function where this task will start executing.
data	The data to pass to the task function.
pstacktop	The highest address of the stack space.
pstackbot	The lowest address of the stack space.
priority	The priority for this new task (63 is lowest priority and 1 is highest). Look in **C:\Nburn\include\constants.h** to see which priorities are used by the OS. For **non-network platforms (e.g. Mod5213)**, look in **C:\Nburn\include_nn\constants.h** to see which priorities are used by the OS.

Returns:

OS_NO_ERR (0) --- If successful
OS_PRIO_EXIST (40) --- If the requested priority already exists

See Also:

OSTaskDelete --- Delete a task
OSChangePrio --- Change a task's priority
OSSimpleTaskCreate --- A macro that sets up the stack and starts the task at the proper priority

Example:

```
// Make sure they're 4 byte aligned to keep the Coldfire happy
asm( " .align 4 " );

DWORD MyTaskStk[USER_TASK_STK_SIZE] __attribute__( ( aligned( 4 ) ) );

// The function the new task will start in. pdata will have the value
// of my_data as provided in the OSTaskCreate Call
void mytask(void * pdata)
 {
 }

if (OSTaskCreate(mytask,
                 (void*)my_data,
                 (void*)&MyTaskStk[USER_TASK_STK_SIZE],
                 (void *)MyTaskStk, MyPrio
                )!=OS_NO_ERR)
                  { // Handle error
                  }
```

2.2. OSTaskCreatewName

Synopsis:

```
BYTE OSTaskCreatewName( void ( *task ) ( void *dptr ),
                        void *data,
                        void *pstktop,
                        void *pstkbot,
                        BYTE  prio,
                        const char * name);
```

Description:

Only available on select network platforms. Located in \nburn\include directory. Same as OSTaskCreate, but adds a name that can be assigned to the task, which makes it easier to identify the task when using the debugger, Task Scan or Smart Traps.

2.3. OSSimpleTaskCreate (MACRO)

Synopsis:

```
OSSimpleTaskCreate( function, priority );
```

Description:

This Macro sets up the stack and starts the task at the proper priority. For example, if I want to start a task called "my_task", I would use the OSSimpleTaskCreate macro as follows:

```
void my_task( void *)
  {
    The my_task function
  }

OSSimpleTaskCreate( my_task, MAIN_PRIO-1 );
```

Parameters:

Type	Description
function	The address of the function where this task will start executing.
priority	The priority for this new task (63 is lowest priority, 1 is highest). Look in **C:\Nburn\include\constants.h** to see which priorities are used by the OS. For **non-network platforms (e.g. Mod5213)**, look in **C:\Nburn\include_nn\constants.h** to see which priorities are used by the OS.

See Also:

OSTaskCreate --- Create a new task
OSTaskDelete --- Delete a task
OSChangePrio --- Change a task's priority

2.4. OSSimpleTaskCreatewName (MACRO)

Synopsis:

```
OSSimpleTaskCreatewName( function, priority, name );
```

Description:

Only available on select network platforms. Located in \nburn\include directory. Same as OSTaskCreate, but adds a name that can be assigned to the task, which makes it easier to identify the task when using the debugger, Task Scan or Smart Traps.

```
void my_task( void *)
  {
    The my_task function
  }

OSSimpleTaskCreatewName( my_task, MAIN_PRIO-1, "My Task" );
```

2.5. OSTaskDelete

Synopsis:

```
void OSTaskDelete( void );
```

Description:

This function deletes the current calling task, but we do not recommend the use of this function because it can cause memory leaks. The preferred method for terminating a task is to set a flag or semaphore that the task is listening for. The flag can then be set by an outside task, which enables the task to be deleted to free any resources and terminate gracefully by simply returning.

Parameters:

None

Returns:

Nothing --- This is a void function

See Also:

OSTaskCreate --- Create a new task
OSSimpleTaskCreate --- A macro that sets up the stack and starts the task at the proper priority
OSChangePrio ---Change a task's priority

2.6. OSChangePrio

Synopsis:

```
BYTE OSChangePrio( BYTE newpriority );
```

Description:

This function changes the priority of the calling task. Note: The uC/OS can only have one task at each priority level. Task priorities can range from 1 to 63, where 63 is the lowest priority level and 1 is highest priority level. Priorities 1-4 and the NetBurner system priority levels are reserved as described below. The recommended user priority levels for your application are in the range of 46 to 62. This avoids any conflicts with network communications.

System priorities are defined in C:\Nburn\include\constants.h for all network platforms and in C:\Nburn\include_nn\constants.h for all non-network (e.g. Mod5213) platforms.

```
#define MAIN_PRIO (50)
#define HTTP_PRIO (45)
#define PPP_PRIO (44)
#define TCP_PRIO (40)
#define IP_PRIO (39)
#define ETHER_SEND_PRIO (38)
```

Parameter:

Type	Name	Description
BYTE	newpriority	The new priority of the calling task.

Returns:

OS_NO_ERR (0) --- If successful
OS_PRIO_EXIST (40) --- If the requested priority already exists

See Also:

OSTaskCreate --- Create a new task
OSTaskDelete --- Delete a task
OSSimpleTaskCreate --- A macro that sets up the stack and starts the task at the proper priority

2.7. OSTimeDly

Synopsis:

```
void OSTimeDly( WORD ticks );
```

Description:

This function delays this task for "ticks" ticks of the system timer. Remember: The number of ticks per second is defined by the constant TICKS_PER_SECOND.

Parameter:

Type	Name	Description
WORD	ticks	The number of ticks per second

Returns:

Nothing --- This is a void function

See Also:

OSChangeTaskDly --- Change the interval for a waiting task

Example:

```
OSTimeDly( 5*TICKS_PER_SECOND );   // Delay for 5 seconds
```

2.8. OSChangeTaskDly

Synopsis:

```
void OSChangeTaskDly( WORD task_prio, WORD newticks );
```

Description:

This function allows the User to modify the timeout delay for a task that is waiting.

Warning: Use of this function is discouraged.

Parameters:

Type	Name	Description
WORD	task_prio	The task's priority.
WORD	newticks	The **new** number of ticks per second.

Returns:

Nothing --- This is a void function

See Also:

OSTimeDly --- Delay or Sleep for a fixed interval
OSSemPend --- Pend on a semaphore
OSMboxPend --- Pend on a mailbox
OSQPend --- Pend on a queue
OSFifoPend --- Pend on a fifo

2.9. OSLock

Synopsis:

```
void OSLock( void );
```

Description:

Calling the OSLock function will prevent the OS from changing tasks. This is used to protect critical variables that must be accessed one task at a time. Use the OSUnlock function to release your lock. **Important:** You must call OSUnlock once **for each** call to OSLock.

Warning: Do not keep a task locked for long period of time, or the performance of the network subsystem will degrade, and eventually loose packets.

Parameters:

None

Returns:

Nothing --- This is a void function

See Also:

OSUnlock --- Unlocks the OS
OSLockObj --- A C++ class to make task locking easy

2.10. OSUnlock

Synopsis:

```
void OSUnlock( void );
```

Description:

This function unlocks the OS. **Important:** You must call OSUnlock once **for each** call to OSLock.

Parameters:

None

Returns:

Nothing --- This is a void function

See Also:

OSLock --- Locks the OS and prevent task switches
OSLockObj --- A C++ class to make task locking easy

2.11. OSLockObj

Synopsis:

```
class OSLockObj
  {
   public:
   OSLockObj();
   ~OSLockObj();
  };
```

Description:

A simple C++ wrapper class that helps use OS locks effectively. When an OSLockObj is constructed it locks the OS. When it is destructed it unlocks the OS. If you have a function that needs an OS lock and has multiple points of exit, create an OSLockObj at the beginning of the function. **Important:** No matter how you leave the function, the destructor will release the lock.

Example:

```
int foo()
  {
    // The destructor will unlock the OS when lock goes out of scope
    OSLockObj lock;
    ...
    if () return 1;
    ...
    if () return 3;
    ...
    ...
    if () return 0;
  }
```

See Also:

OSLock --- Locks the OS and prevents task switches
OSUnlock --- Unlocks the OS

2.12. OSSemInit

Synopsis:

```
BYTE OSSemInit( OS_SEM * psem, long value );
```

Description:

Semaphores are used to control access to shared resource, or to communicate between tasks. This function is used to initialize a semaphore structure. **Note:** This must be done **before** using a semaphore.

Parameters:

Type	Name	Description
OS_SEM	*psem	A pointer to the OS_SEM structure to initialize.
long	value	The initial count value for the semaphore.

Returns:

OS_NO_ERR (0) --- If successful
OS_SEM_ERR (50) --- If value is < 0 (zero), it cannot initialize

Example:

```
OS_SEM MySemaphore;
.
.
OSSemInit(& MySemaphore,0);
.
.
// In a different task/function...
OSSemPost(& MySemaphore); // Add one to the semaphores value
.
.
// In a yet another different task/function...
// Wait 5 seconds or until the semaphore has a positive value
// Decrement the semaphore if we don't timeout...
if (OSSemPend(& MySemaphore, 5*TICKS_PER_SECOND)==OS_TIMEOUT){// We timed out
the 5 seconds}else {// We got the semaphore}
```

See Also:

OSSemPost --- Post to a semaphore
OSSemPend --- Pend on a semaphore

2.13. OSSemPost

Synopsis:

```
BYTE OSSemPost( OS_SEM  * psem );
```

Description:

This function increases the value of the semaphore by one. **Note:** If any **higher** priority tasks were **waiting** on the semaphore - it **releases** them.

Parameter:

Type	Name	Description
OS_SEM	*psem	A pointer to the OS_SEM structure to initialize.

Returns:

OS_NO_ERR (0) --- If successful
OS_SEM_OVF (51) --- If the value of the semaphore overflows

See Also:

OSSemInit --- Initialize an OS_SEM structure
OSSemPend --- Pend on a semaphore

2.14. OSSemPend

Synopsis:

```
BYTE OSSemPend( OS_SEM * psem, WORD timeout );
```

Description:

Wait timeout ticks for the value of the semaphore to be non zero. **Note:** A timeout value of 0 (zero) waits forever.

Parameters:

Type	Name	Description
OS_SEM	*psem	A pointer to an OS_SEM structure.
WORD	timeout	The number of time ticks to wait.

Returns:

OS_NO_ERR (0) --- If successful
OS_TIMEOUT (10) --- If the function timed out or if the NoWait function failed

See Also:

OSSemInit --- Initialize an OS_SEM structure
OSSemPendNoWait --- Does not wait for the value of the semaphore to be non zero
OSSemPost --- Post to a semaphore

2.15. OSSemPendNoWait

Synopsis:

```
BYTE OSSemPendNoWait( OS_SEM * psem );
```

Description:

OSSemPendNoWait is identical to the OSSemPend function, but it does not wait.

Parameter:

Type	Name	Description
OS_SEM	*psem	A pointer to the OS_SEM structure.

Returns:

OS_NO_ERR (0) --- If successful
OS_TIMEOUT (10) --- If it fails

See Also:

OSSemInit --- Initialize an OS_SEM structure
OSSemPend --- Pend on a semaphore
OSSemPost --- Post to a semaphore

2.16. OSMboxInit

Synopsis:

```
BYTE OSMboxInit( OS_MBOX * pmbox, void * msg );
```

Description:

Mailboxes are used to communicate between tasks. This function is used to initialize an OS_MBOX structure. **Note:** This must be done **before** using the mailbox.

Parameters:

Type	Name	Description
OS_MBOX	*pmbox	A pointer to the OS_MBOX structure to initialize.
void	*msg	The initial mail box message (NULL) for none.

Returns:

OS_NO_ERR (0) --- If successful

Example:

```
OS_MBOX MyMailBox;
OSMboxInit(& MyMailBox,0);
// In a different task/function...
// Put a message in the Mailbox.
OSMboxPost(& MyMailBox, (void *)somevalue);
// In a yet another different task/function...
// Wait 5 seconds or until the mailbox has a message
BYTE err;
void * pData=OSMboxPend(& MyMailBox, 5*TICKS_PER_SECOND,&err);
if (pData==NULL)
{ // We timed out the 5 seconds
}
else
{ // We got the message
}
```

See Also:

OSMboxPend --- Pend on a mailbox
OSMboxPost --- Post to a mailbox
OSSemPendNoWait --- Does not wait for the value of the semaphore to be non zero

2.17. OSMboxPost

Synopsis:

```
BYTE OSMboxPost( OS_MBOX * pmbox, void * msg );
```

Description:

This function posts a message to a Mail box.

Parameters:

Type	Name	Description
OS_MBOX	*pmbox	A pointer to an OS_MBOX structure.
void	*msg	The message to post.

Returns:

OS_NO_ERR (0) --- If successful
OS_MBOX_FULL (20) --- If the mailbox is full

See Also:

OSMboxInit --- Initialize an OS_MBOX structure
OSMboxPend --- Pend on a Mailbox
OSSemPendNoWait --- Does not wait for the value of the semaphore to be non zero

2.18. OSMboxPend

Synopsis:

```
void * OSMboxPend( OS_MBOX * pmbox, WORD timeout, BYTE * err );
```

Description:

Wait timeout ticks for some other task to post to the Mailbox. **Note:** OSMboxPend will wait forever if **0** (zero) is specified.

Parameters:

Type	Name	Description
OS_MBOX	*pmbox	A pointer to an OS_MBOX structure.
WORD	timeout	The number of time ticks to wait.
Byte	*err	A variable to receive the result code.

Returns:

The posted message
NULL --- If the function timed out

Note: err can have either OS_NO_ERR or OS_TIMEOUT return codes.

See Also:

OSMboxInit --- Initialize an OS_MBOX structure
OSMboxPendNoWait --- Does not wait for some other task to post to the Mailbox
OSMboxPost --- Post to a Mailbox

2.19. OSMboxPendNoWait

Synopsis:

```
void * OSMboxPendNoWait( OS_MBOX * pmbox, BYTE * err );
```

Description:

OSMboxPendNoWait is identical to the OSMboxPend function, but it does **not** wait.

Parameters:

Type	Name	Description
OS_MBOX	*pmbox	A pointer to an OS_MBOX structure.
Byte	*err	A variable to receive the result code.

Returns:

The posted message
NULL --- If it fails

Note: err can have either OS_NO_ERR **or** OS_TIMEOUT return codes.

See Also:

OSMboxPend --- Pend on a Mailbox
OSMboxPendNoWait --- Does not wait for some other task to post to the Mailbox
OSMboxPost --- Post to a Mailbox

2.20. OSQInit

Synopsis:

```
BYTE OSQInit( OS_Q * pq, void * * start, BYTE siz );
```

Description:

A queue functions as a fixed size FIFO for communication between tasks. This function initializes an OS_Q structure.

Parameters:

Type	Name	Description
OS_Q	*pq	A pointer to an OS_Q structure.
void	**start	A pointer to an array of (void *) pointers to hold queue messages.
BYTE	siz	The number of pointers in the Q data storage area.

Returns:

OS_NO_ERR (0) --- If successful

Example:

```
OS_Q MyQueue;
void * MyQueueStorage[NUM_ELEMENTS];
OSQInit(& MyQueue,MyQueueStorage,NUM_ELEMENTS);
// In a different task/function...
// Put a message in the Queue
OSQPost(& MyQueue, (void *)somevalue);
// In a yet another different task/function...
// Wait 5 seconds or until the queue has a message.
BYTE err;
void * pData=OSQPend(& MyQueue, 5*TICKS_PER_SECOND,&err);
if (pData==NULL)
{// We timed out the 5 seconds
}
else
{// We got the message
}
```

See Also:

OSQPost --- Post to a Queue
OSQPend --- Pend on a Queue
OSQPendNoWait --- Does not wait for another task to post to the queue

2.21. OSQPost

Synopsis:

```
BYTE OSQPost( OS_Q * pq, void * msg );
```

Description:

This function posts a message to a Queue. **Note:** Any **higher** priority task **waiting** on this queue **will** be started.

Parameters:

Type	Name	Description
OS_Q	*pq	A pointer to an OS_Q structure.
void	*msg	The message to be posted to the queue.

Returns:

OS_NO_ERR (0) --- If successful
OS_Q_FULL (30) --- If the queue is full and has no more room

See Also:

OSQInit --- Initialize an OS_QUEUE structure
OSQPend --- Pend on a Queue
OSQPendNoWait --- Does not wait for another task to post to the queue

2.22. OSQPostFirst

Synopsis:

```
BYTE OSQPostFirst( OS_Q *pq, void *msg );
```

Description:

This function posts a message like OSQPost, but posts the message at the head of the queue. Note that any higher priority task waiting on this queue will be started.

Parameters:

Type	Name	Description
OS_Q	*pq	A pointer to an OS_Q structure.
void	*msg	The message to post at the head of the queue.

Returns:

OS_NO_ERR (0) --- Successfully posted at the head of the queue.
OS_Q_FULL (30) --- The queue is already full; cannot post message.

See Also:

OSQInit --- Initialize an OS_Q structure.
OSQPost --- Post to a queue.
OSQPend --- Pend on a queue.
OSQPendNoWait --- Does not wait for another task to post to the queue.

2.23. OSQPend

Synopsis:

```
void * OSQPend( OS_Q * pq, WORD timeout, BYTE * err );
```

Description:

Wait timeout ticks for another task to post to the queue. **Note:** A timeout value of 0 (zero) waits forever. An err **holds** the error code if the function fails.

Parameters:

Type	Name	Description
OS_Q	*pq	A pointer to an OS_Q structure.
WORD	timeout	The number of time ticks to wait.
BYTE	*err	A variable to receive the result code.

Returns:

The posted message
NULL --- If the function failed

Note: err can have OS_NO_ERR **or** OS_TIMEOUT return codes

See Also:

OSQInit --- Initialize an OS_QUEUE structure
OSQPendNoWait --- Does not wait for another task to post to the queue
OSQPost --- Post to a Queue

2.24. OSQPendNoWait

Synopsis:

```
void * OSQPendNoWait( OS_Q * pq, BYTE * err );
```

Description:

OSQPendNoWait is identical to the OSQPend function but it does not wait.

Parameters:

Type	Name	Description
OS_Q	*pq	A pointer to an OS_Q structure.
BYTE	*err	A variable to receive the result code.

Returns:

The posted message
NULL --- If the function failed

Note: err can have OS_NO_ERR **or** OS_TIMEOUT return codes

See Also:

OSQPend --- Pend on a Queue
OSQInit --- Initialize an OS_QUEUE structure
OSQPost --- Post to a Queue

2.25. OSFifoInit

Synopsis:

```
BYTE OSFifoInit( OS_FIFO * pFifo );
```

Description:

A FIFO is used to pass structures from one task to another. **Note:** The structure to be passed must have an unused (void *) pointer as its first element. This precludes passing C++ objects with virtual member functions.

Parameter:

Type	Name	Description
OS_FIFO	*pFifo	A pointer to an OS_FIFO structure.

Returns:

OS_NO_ERR (0) --- If successful

See Also:

OSFifoPost --- Post to a fifo
OSFifoPostFirst --- Post to the head of a fifo
OSFifoPend --- Pend on a fifo
OSFifoPendNoWait --- Pend on a fifo without waiting

Example:

```
OS_FIFO MyFifo;
typedef struct
{void * pUsedByFifo; // Don't modify this value, and keep it first
// The other elements in my structure
}MyStructure;
OSFifoInit(& MyFifo);
// In a different task/function...
MyStructure mydata;
// Put a message in the Fifo
OSFifoPost(& MyFifo, (OS_FIFO_EL *)&mydata);
// In yet another different task/function...
// Wait 5 seconds or until the Fifo has a object
BYTE err;
MyStructure * pData= (MyStructure *)OSFifoPend(& MyQueue,
5*TICKS_PER_SECOND);
if (pData==NULL)
{// we timed out the 5 seconds}
```

```
else
{// We got the object
}
```

2.26. OSFifoPost

Synopsis:

```
BYTE OSFifoPost( OS_FIFO * pFifo, OS_FIFO_EL * pToPost );
```

Description:

This function posts to a FIFO. **Note:** See the description of FIFOs in OSFifoInit for details on how to use this function.

Parameters:

Type	Name	Description
OS_FIFO	*pFifo	A pointer to an OS_FIFO structure.
OS_FIFO_EL	*pToPost	A pointer to the user's structure cast as an OS_FIFO_EL to be posted to the Fifo.

Returns:

OS_NO_ERR (0) --- If successful

See Also:

OSFifoInit --- Initialize an os_fifo structure
OSFifoPostFirst --- Post to the head of a fifo
OSFifoPend --- Pend on a fifo
OSFifoPendNoWait --- Pend on a fifo without waiting

2.27. OSFifoPostFirst

Synopsis:

```
BYTE OSFifoPostFirst( OS_FIFO * pFifo, OS_FIFO_EL * pToPost );
```

Description:

This function is identical to OSFifoPost (post to a FIFO), but the element posted is put on the beginning of the FIFO list. So, the task that pends next will get the structure/object posted here, instead of any prior objects posted to the FIFO. **Note:** See the description of FIFOs in OSFifoInit for details on how to use this function.

Parameters:

Type	Name	Description
OS_FIFO	*pFifo	A pointer to an OS_FIFO structure.
OS_FIFO_EL	*pToPost	A pointer to the user's structure cast as an OS_FIFO_EL to be posted to the Fifo.

Returns:

OS_NO_ERR (0) --- If successful

See Also:

OSFifoInit --- Initialize an os_fifo structure
OSFifoPost --- Post to a fifo
OSFifoPend --- Pend on a fifo
OSFifoPendNoWait --- Pend on a fifo without waiting

2.28. OSFifoPend

Synopsis:

```
OS_FIFO_EL * OSFifoPend( OS_FIFO * pFifo, WORD timeout );
```

Description:

This function pends on a FIFO. **Note:** See the description of FIFOs in OSFifoInit for details on how to use this function.

Parameters:

Type	Name	Description
OS_FIFO	*pFifo	A pointer to an OS_FIFO structure.
WORD	timeout	The number of ticks to wait on the Fifo.

Returns:

A pointer to the posted structure
NULL --- If the function timed out

See Also:

OSFifoInit --- Initialize an os_fifo structure
OSFifoPost --- Post to a fifo
OSFifoPostFirst --- Post to the head of a fifo
OSFifoPendNoWait --- Pend on a fifo without waiting

2.29. OSFifoPendNoWait

Synopsis:

```
OS_FIFO_EL * OSFifoPendNoWait( OS_FIFO * pFifo );
```

Description:

This function is identical to the OSFifoPen function, but it does **not** wait.

Parameter:

Type	Name	Description
OS_FIFO	*pFifo	A pointer to an OS_FIFO structure.

Returns:

A pointer to the posted structure
NULL --- If there was nothing in the fifo

See Also:

OSFifoInit --- Initialize an os_fifo structure
OSFifoPost --- Post to a fifo
OSFifoPostFirst --- Post to the head of a fifo
OSFifoPend --- Pend on a fifo

2.30. OSCritInit

Synopsis:

```
BYTE OSCritInit( OS_CRIT * pCrit );
```

Description:

This function initializes the critical section. Important: You must call OSCritInit before using the critical section. Note: This function should be part of the initialization process.

Parameter:

Type	Name	Description
OS_CRIT	*pCrit	A pointer to the critical section.

Returns:

OS_NO_ERR --- If successful

See Also:

OSCritEnter --- Tries to enters or claim the critical section
OSCritEnterNoWait --- Tries to enter or claim the critical section without waiting
OSCritLeave --- Releases the critical section

2.31. OSCritEnter

Synopsis:

```
BYTE OSCritEnter( OS_CRIT * pCrit, WORD timeout );
```

Description:

This function tries to enter or claim the critical section. Important: You must call OSCritLeave once for each successful OSCritEnter call to release the critical section so that another task can manipulate it.

Parameters:

Type	Name	Description
OS_Crit	*pCrit	A pointer to the critical section we want to enter/claim.
WORD	timeout	How many time ticks do we want to wait for this critical section? **Note:** A timeout of 0 (**zero**) waits forever.

Returns:

OS_NO_ERR --- If we were successful in claiming the critical section or if our task owns it
OS_TIMEOUT ---- If we were unable to claim the section

See Also:

OSCritInit --- Initializes the critical section
OSCritEnterNoWait --- Tries to enter or claim the critical section without waiting
OSCritLeave --- Releases the critical section

2.32. OSCritEnterNoWait

Synopsis:

```
BYTE OSCritEnterNoWait( OS_CRIT * pCrit );
```

Description:

This function tries to enter or claim the critical section. However, this function does not wait if it is unable to enter or claim the critical section. Important: You must call OSCritLeave once for each successful OSCritEnterNoWait call to release the critical section so another task can manipulate it.

Parameter:

Type	Name	Description
OS_CRIT	*pCrit	A pointer to the critical section we want to enter/claim.

Returns:

OS_NO_ERR --- If we were successful in claiming the critical section, or if our task owns it
OS_TIMEOUT --- If we were unable to claim the section

See Also:

OSCritInit --- Initializes the critical section
OSCritEnter --- Tries to enters or claim the critical section
OSCritLeave --- Releases the critical section

2.33. OSCritLeave

Synopsis:

```
BYTE OSCritLeave( OS_CRIT * pCrit );
```

Description:

This function releases the critical section. Important: This function must be called once for each successful OSCritEnter or OSCritEnterNoWait call to release the critical section so another task can manipulate it.

Parameter:

Type	Name	Description
OS_CRIT	*pCrit	A pointer to the critical section we want to leave/release.

Returns:

OS_NO_ERR --- If we were successful in releasing the critical section OS_CRIT_ERR --- If we are trying to release a critical section that we do not own

See Also:

OSCritInit --- Initializes the critical section
OSCritEnter --- Tries to enters or claim the critical section
OSCritEnterNoWait --- Tries to enter or claim the critical section without waiting

2.34. Examples

2.34.1. Example # 1

When I want to insert something at the beginning of a doubly linked list:

```
typedef MyObject
 {
    MyObject * pNext;
    MyObject * pPrev;
    *
    *
    *
 };

MyObject* pHead;
void InsertAtHead(MyObject * newObject)
 {
    /* Step 1 */
    newObject->pNext=pHead;
    /* Step 2 */
    newObject->pPrev=NULL;
    /* Step 3*/
    pHead->pPrev=newObject;
    /* Step 4 */
    pHead=newObject;
 }
```

Suppose another higher priority task interrupts us (between steps 3 and 4) and inserts its own element at the head of the list. The list would not be correct, because pHead is reset to the object we are inserting in the task that was interrupted.

To prevent this type of error from happening, you should use an OS_CRIT counted critical section. This will not lock or otherwise restrict the RTOS unless another task wants to claim the same critical section. The OS_CRIT object should be declared globally or as part of the structure/object you want to protect. Therefore, in the previous example, we would change the code to:

```
MyObject* pHead;
OS_CRIT MyListCritical;
void InsertAtHead(MyObject * newObject)
 {
    OSCritEnter(&MyListCritical,0);
    /* Step 1 */
    newObject->pNext=pHead;
    /* Step 2 */
    newObject->pPrev=NULL;
    /* Step 3*/
    pHead->pPrev=newObject;
    /* Step 4 */
    pHead=newObject;
    OSCritLeave(&MyListCritical);
 }
```

Now, if a higher priority task tries to interrupt us between steps 3 and 4, the higher priority task will interrupt and call our InsertAtHeadFunction. But, as soon as it gets to the OSCritEnter call, it will be stopped.

The higher priority task will discover that the MyListCritical object is already claimed/occupied by a lower priority task, so it will block and allow the lower priority tasks to run. This should allow our interrupted task to continue to the point where it leaves the critical section. When this happens, the critical section becomes available, and the higher priority task will run.

2.34.2. Example # 2

Suppose we have an instrument like a GPS (or a DVM) connected to one of our serial ports. This instrument answers questions. The questions may come from a logging task, a web page request, a Telnet session, etc. The problem arises when a low priority task (e.g. logging) asks "Where are we?" and before the GPS answers, the higher priority task (e.g. Telnet) asks "What time is it"?

Example pseudo code:

```
Logging task...
/*1 */
Send(fdserial,"Where are we?");
/*2 */
WaitForResponsePacket(fdserial, buffer);
/*3*/
SavePosition(buffer);
Telnet task
/*1 */
Send(fdserial,"What time is it?");
/*2 */
WaitForResponsePacket(fdserial, buffer);
/*3*/
SendReply toRequestor(buffer);
```

The logging task does step 1, it sends "Where are we?" The telnet task interrupts, and sends "What time is it?" The GPS answers the first question - "Where are we?" Because the Telnet task is a higher priority, it receives this ("Where are we?") response. Then the logging task wakes up and gets the next response - to the second question ("What time is it?"). Now we have logged the time to the where, and the where to the time request.

Note: Adding an OSCritEnter function before step 1 in both tasks, and an OSCritLeave function after step 2 in each task will solve this problem.

2.35. OSIntEnter

Synopsis:

```
void OSIntEnter( void );
```

Description:

This function must be called in any user interrupt routine, before any RTOS functions are called. It must be followed by a call to OSIntExit. Important: OSIntEnter is taken care of in the INTERRUPT Macro for all NetBurner Platforms. Please read the Chapter on Interrupts in your NetBurner Runtime Libraries User's Manual for additional information. By default, this manual is found in C:\Nburn\docs.

Parameters:

None

Returns:

Nothing --- This is a void function

See Also:

OSIntExit --- Must be called when a user interrupt is exited

2.36. OSIntExit

Synopsis:

```
void OSIntExit( void );
```

Description:

This function must be called when a user interrupt is exited. Important: OSIntExit is taken care of in the INTERRUPT Macro for all NetBurner Platforms. Please read the Chapter on Interrupts in your NetBurner Runtime Libraries User's Manual for additional information. By default, this manual is found in C:\Nburn\docs.

Parameters:

None

Returns:

Nothing ---This is a void function

See Also:

OSIntExit --- Must be called when a user interrupt is exited

2.37. USER_ENTER_CRITICAL

Synopsis:

```
void USER_ENTER_CRITICAL( );
```

Description:

This function sets a level 7 interrupt mask when entered, allowing the user to have full processor time. This function will also disable all uCOS functionality and block all hardware interrupts. Important: You must call USER_EXIT_CRITICAL once for each USER_ENTER_CRITICAL call to release the critical section.

Parameters:

None

Returns:

Nothing --- This is a void function

See Also:

USER_EXIT_CRITICAL --- Sets the interrupt mask to the value before critical section was entered

2.38. USER_EXIT_CRITICAL

Synopsis:

```
void USER_EXIT_CRITICAL( );
```

Description:

This function sets the interrupt mask to the value before the critical section was entered. Important: You must call USER_EXIT_CRITICAL once for each USER_ENTER_CRITICAL call to release the critical section.

Parameters:

None

Returns:

Nothing ---This is a void function

See Also:

USER_ENTER_CRITICAL --- Sets a level 7 interrupt mask when entered

2.39. OSDumpTCBStacks

Synopsis:

```
void OSDumpTCBStacks( void );
```

Description:

This function dumps information about the UCOS stacks and tasks to Stdout. This function is useful for debugging. Note: This function is only valid when UCOS_STACKCHECK is defined.

Parameters:

None

Returns:

Nothing ---This is a void function

Example:

Prio	Stack Ptr	Stack Bottom	Free Now	Min. Free
63	0x20432d4	0x2042f20	237	237
50	0x20451cc	0x2043320	1963	1827
40	0x2028250	0x20262cc	2017	2017
39	0x2020f2c	0x201efcc	2008	2008
38	0x2022f54	0x2020fde	2013	2013
45	0x2024f2c	0x2023020	1987	1987

See Also:

OSDumpTasks --- Dump all of the task info to stdout

2.40. OSDumpTasks

Synopsis:

```
void OSDumpTasks( void );
```

Description:

This function dumps the state and call stack for every task to stdout. This function is useful for debugging. Note: This function is only valid when UCOS_STACKCHECK is defined.

Parameters:

None

Returns:

Nothing --- This is a void function

Example:

Prio	State	Ticks	Call Stack
63	Ready	Forever	At: 02006598
50	Running	----------	02006860->0200a7bc-><END>
40	Timer	63531	----------
39	Fifo	10	02007c98->020046ae-><END>
38	Fifo	Forever	02007c98->02005d54-><END>
45	Semaphore	Forever	02006f16->02009880->0200885a-><END>

See Also:

OSDumpTCBStacks --- Dump all of the task stack information to stdout

2.41. ShowTaskList

Synopsis:

```
void ShowTaskList( void );
```

Description:

This function dumps the current RTOS task states to stdio. The output takes on multiple lines of the following format for each logged state:

```
at t= [T] [Message]
```

Followed by a tally of the number of task states logged since system start:

```
Total messages: [N]
```

`[T]` represents the number of ticks in hexadecimal since system start; `[N]` represents the number of task state messages in decimal logged since system start; `[Message]` represents one of the output messages listed in the below table.

Message	Description
`Wait for Semaphore`	Task is asleep and pending for semaphore
`Wake from Semaphore`	Task gets a semaphore and wakes up
`Task locked`	Task becomes locked
`Task lock++`	Task gets an added nested lock
`Task lock--`	Task gets a nested lock unlocked
`Task unlocked`	Task becomes completely unlocked
`Task priority changed`	The task's priority level is changed
`Unknown flag [F]`	The flag value defining the task's state is undefined
`Switched to Task [P]`	Task priority `[P]` (in decimal) gets control
`Switched to Task [P] PC=[X]`	Task priority `[P]` gets control with the program counter containing the address `[X]` (in hexadecimal) of the instruction being executed

Note: Usage of this function is valid only when defining UCOS_TASKLIST in debug mode. In order to enable this macro definition, it must be uncommented in \Nburn\include\predef.h, followed by rebuilding the system files to incorporate the modification. Attempting to load a compiled non-debug application image with the macro defined will cause a trap error.

Parameter:
None

Returns:
None

2.42. OSFlagCreate

Synopsis:

```
void OSFlagCreate( OS_FLAGS *pf )
```

Description:

This function initializes an OS_FLAGS object that has already been declared. This function must be called before you can use an OS_FLAGS object.

Parameter:

Type	Name	Description
OS_FLAGS	*pf	A pointer to the location of the object to be initialized.

Returns:

Nothing --- This is a void function.

Example:

```
OS_FLAGS test_flag;            // Declare an OS_FLAGS object
OSFlagCreate( &test_flag );    // Initialize the object
```

2.43. OSFlagSet

Synopsis:

```
void OSFlagSet( OS_FLAGS *flags, DWORD bits_to_set )
```

Description:

This function sets the corresponding bits asserted in `bits_to_set` of an OS_FLAGS object pointed to by `*flags`.

Parameters:

Type	Name	Description
OS_FLAGS	`*flags`	A pointer to the OS_FLAGS object to be configured.
DWORD	`bits_to_set`	A bit or set of bits to be set.

Returns:

Nothing --- This is a void function.

Example:

```
OSFlagSet( &test_flag, 0x000000F0 );    // Set bits 4-7 of OS_FLAG
                                        // object "test_flag"
```

2.44. OSFlagState

Synopsis:

```
DWORD OSFlagState( OS_FLAGS *flags )
```

Description:

This function returns the current values of the flags stored in the OS_FLAGS object structure.

Parameter:

Type	Name	Description
OS_FLAGS	*flags	A pointer to the OS_FLAGS object whose flag states are to be returned.

Returns:

The flag states of the OS_FLAGS object.

Example:

```
DWORD uint32_flags = OSFlagState( &test_flag );

if ( uint32_flags & 0x00000080 )
{
   iprintf( "Flag bit 7 is set.\r\n" );
}
else
{
   iprintf( "Flag bit 7 is clear.\r\n" );
}
```

2.45. OSFlagClear

Synopsis:

```
void OSFlagClear( OS_FLAGS *flags, DWORD bits_to_clr )
```

Description:

This function clears the bits asserted in `bits_to_clr` of an OS_FLAGS object pointed to by `*flags`.

Parameters:

Type	Name	Description
OS_FLAGS	*flags	A pointer to the OS_FLAGS object to be configured.
DWORD	bits_to_clr	A bit or set of bits to be cleared.

Returns:

Nothing --- This is a void function.

Example:

```
OSFlagClear( &test_flag, 0x000000F0 );    // Clear bits 4-7 of OS_FLAG
                                          // object "test_flag"
```

2.46. OSFlagPendAll

Synopsis:

```
BYTE OSFlagPendAll( OS_FLAGS *flags, DWORD bit_mask, WORD timeout )
```

Description:

This function waits a number of time ticks specified by `timeout` until all the flags indicated by `bit_mask` are set.

Parameters:

Type	Name	Description
OS_FLAGS	`*flags`	A pointer to the OS_FLAGS object with the desired flag bits.
DWORD	`bit_mask`	A bit or set of bets to wait on.
WORD	`timeout`	Number of time ticks to wait on all specified flag bits to be set.

Returns:

OS_NO_ERR (0) --- All the flags indicated by `bit_mask` are set before `timeout` expires.
OS_TIMEOUT (10) --- `timeout` expired.

Example:

```
if ( OSFlagPendAll ( &test_flag, 0x10001000, 20 ) != OS_NO_ERR )
{
   iprintf( "Flag bits 15 and 31 were not set after 20 ticks.\r\n" );
}
else
{
   iprintf( "Both flag bits are set.\r\n" );
}
```

2.47. OSFlagPendAllNoWait

Synopsis:

```
BYTE OSFlagPendAllNoWait( OS_FLAGS *flags, DWORD bit_mask )
```

Description:

This function immediately checks to see if all the flag bits indicated by `bit_mask` are set; it does not wait.

Parameters:

Type	Name	Description
OS_FLAGS	`*flags`	A pointer to the OS_FLAGS object with the desired flag bits.
DWORD	`bit_mask`	A bit or set of bits to check on.

Returns:

OS_NO_ERR (0) --- All flags indicated by `bit_mask` are set.
OS_TIMEOUT (10) --- None or not all of the flags indicated by `bit_mask` are set.

Example:

```
if ( OSFlagPendAllNoWait( &test_flag, 0xFFFFFFFF ) != OS_NO_ERR )
{
   iprintf( "Not all of the flag bits are set.\r\n" );
}
else
{
   iprintf( "All 32 of the flag bits are set.\r\n" );
}
```

2.48. OSFlagPendAny

Synopsis:

```
BYTE OSFlagPendAny( OS_FLAGS *flags, DWORD bit_mask, WORD timeout )
```

Description:

This function waits a number of time ticks specified by `timeout` until any of the flags indicated by `bit_mask` are set.

Parameters:

Type	Name	Description
OS_FLAGS	`*flags`	A pointer to the OS_FLAGS object with the desired flag bits.
DWORD	`bit_mask`	A bit or set of bits to wait on.
WORD	`timeout`	Number of time ticks to wait on any specified flag bits to be set.

Returns:

OS_NO_ERR (0) --- At least one of the flag bits are set before `timeout` expires.
OS_TIMEOUT (10) --- None of the flag bits are set before `timeout` expires.

Example:

```
if ( OSFlagPendAny( &test_flag, 0xFFFFFFFF, 20 ) != OS_NO_ERR )
{
   iprintf( "None of the flag bits are set before time expired.\r\n" );
}
else
{
   iprintf( "At least one of the 32 desired flag bits are set.\r\n" );
}
```

2.49. OSFlagPendAnyNoWait

Synopsis:

```
BYTE OSFlagPendAnyNoWait( OS_FLAGS *flags, DWORD bit_mask )
```

Description:

This function immediately checks to see if any of the flag bits indicated by `bit_mask` are set; it does not wait.

Parameters:

Type	Name	Description
OS_FLAGS	`*flags`	A pointer to the OS_FLAGS object with the desired flag bits.
DWORD	`bit_mask`	A bit or set of bits to check on.

Returns:

OS_NO_ERR (0) --- At least one of the flags indicated by `bit_mask` are set.
OS_TIMEOUT (10) --- None of the flags indicated by `bit_mask` are set.

Example:

```
if ( OSFlagPendAnyNoWait( &test_flag, 0x80010402 ) != OS_NO_ERR )
{
   iprintf( "Bits 1, 10, 16 and 31 are not set.\r\n" );
}
else
{
   iprintf( "At least one of the designated bits are set.\r\n" );
}
```