NOTE: this is an **ALPHA** document currently. There may be errors as well as omissions in this document. Your feedback is welcome and appreciated.

Table of Contents

Function Index

cyg_flag_destroy	destroy (invalidate) a flag
cyg_flag_setbits	set bits (conditions) in a flag
cyg_flag_maskbits	clear conditions (bits) in a flag
cyg_flag_wait	wait forever on a flag
cyg_flag_timed_wait	wait on a flag until timeout
cyg_flag_poll	test for pattern match but do not block
cyg_flag_peek	returns bits (conditions) currently set in a flag
cyg_flag_waiting	check to see if threads wait on a given flag
cyg_interrupt_create	create an interrupt handler
cyg_interrupt_delete	delete an interrupt handler
cyg_interrupt_attach	attach an interrupt vector
cyg_interrupt_detach	detach an interrupt
cyg_interrupt_get_vsr	get VSR pointer of an interrupt
cyg_interrupt_set_vsr	set the VSR of an interrupt
cyg_interrupt_disable	disable all interrupts
cyg_interrupt_enable	re-enable interrupts
cyg_interrupt_mask	mask a single interrupt vector
cyg_interrupt_mask_intunsafe	mask interrupt, not interrupt safe
cyg_interrupt_unmask	unmask an interrupt
cyg_interrupt_unmask_intunsafe	unmask an interrupt, interrupt unsafe
cyg_interrupt_acknowledge	acknowledge an interrupt
cyg_interrupt_configure	configure an interrupt
cyg_interrupt_set_cpu	set a CPU
cyg_interrupt_get_cpu	get CPU
cyg_mbox_create	create an mbox
cyg_mbox_delete	delete an mbox
cyg_mbox_get	get a pointer from an mbox
cyg_mbox_timed_get	get a pointer from an mbox with timeout
cyg_mbox_tryget	get a pointer from a mbox with no block
cyg_mbox_peek_item	get a pointer from an mbox without removing it
cyg_mbox_put	place a pointer in an mbox
cyg_mbox_timed_put	place a message into an mbox with timeout

cyg_mbox_tryput	place a message in an mbox with no blocking
cyg_mbox_peek	get number of messages in mbox
cyg_mbox_waiting_to_get	check to see if threads wait to read from an mbox
cyg_mbox_waiting_to_put	check to see if thread waits to write to an mbox
cyg_mempool_fix_create	create a fixed sized memory pool
cyg_mempool_fix_delete	delete a fixed sized memory pool
cyg_mempool_fix_alloc	allocate a fixed sized block of memory with no timeout
cyg_mempool_fix_timed_alloc	allocate a fixed sized block of memory with timeout
cyg_mempool_fix_try_alloc	allocate a fixed sized block of memory, don't block
cyg_mempool_fix_free	free a block of memory allocated from a fixed sized pool
cyg_mempool_fix_waiting	check to see if threads are waiting to allocate
cyg_mempool_fix_get_info	get info on a fixed sized mempool
cyg_mempool_var_create	create a variable sized memory pool
cyg_mempool_var_delete	delete a variable sized memory pool
cyg_mempool_var_alloc	allocate a variable size of memory with no timeout
cyg_mempool_var_timed_alloc	allocate a variable size of memory with timeout
cyg_mempool_var_try_alloc	allocate a variable size of memory, don't block
cyg_mempool_var_free	free a block of memory allocated from a variable sized pool
cyg_mempool_var_waiting	check to see if threads are waiting to allocate
cyg_mempool_var_get_info	get info on a variable sized mempool
cyg_mutex_init	initialize a mutex
cyg_mutex_destroy	destroy (invalidate) a mutex
cyg_mutex_lock	lock a mutex or wait to lock one
cyg_mutex_trylock	attempt to lock a mutex
cyg_mutex_unlock	unlocks a mutex
cyg_mutex_release	release all threads waiting on a mutex
cyg_mutex_set_ceiling	set ceiling priority of mutex
cyg_mutex_set_protocol	set the protocol of a mutex

cyg_scheduler_start	start scheduler
cyg_scheduler_lock	lock scheduler
cyg_scheduler_safe_lock	lock the scheduler if it's not already locked
cyg_scheduler_unlock	unlock the scheduler
cyg_scheduler_read_lock	read scheduler lock count
cyg_semaphore_init	initialize a counting semaphore
cyg_semaphore_destroy	destroy (invalidate) a semaphore
cyg_semaphore_wait	wait on a counting semaphore
cyg_semaphore_timed_wait	wait on a semaphore with timeout
cyg_semaphore_trywait	get a semaphore if available
cyg_semaphore_post	increment semaphore count
cyg_semaphore_peek	get current semaphore count
cyg_thread_create	create a new thread
cyg_thread_exit	exit a thread
cyg_thread_delete	delete a thread
cyg_thread_suspend	suspend a thread
cyg_thread_resume	resume a suspended thread
cyg_thread_kill	kill a thread
cyg_thread_release	release a thread from a wait
cyg_thread_yield	yield the thread to another thread of equal priority
cyg_thread_self	get calling thread's thread ID
cyg_thread_idle_thread	get the idle thread's thread ID
cyg_thread_set_priority	set the priority of a thread
cyg_thread_get_priority	get the priority of a thread
cyg_thread_get_current_priority	get current priority of a thread
cyg_thread_delay	delay the calling thread for a number of ticks
cyg_thread_get_stack_base	get a thread's stack base address
cyg_thread_get_stack_size	get a thread's stack size
cyg_thread_measure_stack_usage	get a thread's current stack usage
cyg_thread_new_data_index	get a free data index for all threads
cyg_thread_free_data_index	free a data index for all threads
cyg_thread_get_data	read per thread data from a given index

cyg_thread_get_data_ptr	get a data pointer to per thread data
cyg_thread_set_data	set per thread data at a given index
cyg_thread_add_destructor	add a destructor
cyg_thread_rem_destructor	remove (disable) a destructor

---

cyg_alarm_create

Name: ***cyg_alarm_create*** **( )** - create an alarm

Synopsis:

```
void cyg_alarm_create
(
  cyg_handle_t   counter,  /* counter to attach to alarm      */
  cyg_alarm_t    *alarmfn, /* alarm call back function        */
  cyg_addrword_t data,     /* data to be passed to callback   */
  cyg_handle_t   *handle,  /* returned handle to alarm object */
  cyg_alarm      *alarm    /* alarm object                    */
)
```

Description: This creates a new alarm. Alarms are periodic events, generally tied to the system counter. The period is defined when cyg_alarm_initialize is called.

When the alarm expires "*alarmfn" is called with "data" as an argument. Alarms can be setup to be recurring or to execute only once.

The callback "alarmfn" is of the form: void cyg_alarm_fn(Cyg_Alarm *alarm, CYG_ADDRWORD data).

The newly created alarm is written to "*handle".

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_alarm_delete, cyg_alarm_initialize, cyg_alarm_get_times, cyg_alarm_enable, cyg_alarm_disable

---

cyg_alarm_delete

Name: ***cyg_alarm_delete*** **( )** - delete alarm

Synopsis:

```
void cyg_alarm_delete
(
  cyg_handle_t alarm /* alarm to delete */
)
```

Description: This function deletes an alarm from the system and invalidates the handle to the alarm. The alarm cannot be used once it is deleted.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_alarm_create, cyg_alarm_disable

---

cyg_alarm_initialize

Name: ***cyg_alarm_initialize*** **( )** - initialize (start) an alarm

Synopsis:

```
void cyg_alarm_initialize
(
  cyg_handle_t     alarm,   /* handle of alarm to initialize */
  cyg_tick_count_t trigger, /* absolute trigger time         */
  cyg_tick_count_t interval /* re-trigger interval           */
)
```

Description: This initializes an alarm to trigger at the absolute time of "trigger". The trigger time is an absolute time. You can get the current trigger time of a clock by calling cyg_counter_current_value on the counter.

If the alarm is to be triggered at a regular interval, "interval" can be set to a non 0 value.

When the trigger fires the alarm function associated with the alarm will be called.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_alarm_initialize, cyg_alarm_get_times, cyg_alarm_enable, cyg_alarm_disable

---

cyg_alarm_get_times

Name: ***cyg_alarm_get_times*** **( )** - get alarm times

Synopsis:

```
void cyg_alarm_get_times
(
  cyg_handle_t     alarm,    /* alarm to get the times of   */
  cyg_tick_count_t *trigger, /* next trigger time           */
  cyg_tick_count_t *interval /* current re-trigger interval */
)
```

Description: This function will return the next absolute trigger time for the alarm and its re-trigger interval. If either of the parameters are not needed, you can safely pass NULL instead of an actual pointer.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing.

See Also: cyg_alarm_initialize, cyg_alarm_enable, cyg_alarm_disable

cyg_alarm_enable

Name: ***cyg_alarm_enable*** **( )** - re-enable an alarm

Synopsis:

```
void cyg_alarm_enable
(
  cyg_handle_t alarm /* alarm to re-enable */
)
```

Description: This re-enables an alarm that has previously been disabled by a call to cyg_alarm_disable. This is most often used with a periodic alarm.

A periodic alarm that has been disabled and later re-enabled will fire at the same intervals it did previously. For example, a periodic alarm that fired every 10 seconds at time T0, T10, T20, T30... etc that was disabled for 15 seconds at time T31 and then re-enabled would then start firing again at T50, T60, T70 etc.

You can call cyg_alarm_initialize if you want to reset the periodic interval.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_alarm_initialize, cyg_alarm_get_times, cyg_alarm_disable

---

cyg_alarm_disable

Name: ***cyg_alarm_disable*** **( )** - disable an alarm

Synopsis:

```
void cyg_alarm_disable
(
  cyg_handle_t alarm /* alarm to disable */
)
```

Description: This disables an alarm. This is most often used with a periodic alarm.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_alarm_initialize, cyg_alarm_get_times, cyg_alarm_enable

---

cyg_clock_create

Name: ***cyg_clock_create*** **( )** - create a clock

Synopsis:

```
void cyg_clock_create
(
  cyg_resolution_t resolution, /* resolution     */
  cyg_handle_t     *handle,    /* created handle */
  cyg_clock        *clock      /* clock object   */
)
```

Description: This creates a new clock with a given resolution. A clock is nothing more than a counter with an associated resolution. It is assumed that the underlying counter of any clock has a source of regular ticks. A clock can be converted to a counter, but a counter cannot necessarily be converted to a clock.

This function does not return the newly created clock directly, it returns it through a pointer to "handle".

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_clock_delete

---

cyg_clock_delete

Name: ***cyg_clock_delete*** **( )** - delete a clock

Synopsis:

```
void cyg_clock_delete
(
  cyg_handle_t clock /* clock to delete */
)
```

Description: This deletes a clock. Be sure that no other parts of the system are using the clock when you call this function.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_clock_delete

cyg_clock_to_counter

Name: ***cyg_clock_to_counter*** **( )** - converts a clock to a counter

Synopsis:
```
void cyg_clock_to_counter
(
  cyg_handle_t clock,   /* clock to convert          */
  cyg_handle_t *counter /* address of counter object */
)
```

Description: This converts a clock to a counter. A clock is nothing more than a counter with an associated resolution. The counter is not returned directly but through a pointer.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_real_time_clock, cyg_current_time

---

cyg_clock_set_resolution

Name: ***cyg_clock_set_resolution*** **( )** - set clock resolution

Synopsis:
```
void cyg_clock_set_resolution
(
  cyg_handle_t     clock,     /* clock                          */
  cyg_resolution_t resolution /* new resolution to set clock */
)
```

Description: This sets the resolution of the given clock. The resolution is described as struct {cyg_uint32 dividend; cyg_uint32 divisor;}.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_clock_get_resolution, cyg_current_time

---

cyg_clock_get_resolution

Name: ***cyg_clock_get_resolution*** **( )** - get resolution of a clock

Synopsis:
```
cyg_resolution_t cyg_clock_get_resolution
(
  cyg_handle_t clock /* clock to get resolution of */
)
```

Description: This gets the resolution of the given clock. The resolution is described as struct {cyg_uint32 dividend; cyg_uint32 divisor;}.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the resolution of the specified clock.

See Also: cyg_clock_set_resolution, cyg_current_time

---

cyg_real_time_clock

Name: ***cyg_real_time_clock*** **( )** - get the real time system clock

Synopsis:

```
cyg_handle_t cyg_real_time_clock
(
  void
)
```

Description: This gets the system's real time clock. The real time clock is used for system delays, blocking waits, etc.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the system's real time clock.

See Also: cyg_clock_set_resolution, cyg_clock_get_resolution, cyg_current_time

---

cyg_current_time

Name: ***cyg_current_time*** **( )** - get the current system time

Synopsis:

```
cyg_tick_count_t cyg_current_time
(
  void
)
```

Description: This gets the current system time in ticks. The system time is represented as a 64 bit number. Since the system time is a 64 bit number, there is no danger of overflow since a tick every 1ns would not roll over for over 500 years.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the current system time.

See Also: cyg_clock_set_resolution, cyg_clock_get_resolution, cyg_real_time_clock

---

---

cyg_cond_init

Name: ***cyg_cond_init*** **( )** - initialize a condition variable

Synopsis:

```
void cyg_cond_init
(
  cyg_cond_t  *cond, /* condition variable to initialize */
  cyg_mutex_t *mutex /* associated mutex                  */
)
```

Description: This initializes a condition variable for use. Condition variables are a synchronization mechanism which allows one thread to signal multiple threads simultaneously.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing.

See Also: cyg_cond_destroy

---

cyg_cond_destroy

Name: ***cyg_cond_destroy*** **( )** - destroy (invalidate) a condition variable

Synopsis:

```
void cyg_cond_destroy
(
  cyg_cond_t *cond /* condition variable to destroy (invalidate) */
)
```

Description: This destroys (invalidates) a condition variable. Be careful not to destroy a condition variable that other threads are waiting on or is otherwise in use. If you destroy a condition variable that is in use, you will risk deadlocking the system.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_cond_init

---

cyg_cond_wait

Name: ***cyg_cond_wait*** **( )** - wait on a condition variable

Synopsis:
```
cyg_bool_t cyg_cond_wait
(
  cyg_cond_t *cond /* condition variable to wait for */
)
```

Description: Wait on a condition variable.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if no error, "false" otherwise.

See Also: cyg_cond_signal, cyg_cond_broadcast, cyg_cond_timed_wait

---

cyg_cond_signal

Name: ***cyg_cond_signal*** **( )** - wake one thread waiting on a condition variable

Synopsis:
```
void cyg_cond_signal
(
  cyg_cond_t *cond /* condition variable to signal */
)
```

Description: This wakes a single thread waiting on a condition variable. If multiple threads are waiting on the condition variable the scheduler implementation determines which thread will wake first. Generally it is the thread with the highest priority.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_cond_wait, cyg_cond_broadcast, cyg_cond_timed_wait

---

cyg_cond_broadcast

Name: ***cyg_cond_broadcast*** **( )** - wake all threads waiting on a condition variable

Synopsis:
```
void cyg_cond_broadcast
(
  cyg_cond_t *cond /* condition variable to signal */
)
```

Description: This wakes all threads waiting on a condition variable.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_cond_wait, cyg_cond_signal, cyg_cond_timed_wait

---

cyg_cond_timed_wait

Name: ***cyg_cond_timed_wait*** **( )** - wake one thread on a condition variable with timeout

Synopsis:

```
cyg_bool_t cyg_cond_timed_wait
(
  cyg_cond_t       *cond,  /* condition variable to wait for */
  cyg_tick_count_t abstime /* absolute timeout               */
)
```

Description: This waits on a condition variable. If the system time goes beyond "abstime" the wait will timeout and an error is returned. You can get the current system time by calling cyg_current_time.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if no timeout, "false" otherwise

See Also: cyg_cond_wait, cyg_cond_signal, cyg_cond_broadcast

---

cyg_counter_create

Name: ***cyg_counter_create*** **( )** - create a new counter

Synopsis:

```
void cyg_counter_create
(
  cyg_handle_t *handle, /* returned counter handle */
  cyg_counter  *counter /* counter object          */
)
```

Description: This creates a new counter. The new counter is returned through "handle". Counters have to be externally incremented before they can advance.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing.

See Also: cyg_counter_delete, cyg_counter_current_value, cyg_counter_set_value, cyg_counter_tick, cyg_counter_multi_tick

---

cyg_counter_delete

Name: ***cyg_counter_delete*** **( )** - delete a counter

Synopsis:

```
void cyg_counter_delete
(
  cyg_handle_t counter /* counter to delete */
)
```

Description: This deletes a counter. Be sure not to continue incrementing the counter once it's been deleted.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_counter_create

---

cyg_counter_current_value

Name: ***cyg_counter_current_value*** **( )** - get current counter value

Synopsis:
```
cyg_tick_count_t cyg_counter_current_value
(
  cyg_handle_t counter /* counter to get the value of */
)
```

Description: This gets the specified counter's current value

Include: #include <**cyg/kernel/kapi.h**>

Returns: the number of ticks that has elapsed for this counter.

See Also: cyg_counter_set_value, cyg_counter_tick, cyg_counter_multi_tick

---

cyg_counter_set_value

Name: ***cyg_counter_set_value*** **( )** - set the value of the counter

Synopsis:
```
void cyg_counter_set_value
(
  cyg_handle_t     counter,  /* counter to set       */
  cyg_tick_count_t new_value /* new value of counter */
)
```

Description: This sets the value of the specified counter to a new value.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_counter_current_value, cyg_counter_tick, cyg_counter_multi_tick

---

cyg_counter_tick

Name: ***cyg_counter_tick*** **( )** - increment a counter by a single tick

Synopsis:
```
void cyg_counter_tick
(
  cyg_handle_t counter /* counter to advance */
)
```

Description: This increments a counter by a single tick.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_counter_current_value, cyg_counter_set_value, cyg_counter_multi_tick

---

cyg_counter_multi_tick

Name: ***cyg_counter_multi_tick*** **( )** - advance a counter by multiple ticks

Synopsis:

```
void cyg_counter_multi_tick
(
  cyg_handle_t     counter, /* counter to advance         */
  cyg_tick_count_t ticks    /* number of ticks to advance */
)
```

Description: This increments a counter by a multiple number of ticks.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_counter_current_value, cyg_counter_set_value, cyg_counter_tick

---

---

cyg_exception_set_handler

Name: ***cyg_exception_set_handler*** **( )** - create a new exception handler

Synopsis:

```
void cyg_exception_set_handler
(
  cyg_code_t              exception_number, /* exception number          */
  cyg_exception_handler_t *new_handler,     /* pointer to new handler    */
  cyg_addrword_t          new_data,         /* new handler data argument */
  cyg_exception_handler_t **old_handler,    /* receives old handler      */
  cyg_addrword_t          *old_data         /* receives old data         */
)
```

Description: This creates a new exception handler and retrieves the old one at the same time. This is highly architecture dependent. The exception handler has the following prototype: void cyg_exception_handler_t (cyg_addrword_t data, cyg_code_t exception_number, cyg_addrword_t info).

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_exception_clear_handler, cyg_exception_call_handler

---

cyg_exception_clear_handler

Name: ***cyg_exception_clear_handler*** **( )** - remove an exception handler

Synopsis:

```
void cyg_exception_clear_handler
(
  cyg_code_t exception_number /* exception handler to remove */
)
```

Description: This removes an exception handler from the system, i.e. "clears" the exception handler to the default handler.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_exception_set_handler, cyg_exception_call_handler

---

cyg_exception_call_handler

Name: ***cyg_exception_call_handler*** **( )** - invoke an exception handler

Synopsis:

```
void cyg_exception_call_handler
(
  cyg_handle_t   thread,           /* thread ID        */
  cyg_code_t     exception_number, /* exception number */
  cyg_addrword_t error_code        /* error code       */
)
```

Description: This invokes an exception handler with "error_code" being the info (the 3rd argument) of the exception handler.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_exception_set_handler, cyg_exception_clear_handler

---

---

cyg_flag_init

Name: ***cyg_flag_init*** **( )** - initialize a flag for use

Synopsis:

```
void cyg_flag_init
(
  cyg_flag_t *flag /* flag to initialize */
)
```

Description: This initializes a flag for use. Flags are synchronization mechanism that allows threads to wait on a condition or a set of conditions. Each condition is represented as a bit. Bits are user defined.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_flag_destroy

---

cyg_flag_destroy

Name: ***cyg_flag_destroy*** **( )** - destroy (invalidate) a flag

Synopsis:

```
void cyg_flag_destroy
(
  cyg_flag_t *flag /* flag to destroy (invalidate) */
)
```

Description: This destroys or invalidates a flag. Be certain that no threads are waiting on or otherwise using a flag when you call this function or you may deadlock the system.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_flag_init

---

cyg_flag_setbits

Name: ***cyg_flag_setbits*** **( )** - set bits (conditions) in a flag

Synopsis:

```
void cyg_flag_setbits
(
  cyg_flag_t       *flag, /* flag to modify */
  cyg_flag_value_t value  /* bits to set    */
)
```

Description: This sets bits (conditions) to true in a flag. Any bit in "value" that is set to true (1) will set the equivalent bit in the flag. This may wake threads waiting on this flag as a result.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_flag_maskbits, cyg_flag_wait, cyg_flag_timed_wait, cyg_flag_poll, cyg_flag_peek, cyg_flag_waiting

---

cyg_flag_maskbits

Name: ***cyg_flag_maskbits*** **( )** - clear conditions (bits) in a flag

Synopsis:

```
void cyg_flag_maskbits
(
  cyg_flag_t       *flag, /* flag to modify */
  cyg_flag_value_t value  /* bits to clear  */
)
```

Description: This clears bits (conditions) in a flag. Any bit that is set to false (0) in "value" will be subsequently cleared in the flag. If "value" is set to 0, all conditions will be cleared, if "value" is set to all ones, no conditions will be cleared. Since this just clears conditions, no thread will run as a result of a call to this function.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_flag_setbits, cyg_flag_wait, cyg_flag_timed_wait, cyg_flag_poll, cyg_flag_peek, cyg_flag_waiting

---

cyg_flag_wait

Name: ***cyg_flag_wait*** **( )** - wait forever on a flag

Synopsis:

```
cyg_flag_value_t cyg_flag_wait
(
  cyg_flag_t       *flag,   /* flag to wait on     */
  cyg_flag_value_t pattern, /* pattern to wait for */
  cyg_flag_mode_t  mode     /* mode of waiting     */
)
```

Description: This causes the calling thread to wait on a set of bits (conditions) to be set in a given flag. The "mode" indicates how the pattern will be interpreted:

CYG_FLAG_WAITMODE_AND - return match if all conditions in the pattern are set in the flag

CYG_FLAG_WAITMODE_OR - return match if any of the conditions in the pattern are set in the flag.

CYG_FLAG_WAITMODE_CLR - automatically clear the conditions that caused the calling thread to return a match, IF there was a match.

CYG_FLAG_WAITMODE_CLR can be combined with CYG_FLAG_WAITMODE_AND or CYG_FLAG_WAITMODE_OR to clear the bits that caused the condition to be met by oring the bitfields together.

If the conditions are met, the pattern that caused the pattern match is returned. A value of 0 will be returned if the thread was awakened for another reason other than a pattern match or a bad value was specified as the mode.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the pattern that caused a match or 0 if an error.

See Also: cyg_flag_setbits, cyg_flag_maskbits, cyg_flag_timed_wait, cyg_flag_poll, cyg_flag_peek, cyg_flag_waiting

---

cyg_flag_timed_wait

Name: ***cyg_flag_timed_wait*** **( )** - wait on a flag until timeout

Synopsis:

```
cyg_flag_value_t cyg_flag_timed_wait
(
  cyg_flag_t       *flag,   /* flag to wait on        */
  cyg_flag_value_t pattern, /* pattern to wait for    */
  cyg_flag_mode_t  mode     /* mode of waiting        */
  cyg_tick_count_t abstime  /* absolute timeout value */
)
```

Description: This causes the calling thread to wait on a set of bits (conditions) to be set in a given flag. If the system clock goes beyond "abstime" the wait will timeout and an error will be returned. The "mode" indicates how the pattern will be interpreted:

CYG_FLAG_WAITMODE_AND - return match if all conditions in the pattern are set in the flag

CYG_FLAG_WAITMODE_OR - return match if any of the conditions in the pattern are set in the flag.

CYG_FLAG_WAITMODE_CLR - automatically clear the conditions that caused the calling thread to return a match, IF there was a match.

CYG_FLAG_WAITMODE_CLR can be combined with CYG_FLAG_WAITMODE_AND or CYG_FLAG_WAITMODE_OR to clear the bits that caused the condition to be met by oring the bitfields together.

If the conditions are met, the pattern that caused the pattern match is returned. A value of 0 will be returned if the thread timed out, was awakened for another reason other than a pattern match or a bad value was specified as the mode.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the pattern that caused a match or 0 if an error or timeout.

See Also: cyg_flag_setbits, cyg_flag_maskbits, cyg_flag_wait, cyg_flag_poll, cyg_flag_peek, cyg_flag_waiting

---

cyg_flag_poll

Name: ***cyg_flag_poll*** **( )** - test for pattern match but do not block

Synopsis:

```
cyg_flag_value_t cyg_flag_poll
(
  cyg_flag_t       *flag,   /* flag to wait on     */
  cyg_flag_value_t pattern, /* pattern to wait for */
  cyg_flag_mode_t  mode     /* mode of waiting     */
)
```

Description: This causes the calling thread to check if a set of bits (conditions) have been set in a given flag. The "mode" indicates how the pattern will be interpreted:

CYG_FLAG_WAITMODE_AND - return match if all conditions in the pattern are set in the flag

CYG_FLAG_WAITMODE_OR - return match if any of the conditions in the pattern are set in the flag.

CYG_FLAG_WAITMODE_CLR - automatically clear the conditions that caused the calling thread to return a match, IF there was a match.

CYG_FLAG_WAITMODE_CLR can be combined with CYG_FLAG_WAITMODE_AND or CYG_FLAG_WAITMODE_OR to clear the bits that caused the condition to be met by oring the bitfields together.

If the conditions are met, the pattern that caused the pattern match is returned. A value of 0 will be returned if the thread timed out, was awakened for another reason other than a pattern match or a bad value was specified as the mode.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the pattern that caused a match or 0 if there was no match.

See Also: cyg_flag_setbits, cyg_flag_maskbits, cyg_flag_wait, cyg_flag_timed_wait, cyg_flag_peek, cyg_flag_waiting

---

cyg_flag_peek

Name: ***cyg_flag_peek*** **( )** - returns bits (conditions) currently set in a flag

Synopsis:

```
cyg_flag_value_t cyg_flag_peek
(
  cyg_flag_t *flag /* flag to peek at */
)
```

Description: This returns the current bits (conditions) that are set in a given flag.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the bits (conditions) as a bitmask that have been set in the flag.

See Also: cyg_flag_setbits, cyg_flag_maskbits, cyg_flag_wait, cyg_flag_timed_wait, cyg_flag_poll, cyg_flag_waiting

---

cyg_flag_waiting

Name: ***cyg_flag_waiting*** **( )** - check to see if threads wait on a given flag

Synopsis:

```
cyg_bool_t cyg_flag_waiting
(
  cyg_flag_t *flag /* flag to check */
)
```

Description: This reports whether any threads are currently being blocked waiting for bits (conditions) to be set in the given flag.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if threads are being blocked, "false" otherwise.

See Also: cyg_flag_setbits, cyg_flag_maskbits, cyg_flag_wait, cyg_flag_timed_wait, cyg_flag_poll, cyg_flag_peek

---

cyg_interrupt_create

Name: ***cyg_interrupt_create*** **( )** - create an interrupt handler

Synopsis:

```
void cyg_interrupt_create
(
  cyg_vector_t   vector,   /* interrupt vector              */
  cyg_priority_t priority, /* priority of interrupt         */
  cyg_addrword_t data,     /* data pointer                  */
  cyg_ISR_t      *isr,     /* interrupt service routine     */
  cyg_DSR_t      *dsr,     /* deferred service routine      */
  cyg_handle_t   *handle,  /* returned handle to interrupt */
  cyg_interrupt  *intr     /* put interrupt here            */
)
```

Description: This creates a new interrupt handler. Interrupts are highly architecture dependent. The queue priority is used only in the case that interrupts are chained. Interrupts need to be attached before they will be called by the system.

The "isr" has the prototype of cyg_uint32 cyg_ISR(cyg_vector vector, CYG_ADDRWORD data). The ISR is called from the VSR. The VSR is usually implemented by eCos itself. If the ISR returns CYG_ISR_HANDLED the DSR will NOT be called, if the ISR returns CYG_ISR_CALL_DSR the DSR is called.

The "dsr" has the prototype of void cyg_DSR(cyg_vector vector, cyg_ucount32 count, CYG_ADDRWORD data). The DSR returns nothing.

ISR's cannot access the vast majority of kernel routines. The DSR can access more routines. What can and cannot be called safely from these routines I have not found in the documentation yet.

The "handle" returns a handle to the newly created handler. The "intr" argument is an interrupt object that is used for memory storage.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_delete, cyg_interrupt_attach, cyg_interrupt_detach, cyg_interrupt_get_vsr, cyg_interrupt_set_vsr, cyg_interrupt_configure

cyg_interrupt_delete

Name: ***cyg_interrupt_delete*** **( )** - delete an interrupt handler

Synopsis:

```
void cyg_interrupt_delete
(
  cyg_handle_t interrupt /* interrupt to delete */
)
```

Description: This removes an interrupt handler from the system

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_create, cyg_interrupt_attach, cyg_interrupt_detach

---

cyg_interrupt_attach

Name: ***cyg_interrupt_attach*** **( )** - attach an interrupt vector

Synopsis:

```
void cyg_interrupt_attach
(
  cyg_handle_t interrupt /* interrupt to attach */
)
```

Description: This attaches an interrupt to the physical layer. An interrupt cannot be called by the hardware until it's attached.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_create, cyg_interrupt_delete, cyg_interrupt_detach

---

cyg_interrupt_detach

Name: ***cyg_interrupt_detach*** **( )** - detach an interrupt

Synopsis:

```
void cyg_interrupt_detach
(
  cyg_handle_t interrupt /* interrupt to detach */
)
```

Description: This detaches an interrupt from the physical layer.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_create, cyg_interrupt_delete, cyg_interrupt_attach

---

cyg_interrupt_get_vsr

Name: ***cyg_interrupt_get_vsr*** **( )** - get VSR pointer of an interrupt

Synopsis:

```
void cyg_interrupt_get_vsr
(
  cyg_vector_t vector, /* vector to get                */
  cyg_VSR_t    **vsr   /* pointer to store vsr pointer */
)
```

Description: This gets an interrupt's associated VSR through the second argument. It is rarely necessary to change or modify the VSR of a system since VSR's are normally setup by the eCos HAL layer.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing.

See Also: cyg_interrupt_create, cyg_interrupt_get_vsr, cyg_interrupt_set_vsr

---

cyg_interrupt_set_vsr

Name: ***cyg_interrupt_set_vsr*** **( )** - set the VSR of an interrupt

Synopsis:

```
void cyg_interrupt_set_vsr
(
  cyg_vector_t vector, /* vector to set      */
  cyg_VSR_t    *vsr    /* pointer to new vsr */
)
```

Description: This sets an interrupt's VSR. It is rarely necessary to change or modify the VSR of a system since VSR's are normally setup by the eCos HAL layer.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing.

See Also: cyg_interrupt_create, cyg_interrupt_delete, cyg_interrupt_get_vsr

---

cyg_interrupt_disable

Name: ***cyg_interrupt_disable*** **( )** - disable all interrupts

Synopsis:

```
void cyg_interrupt_disable
(
  void
)
```

Description: This disables all interrupts in the system. Avoid using this function unless strictly necessary since it will affect interrupt latency. It is better to disable thread context switching. This call can be nested, i.e. every call to cyg_interrupt_disable must be matched with cyg_interrupt_enable to re-enable interrupts.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_enable, cyg_interrupt_mask, cyg_interrupt_mask_intunsafe, cyg_interrupt_unmask, cyg_interrupt_unmask_intunsafe

---

cyg_interrupt_enable

Name: ***cyg_interrupt_enable*** **( )** - re-enable interrupts

Synopsis:

```
void cyg_interrupt_enable
(
  void
)
```

Description: This is the complement to cyg_interrupt_disable. For each call to cyg_interrupt_disable, there must be a matching call to cyg_interrupt_enable to re-enable interrupts. Be cautious in using these functions since they will affect overall system interrupt latency.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_disable, cyg_interrupt_mask, cyg_interrupt_mask_intunsafe, cyg_interrupt_unmask, cyg_interrupt_unmask_intunsafe

---

cyg_interrupt_mask

Name: ***cyg_interrupt_mask*** **( )** - mask a single interrupt vector

Synopsis:

```
void cyg_interrupt_mask
(
  cyg_vector_t vector /* vector to mask */
)
```

Description: This function masks a single interrupt.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_disable, cyg_interrupt_enable, cyg_interrupt_mask_intunsafe, cyg_interrupt_unmask, cyg_interrupt_unmask_intunsafe

cyg_interrupt_mask_intunsafe

Name: ***cyg_interrupt_mask_intunsafe*** **( )** - mask interrupt, not interrupt safe

Synopsis:

```
void cyg_interrupt_mask_intunsafe
(
  cyg_vector_t vector /* vector to mask */
)
```

Description: This function masks a single interrupt. This call is not interrupt safe.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_disable, cyg_interrupt_enable, cyg_interrupt_mask, cyg_interrupt_unmask, cyg_interrupt_unmask_intunsafe

---

cyg_interrupt_unmask

Name: ***cyg_interrupt_unmask*** **( )** - unmask an interrupt

Synopsis:

```
void cyg_interrupt_unmask
(
  cyg_vector_t vector /* vector to unmask */
)
```

Description: This unmasks an interrupt.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_disable, cyg_interrupt_enable, cyg_interrupt_mask, cyg_interrupt_mask_intunsafe, cyg_interrupt_unmask_intunsafe

---

cyg_interrupt_unmask_intunsafe

Name: ***cyg_interrupt_unmask_intunsafe*** **( )** - unmask an interrupt, interrupt unsafe

Synopsis:

```
void cyg_interrupt_unmask_intunsafe
(
  cyg_vector_t vector /* vector to unmask */
)
```

Description: This masks and interrupt. This call is not interrupt safe.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_disable, cyg_interrupt_enable, cyg_interrupt_mask, cyg_interrupt_mask_intunsafe, cyg_interrupt_unmask

---

cyg_interrupt_acknowledge

Name: ***cyg_interrupt_acknowledge*** **( )** - acknowledge an interrupt

Synopsis:

```
void cyg_interrupt_acknowledge
(
  cyg_vector_t vector /* vector to acknowledge */
)
```

Description: This acknowledges (clears) an interrupt.

Include: #include <**cyg/kernel/kapi.h**>

Returns:

See Also: cyg_interrupt_create, cyg_interrupt_delete, cyg_interrupt_attach, cyg_interrupt_detach

---

cyg_interrupt_configure

Name: ***cyg_interrupt_configure*** **( )** - configure an interrupt

Synopsis:

```
void cyg_interrupt_configure
(
  cyg_vector_t vector, /* vector to configure                */
  cyg_bool_t   level,  /* level or edge triggered            */
  cyg_bool_t   up      /* rising/falling edge, high/low level */
)
```

Description: This configures an interrupt for level or edge triggering as well as rising/falling edge or high/low level.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_create

---

cyg_interrupt_set_cpu

Name: ***cyg_interrupt_set_cpu*** **( )** - set a CPU

Synopsis:

```
void cyg_interrupt_set_cpu
(
  cyg_vector_t vector, /* vector to control */
  cyg_cpu_t    cpu     /* CPU to set        */
)
```

Description: I really have no idea what this does in an SMP sytem.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_interrupt_get_cpu

---

cyg_interrupt_get_cpu

Name: ***cyg_interrupt_get_cpu*** **( )** - get CPU

Synopsis:

```
cyg_cpu_t cyg_interrupt_get_cpu
(
  cyg_vector_t vector /* vector to control */
)
```

Description: I really have no idea what this does in an SMP system

Include: #include <**cyg/kernel/kapi.h**>

Returns: something, apparently the CPU a vector is attached to.

See Also: cyg_interrupt_set_cpu

---

cyg_mbox_create

Name: ***cyg_mbox_create*** **( )** - create an mbox

Synopsis:

```
void cyg_mbox_create
(
  cyg_handle_t *handle, /* returned handle to mbox object */
  cyg_mbox     *mbox    /* mbox object                    */
)
```

Description: This creates an mbox. Mboxes are similar to message queues in other systems but in eCos all mboxes are of the same depth and they only pass void pointers around, nothing larger. The kernel determines the depth of the mboxes.

The mbox can be manipulated by "*handle".

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_mbox_delete

---

cyg_mbox_delete

Name: ***cyg_mbox_delete*** **( )** - delete an mbox

Synopsis:

```
void cyg_mbox_delete
(
  cyg_handle_t mbox /* mbox to delete */
)
```

Description: This deletes an mbox. Be careful not to delete any mboxes that other threads may be waiting on or using or the system may deadlock.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_mbox_create

---

cyg_mbox_get

Name: ***cyg_mbox_get*** **( )** - get a pointer from an mbox

Synopsis:

```
void *cyg_mbox_get
(
  cyg_handle_t mbox /* mbox to read data from */
)
```

Description: This reads a pointer from an mbox. If the mbox has no data in it, this function will block the calling thread until data is available.

Include: #include <**cyg/kernel/kapi.h**>

Returns: a pointer to data that was placed in the mbox.

See Also: cyg_mbox_timed_get, cyg_mbox_tryget, cyg_mbox_peek_item, cyg_mbox_put, cyg_mbox_timed_put, cyg_mbox_tryput, cyg_mbox_peek, cyg_mbox_waiting_to_get, cyg_mbox_waiting_to_put

---

cyg_mbox_timed_get

Name: ***cyg_mbox_timed_get*** **( )** - get a pointer from an mbox with timeout

Synopsis:

```
void *cyg_mbox_timed_get
(
  cyg_handle_t     mbox,    /* mbox to read     */
  cyg_tick_count_t abstime /* absolute timeout */
)
```

Description: This reads data from an mbox. If the mbox has no data in it, this function will block the calling thread until data is available.

The delay is specified as an absolute time of the clock tick. To get a relative time use cyg_current_time to get the current system time and add an offset to that value.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the retrieved pointer or NULL if the mailbox wait timed out.

See Also: cyg_mbox_get, cyg_mbox_tryget, cyg_mbox_peek_item, cyg_mbox_put, cyg_mbox_timed_put, cyg_mbox_tryput, cyg_mbox_peek, cyg_mbox_waiting_to_get, cyg_mbox_waiting_to_put

---

cyg_mbox_tryget

Name: ***cyg_mbox_tryget*** **( )** - get a pointer from a mbox with no block

Synopsis:

```
void *cyg_mbox_tryget
(
  cyg_handle_t mbox /* mailbox to read */
)
```

Description: This reads data from an mbox. If the mbox has no data in it, this function will not block but will return immediately and indicate a failure by returning NULL.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the retrieved pointer or NULL if the mailbox was empty.

See Also: cyg_mbox_get, cyg_mbox_timed_get, cyg_mbox_peek_item, cyg_mbox_put, cyg_mbox_timed_put, cyg_mbox_tryput, cyg_mbox_peek, cyg_mbox_waiting_to_get, cyg_mbox_waiting_to_put

---

cyg_mbox_peek_item

Name: ***cyg_mbox_peek_item*** **( )** - get a pointer from an mbox without removing it

Synopsis:

```
void *cyg_mbox_peek_item
(
  cyg_handle_t mbox /* mailbox to read */
)
```

Description: This reads a pointer from an mbox. If the mbox has no data in it, this function will return immediately. If there is data in the mbox this will return the pointer to the data, but it will not remove the pointer from the mailbox queue.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the retrieved pointer or NULL if the mailbox was empty.

See Also: cyg_mbox_get, cyg_mbox_timed_get, cyg_mbox_tryget, cyg_mbox_put, cyg_mbox_timed_put, cyg_mbox_tryput, cyg_mbox_peek, cyg_mbox_waiting_to_get, cyg_mbox_waiting_to_put

---

cyg_mbox_put

Name: ***cyg_mbox_put*** **( )** - place a pointer in an mbox

Synopsis:

```
cyg_bool_t cyg_mbox_put
(
  cyg_handle_t mbox, /* mbox to add pointer to */
  void         *item /* pointer to add to mbox */
)
```

Description: This places a message into an mbox. If the mbox is already full this function will block until the message is placed into the mbox. If the thread is awaken by the kernel during a wait, this function will return an error.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if the message was placed into the mbox, "false" otherwise.

See Also: cyg_mbox_get, cyg_mbox_timed_get, cyg_mbox_tryget, cyg_mbox_peek_item, cyg_mbox_timed_put, cyg_mbox_tryput, cyg_mbox_peek, cyg_mbox_waiting_to_get, cyg_mbox_waiting_to_put

---

cyg_mbox_timed_put

Name: ***cyg_mbox_timed_put*** **( )** - place a message into an mbox with timeout

Synopsis:

```
cyg_bool_t cyg_mbox_timed_put
(
  cyg_handle_t     mbox,   /* mbox to add pointer to */
  void             *item,  /* pointer to add to mbox */
  cyg_tick_count_t abstime /* absolute timeout value */
)
```

Description: This places a pointer into an mbox. If the mbox is already full this function will block until "abstime" or until the message is placed into the mbox.

The delay is specified as an absolute time of the clock tick. To get a relative time use cyg_current_time to get the current system time and add an offset to that value.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if the message was placed into the mbox, "false" otherwise.

See Also: cyg_mbox_get, cyg_mbox_timed_get, cyg_mbox_tryget, cyg_mbox_peek_item, cyg_mbox_put, cyg_mbox_tryput, cyg_mbox_peek, cyg_mbox_waiting_to_get, cyg_mbox_waiting_to_put

---

cyg_mbox_tryput

Name: ***cyg_mbox_tryput*** **( )** - place a message in an mbox with no blocking

Synopsis:

```
cyg_bool_t cyg_mbox_tryput
(
  cyg_handle_t mbox, /* mbox to add pointer to */
  void         *item /* pointer to add to mbox */
)
```

Description: This places a message into an mbox. If the mbox is full, this function will fail to place the message into the mbox. This function will never block.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if the message was placed into the mbox, "false" otherwise.

See Also: cyg_mbox_get, cyg_mbox_timed_get, cyg_mbox_tryget, cyg_mbox_peek_item, cyg_mbox_put, cyg_mbox_timed_put, cyg_mbox_peek, cyg_mbox_waiting_to_get, cyg_mbox_waiting_to_put

---

cyg_mbox_peek

Name: ***cyg_mbox_peek*** **( )** - get number of messages in mbox

Synopsis:

```
cyg_count32 cyg_mbox_peek
(
  cyg_handle_t mbox /* mbox to peek into */
)
```

Description: This function will return the number of messages waiting to be processed in an mbox.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the number of messages currently in the given mbox.

See Also: cyg_mbox_get, cyg_mbox_timed_get, cyg_mbox_tryget, cyg_mbox_peek_item, cyg_mbox_put, cyg_mbox_timed_put, cyg_mbox_tryput, cyg_mbox_waiting_to_get, cyg_mbox_waiting_to_put

---

cyg_mbox_waiting_to_get

Name: ***cyg_mbox_waiting_to_get*** **( )** - check to see if threads wait to read from an mbox

Synopsis:

```
cyg_bool_t cyg_mbox_waiting_to_get
(
  cyg_handle_t mbox /* mbox to check */
)
```

Description: This indicates if any threads are being blocked while waiting for messages to be added to the mbox.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if threads are being blocked, "false" otherwise.

See Also: cyg_mbox_create, cyg_mbox_get, cyg_mbox_timed_get, cyg_mbox_tryget, cyg_mbox_peek_item, cyg_mbox_put, cyg_mbox_timed_put, cyg_mbox_tryput, cyg_mbox_peek, cyg_mbox_waiting_to_put

---

cyg_mbox_waiting_to_put

Name: ***cyg_mbox_waiting_to_put*** **( )** - check to see if thread waits to write to an mbox

Synopsis:

```
cyg_bool_t cyg_mbox_waiting_to_put
(
  cyg_handle_t mbox /* mbox to check */
)
```

Description: This indicates if any threads are being blocked while waiting for messages to be removed from the mbox.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if threads are being blocked, "false" otherwise.

See Also: cyg_mbox_create, cyg_mbox_get, cyg_mbox_timed_get, cyg_mbox_tryget, cyg_mbox_peek_item, cyg_mbox_put, cyg_mbox_timed_put, cyg_mbox_tryput, cyg_mbox_peek, cyg_mbox_waiting_to_get

---

---

cyg_mempool_fix_create

Name: ***cyg_mempool_fix_create*** **( )** - create a fixed sized memory pool

Synopsis:

```
void cyg_mempool_fix_create
(
  void            *base,     /* pointer to memory to use as heap */
  cyg_int32       size,      /* size of memory to use as heap    */
  cyg_int32       blocksize, /* size of blocks in fixed mempool  */
  cyg_handle_t    *handle,   /* returned handle to pool          */
  cyg_mempool_fix *fix       /* fix mempool structure            */
)
```

Description: This creates a memory pool that allows fixed size allocation of memory. Note that "size" will not necessarily be the total size of memory available once the memory pool is created since there is overhead. Fixed sized memory pools have the advantage of speed over variable sized memory pools. The newly created pool can be accessed via "handle".

Include: #include <**cyg/memalloc/kapi.h**>

Returns: nothing

See Also: cyg_mempool_fix_delete, cyg_mempool_fix_get_info

---

cyg_mempool_fix_delete

Name: ***cyg_mempool_fix_delete*** **( )** - delete a fixed sized memory pool

Synopsis:

```
void cyg_mempool_fix_delete
(
  cyg_handle_t fixpool /* fixed sized memory pool to delete */
)
```

Description: This destroys a fixed sized memory pool. Do not destroy a memory pool that is in use otherwise you risk hanging the system.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: nothing

See Also: cyg_mempool_fix_create

---

cyg_mempool_fix_alloc

Name: ***cyg_mempool_fix_alloc*** **( )** - allocate a fixed sized block of memory with no timeout

Synopsis:

```
void *cyg_mempool_fix_alloc
(
  cyg_handle_t fixpool /* fixed sized memory pool to allocate from  */
)
```

Description: This allocates a fixed sized block of memory from a memory pool. The alignment will be at least on a four byte boundary. If memory is not available this call will block the calling task until there is enough memory to fulfill the request.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: a pointer to the new memory, or NULL if the memory could not be allocated.

See Also: cyg_mempool_fix_timed_alloc, cyg_mempool_fix_try_alloc, cyg_mempool_fix_free, cyg_mempool_fix_waiting, cyg_mempool_fix_get_info

---

cyg_mempool_fix_timed_alloc

Name: ***cyg_mempool_fix_timed_alloc*** **( )** - allocate a fixed sized block of memory with timeout

Synopsis:

```
void *cyg_mempool_fix_timed_alloc
(
  cyg_handle_t     fixpool, /* fixed memory pool to allocate from */
  cyg_tick_count_t abstime  /* absolute timeout value             */
)
```

Description: This allocates a fixed sized block of memory from a memory pool. The alignment will be at least on a four byte boundary. If memory is not available this call will block the calling task until there is enough memory to fulfill the request or the system time reaches abstime.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: a pointer to the new memory, or NULL if the timeout was reached.

See Also: cyg_mempool_fix_alloc, cyg_mempool_fix_try_alloc, cyg_mempool_fix_free, cyg_mempool_fix_waiting, cyg_mempool_fix_get_info

---

cyg_mempool_fix_try_alloc

Name: ***cyg_mempool_fix_try_alloc*** **( )** - allocate a fixed sized block of memory, don't block

Synopsis:

```
void *cyg_mempool_fix_try_alloc
(
  cyg_handle_t fixpool /* fixed memory pool to allocate from */
)
```

Description: This allocates a fixed sized block of memory from a memory pool. The alignment will be at least on a four byte boundary. If memory is not available this call will return NULL immediately.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: a pointer to the new memory, or NULL if the memory could not be allocated.

See Also: cyg_mempool_fix_alloc, cyg_mempool_fix_timed_alloc, cyg_mempool_fix_free, cyg_mempool_fix_waiting, cyg_mempool_fix_get_info

---

cyg_mempool_fix_free

Name: ***cyg_mempool_fix_free*** **( )** - free a block of memory allocated from a fixed sized pool

Synopsis:

```
void cyg_mempool_fix_free
(
  cyg_handle_t fixpool, /* pool memory was allocated from */
  void         *p       /* memory to return to pool       */
)
```

Description: This frees memory that was allocated from a fixed sized memory pool. Be certain that you allocate from and free to the same "fixpool". If you allocate from one "fixpool" and free to another the behavior is undefined.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: nothing

See Also: cyg_mempool_fix_alloc, cyg_mempool_fix_timed_alloc, cyg_mempool_fix_try_alloc, cyg_mempool_fix_waiting, cyg_mempool_fix_get_info

---

cyg_mempool_fix_waiting

Name: ***cyg_mempool_fix_waiting*** **( )** - check to see if threads are waiting to allocate

Synopsis:

```
cyg_bool_t cyg_mempool_fix_waiting
(
  cyg_handle_t fixpool /* fixpool to check */
)
```

Description: This checks to see if any threads are being blocked waiting to allocate memory from a fixed sized pool.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: "true" if threads are blocked, "false" otherwise.

See Also: cyg_mempool_fix_alloc, cyg_mempool_fix_timed_alloc, cyg_mempool_fix_try_alloc, cyg_mempool_fix_free

---

cyg_mempool_fix_get_info

Name: ***cyg_mempool_fix_get_info*** **( )** - get info on a fixed sized mempool

Synopsis:

```
void cyg_mempool_fix_get_info
(
  cyg_handle_t     fixpool, /* pool to get info on */
  cyg_mempool_info *info    /* receives info       */
)
```

Description: This returns information about a memory pool. The information that is returned is described by the structure: typedef struct { cyg_int32 totalmem; cyg_int32 freemem; void *base; cyg_int32 size; cyg_int32 blocksize; cyg_int32 maxfree;} cyg_mempool_info;

Include: #include <**cyg/memalloc/kapi.h**>

Returns: nothing

See Also: cyg_mempool_fix_create

---

cyg_mempool_var_create

Name: ***cyg_mempool_var_create*** **( )** - create a variable sized memory pool

Synopsis:

```
void cyg_mempool_var_create
(
  void            *base,    /* pointer to memory to use as heap */
  cyg_int32       size,     /* size of memory to use as heap    */
  cyg_handle_t    *handle,  /* returned handle to pool          */
  cyg_mempool_var *var      /* mempool structure                */
)
```

Description: This creates a memory pool that allows variable size allocation of memory. Note that "size" will not necessarily be the total size of memory available once the memory pool is created since there is overhead. This provides equivalent functionality to standard C calls of free and malloc.

The newly created pool can be accessed via "handle".

Include: #include <**cyg/memalloc/kapi.h**>

Returns: nothing

See Also: cyg_mempool_var_delete, cyg_mempool_var_get_info

---

cyg_mempool_var_delete

Name: ***cyg_mempool_var_delete*** **( )** - delete a variable sized memory pool

Synopsis:

```
void cyg_mempool_var_delete
(
  cyg_handle_t varpool /* variable sized memory pool to delete */
)
```

Description: This destroys a variable sized memory pool. Do not destroy a memory pool that is in use otherwise you risk hanging the system.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: nothing

See Also: cyg_mempool_var_create

---

cyg_mempool_var_alloc

Name: ***cyg_mempool_var_alloc*** **( )** - allocate a variable size of memory with no timeout

Synopsis:

```
void *cyg_mempool_var_alloc
(
  cyg_handle_t varpool, /* variable memory pool to allocate from */
  cyg_int32    size     /* size of memory block to allocate      */
)
```

Description: This allocates an arbitrarily sized block of memory from a memory pool. The alignment will be at least on a four byte boundary. If memory is not available this call will block the calling task until there is enough memory to fulfill the request.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: a pointer to the new memory, or NULL if the memory could not be allocated.

See Also: cyg_mempool_var_timed_alloc, cyg_mempool_var_try_alloc, cyg_mempool_var_free, cyg_mempool_var_waiting, cyg_mempool_var_get_info

---

cyg_mempool_var_timed_alloc

Name: ***cyg_mempool_var_timed_alloc*** **( )** - allocate a variable size of memory with timeout

Synopsis:

```
void *cyg_mempool_var_timed_alloc
(
  cyg_handle_t     varpool, /* variable memory pool to allocate from */
  cyg_int32        size,    /* size of memory block to allocate      */
  cyg_tick_count_t abstime  /* absolute timeout value                */
)
```

Description: This allocates an arbitrary sized block of memory from a memory pool. The alignment will be at least on a four byte boundary. If memory is not available this call will block the calling task until there is enough memory to fulfill the request or the system time reaches abstime.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: a pointer to the new memory, or NULL if the timeout was reached.

See Also: cyg_mempool_var_alloc, cyg_mempool_var_try_alloc, cyg_mempool_var_free, cyg_mempool_var_waiting, cyg_mempool_var_get_info

---

cyg_mempool_var_try_alloc

Name: ***cyg_mempool_var_try_alloc*** **( )** - allocate a variable size of memory, don't block

Synopsis:

```
void *cyg_mempool_var_try_alloc
(
  cyg_handle_t varpool, /* variable memory pool to allocate from */
  cyg_int32    size     /* size of memory block to allocate      */
)
```

Description: This allocates an arbitrarily sized block of memory from a memory pool. The alignment will be at least on a four byte boundary. If memory is not available this call will return NULL immediately.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: a pointer to the new memory, or NULL if the memory could not be allocated.

See Also: cyg_mempool_var_alloc, cyg_mempool_var_timed_alloc, cyg_mempool_var_free, cyg_mempool_var_waiting, cyg_mempool_var_get_info

---

cyg_mempool_var_free

Name: ***cyg_mempool_var_free*** **( )** - free a block of memory allocated from a variable sized pool

Synopsis:

```
void cyg_mempool_var_free
(
  cyg_handle_t varpool, /* pool memory was allocated from */
  void         *p       /* memory to return to pool       */
)
```

Description: This frees memory that was allocated from a variable sized memory pool. Be certain that you allocate from and free to the same "varpool". If you allocate from one "varpool" and free to another the behavior is undefined.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: nothing

See Also: cyg_mempool_var_alloc, cyg_mempool_var_timed_alloc, cyg_mempool_var_try_alloc, cyg_mempool_var_waiting, cyg_mempool_var_get_info

---

cyg_mempool_var_waiting

Name: ***cyg_mempool_var_waiting*** **( )** - check to see if threads are waiting to allocate

Synopsis:

```
cyg_bool_t cyg_mempool_var_waiting
(
  cyg_handle_t varpool /* varpool to check */
)
```

Description: This checks to see if any threads are being blocked waiting to allocate memory from a variable sized pool.

Include: #include <**cyg/memalloc/kapi.h**>

Returns: "true" if threads are blocked, "false" otherwise.

See Also: cyg_mempool_var_alloc, cyg_mempool_var_timed_alloc, cyg_mempool_var_try_alloc, cyg_mempool_var_free

---

cyg_mempool_var_get_info

Name: ***cyg_mempool_var_get_info*** **( )** - get info on a variable sized mempool

Synopsis:

```
void cyg_mempool_var_get_info
(
  cyg_handle_t     varpool, /* pool to get info on */
  cyg_mempool_info *info    /* receives info       */
)
```

Description: This returns information about a memory pool. The information that is returned is described by the structure: typedef struct { cyg_int32 totalmem; cyg_int32 freemem; void *base; cyg_int32 size; cyg_int32 blocksize; cyg_int32 maxfree;} cyg_mempool_info;

Include: #include <**cyg/memalloc/kapi.h**>

Returns: nothing

See Also: cyg_mempool_var_create

---

cyg_mutex_init

Name: ***cyg_mutex_init*** **( )** - initialize a mutex

Synopsis:

```
void cyg_mutex_init
(
  cyg_mutex_t *mutex /* mutex to initialize */
)
```

Description: This initializes a mutex for use. Note that mutexes under eCos cannot be locked multiple times by the same thread. If a thread locks the same mutex multiple times the behavior is undefined.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_mutex_destroy

---

cyg_mutex_destroy

Name: ***cyg_mutex_destroy*** **( )** - destroy (invalidate) a mutex

Synopsis:

```
void cyg_mutex_destroy
(
  cyg_mutex_t *mutex /* mutex to destroy (invalidate) */
)
```

Description: This destroys (invalidates) a mutex. Be careful not to destroy a mutex that other threads are waiting on or is otherwise in use. If you destroy a mutex that is in use, you risk deadlocking the system.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_mutex_init

---

cyg_mutex_lock

Name: ***cyg_mutex_lock*** **( )** - lock a mutex or wait to lock one

Synopsis:
```
cyg_bool_t cyg_mutex_lock
(
  cyg_mutex_t *mutex /* mutex to lock */
)
```

Description: This locks a mutex. If the mutex is not available, the thread will be blocked until the mutex is available or the thread is awaken by a signal.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if the mutex was locked, "false" otherwise.

See Also: cyg_mutex_trylock, cyg_mutex_unlock, cyg_mutex_release

---

cyg_mutex_trylock

Name: ***cyg_mutex_trylock*** **( )** - attempt to lock a mutex

Synopsis:
```
cyg_bool_t cyg_mutex_trylock
(
  cyg_mutex_t *mutex /* mutex to attempt lock */
)
```

Description: This locks a mutex. If the mutex is not available, an error is is returned.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if the mutex was locked, "false" if the mutex couldn't be locked.

See Also: cyg_mutex_lock, cyg_mutex_unlock, cyg_mutex_release

---

cyg_mutex_unlock

Name: ***cyg_mutex_unlock*** **( )** - unlocks a mutex

Synopsis:
```
void cyg_mutex_unlock
(
  cyg_mutex_t *mutex /* mutex to unlock */
)
```

Description: This unlocks a mutex. Note that it is undefined behavior to unlock a mutex that is in an unlocked state, or to unlock a mutex that was locked by another thread.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_mutex_lock, cyg_mutex_trylock, cyg_mutex_release

---

cyg_mutex_release

Name: ***cyg_mutex_release*** **( )** - release all threads waiting on a mutex

Synopsis:
```
void cyg_mutex_release
(
  cyg_mutex_t *mutex /* mutex to release */
)
```

Description: This releases all threads waiting on a mutex. All threads that were waiting on the mutex will be receive an error condition indicating that the mutex was not acquired.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_mutex_lock, cyg_mutex_trylock, cyg_mutex_unlock

---

cyg_mutex_set_ceiling

Name: ***cyg_mutex_set_ceiling*** **( )** - set ceiling priority of mutex

Synopsis:
```
void cyg_mutex_set_ceiling
(
  cyg_mutex_t    *mutex,  /* mutex to set ceiling of */
  cyg_priority_t priority /* ceiling priority       */
)
```

Description: This sets the ceiling priority of a thread that acquires the given mutex. This is only meaningful if the protocol of the mutex is set to CYG_MUTEX_CEILING. Mutexes with ceilings cause the thread that has acquired the mutex to inherit the ceiling priority temporarily to avoid deadlocks.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_mutex_set_protocol

---

cyg_mutex_set_protocol

Name: ***cyg_mutex_set_protocol*** **( )** - set the protocol of a mutex

Synopsis:
```
void cyg_mutex_set_protocol
(
  cyg_mutex_t              *mutex,  /* mutex to set protocol of */
  enum cyg_mutex_protocol protocol /* protocol to use          */
)
```

Description: This sets the protocol of a mutex. The following protocols are valid:

CYG_MUTEX_NONE - no priority inheritance

CYG_MUTEX_INHERIT - inherit priority of thread currently holding mutex

CYG_MUTEX_CEILING - inherit ceiling priority of mutex

A priority will only be inherited if it causes the thread holding the mutex to go up in priority.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_mutex_set_ceiling

---

cyg_scheduler_start

Name: ***cyg_scheduler_start*** **( )** - start scheduler

Synopsis:

```
void cyg_scheduler_start
(
  void
)
```

Description: Start the system scheduler. You normally do not have to call this function as eCos will have called it by default.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_scheduler_lock, cyg_scheduler_safe_lock, cyg_scheduler_unlock, cyg_scheduler_read_lock

---

cyg_scheduler_lock

Name: ***cyg_scheduler_lock*** **( )** - lock scheduler

Synopsis:

```
void cyg_scheduler_lock
(
  void
)
```

Description: This will lock the scheduler. In other words, this will prevent the thread from being pre-empted by another thread. In order to enable thread switching again you must call cyg_scheduler_unlock the same number of times this function has been called.

Use this function rather than disabling interrupts to create atomic operations whenever possible. If possible, you should usemutexes where possible.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_scheduler_safe_lock, cyg_scheduler_unlock, cyg_scheduler_read_lock

---

cyg_scheduler_safe_lock

Name: ***cyg_scheduler_safe_lock*** **( )** - lock the scheduler if it's not already locked

Synopsis:

```
void cyg_scheduler_safe_lock
(
  void
)
```

Description: Lock the scheduler if it's not already locked. If the scheduler has already been locked once or multiple times this function has no effect on the lock count. If the lock count is 0 (i.e. the scheduler has not been locked) this will lock the scheduler and set the lock count to 1.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_scheduler_lock, cyg_scheduler_unlock, cyg_scheduler_read_lock

---

cyg_scheduler_unlock

Name: ***cyg_scheduler_unlock*** **( )** - unlock the scheduler

Synopsis:

```
void cyg_scheduler_unlock
(
  void
)
```

Description: This function will decrement the lock count. For every call to cyg_scheduler_lock there must be a call to cyg_scheduler_unlock in order to actually unlock the scheduler to enable thread switching again. If the scheduler is already unlocked the behavior is undefined.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_scheduler_lock, cyg_scheduler_safe_lock, cyg_scheduler_read_lock

---

cyg_scheduler_read_lock

Name: ***cyg_scheduler_read_lock*** **( )** - read scheduler lock count

Synopsis:

```
cyg_ucount32 cyg_scheduler_read_lock
(
  void
)
```

Description: This gets the current scheduler lock count. If the thread is is not locked this will return 0. If the lock count is N, the cyg_scheduler_unlock function will have to be called N times to enable context switching again.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the current lock count

See Also: cyg_scheduler_lock, cyg_scheduler_safe_lock, cyg_scheduler_unlock

---

---

cyg_semaphore_init

Name: ***cyg_semaphore_init*** **( )** - initialize a counting semaphore

Synopsis:

```
void cyg_semaphore_init
(
  cyg_sem_t   *sem, /* semaphore to initialize */
  cyg_count32 val   /* initial semaphore count */
)
```

Description: This initializes a counting semaphore for use.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_semaphore_destroy

---

cyg_semaphore_destroy

Name: ***cyg_semaphore_destroy*** **( )** - destroy (invalidate) a semaphore

Synopsis:

```
void cyg_semaphore_destroy
(
  cyg_sem_t *sem /* semaphore to invalidate */
)
```

Description: This invalidates a semaphore for further use. Be certain that no other threads are waiting on or otherwise using the semaphore or you will risk deadlocking the system.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_semaphore_init

---

cyg_semaphore_wait

Name: ***cyg_semaphore_wait*** **( )** - wait on a counting semaphore

Synopsis:

```
cyg_bool_t cyg_semaphore_wait
(
  cyg_sem_t *sem /* semaphore to wait on */
)
```

Description: This requests a semaphore. If the semaphore count is set to 0, it will block the calling thread until the semphore count is incremented. If several threads are waiting on the same semaphore, the scheduler will determine which task gets the semaphore first. Generally, it will be the thread with the highest priority.

If the thread is awaken for some other reason, this function may return false without actually acquiring the semaphore.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if the semaphore was acquired, "false" otherwise.

See Also: cyg_semaphore_timed_wait, cyg_semaphore_trywait, cyg_semaphore_post, cyg_semaphore_peek

---

cyg_semaphore_timed_wait

Name: ***cyg_semaphore_timed_wait*** **( )** - wait on a semaphore with timeout

Synopsis:

```
cyg_bool_t cyg_semaphore_timed_wait
(
  cyg_sem_t        *sem,   /* semaphore to wait on   */
  cyg_tick_count_t abstime /* absolute timeout value */
)
```

Description: This requests a semaphore. If the semaphore count is set to 0, it will block the calling thread until the semaphore count is incremented or the system time reaches (or surpasses) "abstime". If several threads are waiting on the same semaphore, the scheduler will determine which task gets the semaphore first. Generally, it will be the thread with the highest priority. You can get the current system time by calling cyg_current_time.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if the semaphore was acquired, "false" otherwise.

See Also: cyg_semaphore_wait, cyg_semaphore_trywait, cyg_semaphore_post, cyg_semaphore_peek

---

cyg_semaphore_trywait

Name: ***cyg_semaphore_trywait*** **( )** - get a semaphore if available

Synopsis:

```
int cyg_semaphore_trywait
(
  cyg_sem_t *sem /* semaphore to get */
)
```

Description: This requests a semaphore. If the semaphore count is set to 0, it will not block the calling thread, but will indicate an error.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if the semaphore was acquired, "false" otherwise.

See Also: cyg_semaphore_wait, cyg_semaphore_timed_wait, cyg_semaphore_post, cyg_semaphore_peek

---

cyg_semaphore_post

Name: ***cyg_semaphore_post*** **( )** - increment semaphore count

Synopsis:

```
void cyg_semaphore_post
(
  cyg_sem_t *sem /* semaphore to increment count of */
)
```

Description: Increment the count of the given semaphore.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_semaphore_wait, cyg_semaphore_timed_wait, cyg_semaphore_trywait, cyg_semaphore_peek

---

cyg_semaphore_peek

Name: ***cyg_semaphore_peek*** **( )** - get current semaphore count

Synopsis:

```
void cyg_semaphore_peek
(
  cyg_sem_t   *sem, /* semaphore to get count of */
  cyg_count32 *val  /* pointer to receive count  */
)
```

Description: This gets the current count of a counting semaphore through a pointer to "val" (the second argument).

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_semaphore_wait, cyg_semaphore_timed_wait, cyg_semaphore_trywait, cyg_semaphore_post

---

---

cyg_thread_create

Name: ***cyg_thread_create*** **( )** - create a new thread

Synopsis:

```
void cyg_thread_create
(
  cyg_addrword_t     sched_info,  /* scheduling info (priority) */
  cyg_thread_entry_t *entry,      /* thread entry point         */
  cyg_addrword_t     entry_data,  /* entry point argument       */
  char               *name,       /* name of thread             */
  void               *stack_base, /* pointer to stack base      */
  cyg_ucount32       stack_size,  /* size of stack in bytes     */
  cyg_handle_t       *handle,     /* returned thread handle     */
  cyg_thread         *thread      /* space to store thread data */
)
```

Description: This creates a new thread. The ID of the thread is returned through "*handle". Since eCos does it's best to never use dynamic memory "*thread" is is used to store thread specific information. Once the thread is created it can be manipulated with "handle".

One thing to note is that like Unix, the lower the priority value (i.e. sched_info) the higher the priority. A priority of 0 is the highest possible priority in the system and CYG_THREAD_MIN_PRIORITY is the lowest possible priority. It is a good idea not to run any thread at CYG_THREAD_MIN_PRIORITY since the idle thread runs at that priority.

Priority values depend on the scheduler. Note that if you use the bitmap scheduler, two threads cannot share the same priority.

Note that threads are created in a suspended state. Before the thread will run, you must call cyg_thread_resume.

The entry function prototype is: void cyg_thread_entry(CYG_ADDRWORD data).

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing.

See Also: cyg_thread_exit, cyg_thread_delete, cyg_thread_kill, cyg_thread_set_priority, cyg_thread_add_destructor, cyg_thread_rem_destructor

---

cyg_thread_exit

Name: ***cyg_thread_exit*** **( )** - exit a thread

Synopsis:

```
void cyg_thread_exit
(
  void
)
```

Description: This stops the calling thread. Be sure that any resources that have been allocated by the thread have been freed before calling this function otherwise you risk deadlocking the system.

This will cause any destructors associated with the thread to be called before the thread exits.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing - this function doesn't return

See Also: cyg_thread_create, cyg_thread_exit, cyg_thread_delete, cyg_thread_resume, cyg_thread_kill, cyg_thread_add_destructor, cyg_thread_rem_destructor

---

cyg_thread_delete

Name: ***cyg_thread_delete*** **( )** - delete a thread

Synopsis:

```
cyg_bool_t cyg_thread_delete
(
  cyg_handle_t thread /* thread to delete */
)
```

Description: This deletes a thread. This is a dangerous function to call. If it's possible, send a message to the thread you want to delete and let itself shutdown with cyg_thread_exit.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" is the thread was deleted, "false" otherwise.

See Also: cyg_thread_create, cyg_thread_exit, cyg_thread_kill, cyg_thread_release, cyg_thread_add_destructor, cyg_thread_rem_destructor

---

cyg_thread_suspend

Name: ***cyg_thread_suspend*** **( )** - suspend a thread

Synopsis:

```
void cyg_thread_suspend
(
  cyg_handle_t thread /* thread to suspend */
)
```

Description: This suspends a thread. A thread can be suspended multiple times. For each call to cyg_thread_suspend, there must be a call to cyg_thread_resume to take the thread out of the suspended state.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_thread_resume

---

cyg_thread_resume

Name: ***cyg_thread_resume*** **( )** - resume a suspended thread

Synopsis:

```
void cyg_thread_resume
(
  cyg_handle_t thread /* thread to resume from a suspended state */
)
```

Description: This decrements the suspend count on a thread. If the suspend count goes to 0, the thread will be resumed and will then continue to run.

If the thread was exited, this will reinitialize the thread.

(? RBW: Will reinitializing thread cause the thread to reinitialize in a suspended state or a running state ? - need to test)

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_thread_suspend

---

cyg_thread_kill

Name: ***cyg_thread_kill*** **( )** - kill a thread

Synopsis:

```
void cyg_thread_kill
(
  cyg_handle_t thread /* thread to kill */
)
```

Description: This will force another thread to exit and cause it to stop. This is dangerous to use because if the thread that is being killed has any resources allocated (memory, semaphores, mutexes, etc.) they will not be freed when the thread is killed. This can cause the system to deadlock.

This will cause any destructors associated with the thread to be called before the thread is killed.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_thread_create, cyg_thread_exit, cyg_thread_delete

---

cyg_thread_release

Name: ***cyg_thread_release*** **( )** - release a thread from a wait

Synopsis:
```
void cyg_thread_release
(
  cyg_handle_t thread /* thread to release */
)
```

Description: This causes a thread that is in a wait to be broken out of it. It is the responsibility of the thread that was released to detect that it was broken out of it's wait. If you make use of this function, you will have to program defensively around any synchronization mechanism.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_thread_delay

---

cyg_thread_yield

Name: ***cyg_thread_yield*** **( )** - yield the thread to another thread of equal priority

Synopsis:
```
void cyg_thread_yield
(
  void
)
```

Description: This yields control of the CPU to the next runnable thread of equal priority. If there is no other runnable thread of equal priority this will effectively do nothing.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_thread_delay

---

cyg_thread_self

Name: ***cyg_thread_self*** **( )** - get calling thread's thread ID

Synopsis:
```
cyg_handle_t cyg_thread_self
(
  void
)
```

Description: This gets the thread ID of the calling thread.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the thread ID of the calling thread

See Also: cyg_thread_create

---

cyg_thread_idle_thread

Name: ***cyg_thread_idle_thread*** **( )** - get the idle thread's thread ID

Synopsis:
```
cyg_handle_t cyg_thread_idle_thread
(
  void
)
```

Description: This function gets the idle thread's thread ID. Be careful with modifying the idle thread's priority, or in general doing anything with it since it's not a standard thread.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the thread ID of the idle thread

See Also:

---

cyg_thread_set_priority

Name: ***cyg_thread_set_priority*** **( )** - set the priority of a thread

Synopsis:
```
void cyg_thread_set_priority
(
  cyg_handle_t   thread,  /* thread ID    */
  cyg_priority_t priority /* new priority */
)
```

Description: This changes the priority of the specified thread. Like Unix, the lower the value of the priority the higher the priority. A priority of 0 is the highest priority in the system. CYG_THREAD_MIN_PRIORITY is the lowest priority and it's value is dependent on the scheduler that you use.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_thread_create, cyg_thread_get_priority, cyg_thread_get_current_priority

---

cyg_thread_get_priority

Name: ***cyg_thread_get_priority*** **( )** - get the priority of a thread

Synopsis:
```
cyg_priority_t cyg_thread_get_priority
(
  cyg_handle_t thread /* thread ID */
)
```

Description: This returns the set priority of the given thread. If the priority of the thread has been modified by a mutex with a ceiling, priority inversion or some other similar mechanism this will **not** return the current priority, but the "normal" priority of the thread.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the set priority of the given thread

See Also: cyg_thread_create, cyg_thread_set_priority, cyg_thread_get_current_priority

---

cyg_thread_get_current_priority

Name: ***cyg_thread_get_current_priority*** **( )** - get current priority of a thread

Synopsis:
```
cyg_priority_t cyg_thread_get_current_priority
(
  cyg_handle_t thread /* thread ID */
)
```

Description: This returns the current priority of the given thread. If the priority of the thread has been modified by a mutex with a ceiling, priority inversion or some other similar mechanism this **will** return the current priority, not the "normal" priority of the thread.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the current running priority of the given thread.

See Also: cyg_thread_create, cyg_thread_set_priority, cyg_thread_get_priority

---

cyg_thread_delay

Name: ***cyg_thread_delay*** **( )** - delay the calling thread for a number of ticks

Synopsis:
```
void cyg_thread_delay
(
  cyg_tick_count_t delay /* number of ticks to delay */
)
```

Description: This delays a thread for an arbitrary number of ticks. The length of a tick is provided by the resolution of the system clock. (RBW: provide more information on getting information on the real time clock)

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_thread_release

---

cyg_thread_get_stack_base

Name: ***cyg_thread_get_stack_base*** **( )** - get a thread's stack base address

Synopsis:
```
cyg_addrword_t cyg_thread_get_stack_base
(
  cyg_handle_t thread /* thread ID */
)
```

Description: This returns the address of a given thread's stack. This may or may not be what was passed in the creation of the thread since debug capabilities and alignment requirements might modify it somewhat.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the base address of the thread's stack.

See Also: cyg_thread_create, cyg_thread_get_stack_size, cyg_thread_measure_stack_usage

---

cyg_thread_get_stack_size

Name: ***cyg_thread_get_stack_size*** **( )** - get a thread's stack size

Synopsis:

```
cyg_uint32 cyg_thread_get_stack_size
(
  cyg_handle_t thread /* thread ID */
)
```

Description: This returns the size of a given thread's stack. This may or may not be what was passed in the creation of the thread since debug capabilities and alignment requirements might modify it somewhat.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the size in bytes of the thread's stack.

See Also: cyg_thread_create, cyg_thread_get_stack_base, cyg_thread_measure_stack_usage

---

cyg_thread_measure_stack_usage

Name: ***cyg_thread_measure_stack_usage*** **( )** - get a thread's current stack usage

Synopsis:

```
cyg_uint32 cyg_thread_measure_stack_usage
(
  cyg_handle_t thread /* thread ID */
)
```

Description: This gets the number of bytes that have been used so far by the given thread. Note that if this function returns 0, it's likely you will overrun the stack in the future. This is essentially a debug function.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the number of bytes currently used by the given thread.

See Also: cyg_thread_create, cyg_thread_get_stack_base, cyg_thread_get_stack_size

---

cyg_thread_new_data_index

Name: ***cyg_thread_new_data_index*** **( )** - get a free data index for all threads

Synopsis:
```
cyg_ucount32 cyg_thread_new_data_index
(
  void
)
```

Description: This gets a new unused data index. The index allocated is useful for storing data that is specific to each thread. For example, the global variable "errno" may be allocated as a per thread data variable.

Include: #include <**cyg/kernel/kapi.h**>

Returns: a free data index or -1 if there were no more free indexes.

See Also: cyg_thread_free_data_index, cyg_thread_get_data, cyg_thread_get_data_ptr, cyg_thread_set_data

---

cyg_thread_free_data_index

Name: ***cyg_thread_free_data_index*** **( )** - free a data index for all threads

Synopsis:
```
void cyg_thread_free_data_index
(
  cyg_ucount32 index /* index to free */
)
```

Description: This frees a data index and makes it available for use again by the system.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_thread_new_data_index, cyg_thread_get_data, cyg_thread_get_data_ptr, cyg_thread_set_data

---

cyg_thread_get_data

Name: ***cyg_thread_get_data*** **( )** - read per thread data from a given index

Synopsis:
```
CYG_ADDRWORD cyg_thread_get_data
(
  cyg_ucount32 index /* index of per thread data */
)
```

Description: Get per thread data. Be careful with this, giving a bad index will cause an ASSERT if you have it turned on, but no other error checking is performed otherwise.

Include: #include <**cyg/kernel/kapi.h**>

Returns: the per thread data stored at a specified index.

See Also: cyg_thread_new_data_index, cyg_thread_free_data_index, cyg_thread_get_data_ptr, cyg_thread_set_data

---

cyg_thread_get_data_ptr

Name: ***cyg_thread_get_data_ptr*** **( )** - get a data pointer to per thread data

Synopsis:
```
CYG_ADDRWORD *cyg_thread_get_data_ptr
(
  cyg_ucount32 index /* index of per thread data */
)
```

Description: Get a pointer to per thread data. You can use this function in lieu of cyg_thread_get_data or cyg_thread_set_data, and it's a little faster to use. The pointer, of course, is only valid in the context of the calling thread.

Include: #include <**cyg/kernel/kapi.h**>

Returns: a pointer to per thread data at the given index.

See Also: cyg_thread_new_data_index, cyg_thread_free_data_index, cyg_thread_get_data, cyg_thread_set_data

---

cyg_thread_set_data

Name: ***cyg_thread_set_data*** **( )** - set per thread data at a given index

Synopsis:
```
void cyg_thread_set_data
(
  cyg_ucount32 index, /* index of per thread data */
  CYG_ADDRWORD data   /* data to write            */
)
```

Description: Set per thread data. Be careful with this, giving a bad index will cause an ASSERT if you have it turned on, but no other error checking is performed otherwise.

Include: #include <**cyg/kernel/kapi.h**>

Returns: nothing

See Also: cyg_thread_new_data_index, cyg_thread_free_data_index, cyg_thread_get_data, cyg_thread_get_data_ptr

---

cyg_thread_add_destructor

Name: ***cyg_thread_add_destructor*** **( )** - add a destructor

Synopsis:
```
cyg_bool_t cyg_thread_add_destructor
(
  cyg_thread_destructor_fn fn,  /* destructor function    */
  cyg_addrword_t           data /* argument to destructor */
)
```

Description: This causes a destructor to be added to the calling thread. Destructors are called before the thread exits or is killed.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if the thread was added successfully, "false" otherwise.

See Also: cyg_thread_exit, cyg_thread_delete, cyg_thread_kill, cyg_thread_rem_destructor

---

cyg_thread_rem_destructor

Name: ***cyg_thread_rem_destructor*** **( )** - remove (disable) a destructor

Synopsis:

```
cyg_bool_t cyg_thread_rem_destructor
(
  cyg_thread_destructor_fn fn,  /* destructor function    */
  cyg_addrword_t           data /* argument to destructor */
)
```

Description: This causes a destructor to be removed from the calling thread. Destructors are called before the thread exits or is killed. In order to successfully remove a destructor, both the "fn" and "data" argument must match exactly a previously installed destructor.

Include: #include <**cyg/kernel/kapi.h**>

Returns: "true" if a destructor was removed successfully, "false" otherwise.

See Also: cyg_thread_exit, cyg_thread_delete, cyg_thread_kill, cyg_thread_add_destructor, cyg_thread_rem_destructor