**ERROR CONDITIONS**

**Invalid tid.**

**Invalid register number.**

**ISR cannot reference remote node.**

**NOTES**

**Can be called from within an ISR, except when the task was not created on the local node.**

**Will not cause a preempt.**

**3.2 Message, Event, and Signal Management**

**The executive supports communication and synchronisation between tasks using messages and events. Asynchronous communication is supported using signals.**

**3.2.1 Message Manager**

**The message queue is the data structure supporting inter-task communication and synchronization. One or more tasks may send messages to the message queue, and one or more tasks may request message from the queue.**

**Message queues are created at run time using the q\_create directive. The creator assigns a 4-byte name and attributes to the queue. The attributes define whether tasks waiting on messages from the queue will wait first-in, first-out (FIFO), or by task priority, and whether the queue will limit the number of messages queued to a specified maximum, or allow an unlimited number of messages.**

**A message queue is identified by both a name, assigned by the creator, and a message queue id (qid), assigned by the executive at q\_create time. The qid is returned to the caller by the q\_create directive, and must be used by tasks to send and receive messages from the message queue. Tasks other than the task which created the message queue can obtain the qid by using the q\_ident directive.**

**Messages are sent to the message queue from any task which knows the qid, using the q\_send, q\_urgent, and q\_broadcast directives.**

**When a message arrives at the queue, it will be copied into one of two places. If there is one or more tasks waiting at the queue, then the message is copied into the message buffer belonging to the waiting task. The task is removed from the wait list and is made ready. If there are no tasks waiting at the queue, then the message is copied into a system message buffer (the executive maintains a pool of system message buffers for this purpose). This system message buffer is entered into the message queue. If the message was sent using q\_send, the message is entered at the tail of the queue. If the message was sent using q\_urgent, the message is entered at the head of the queue. The q\_broadcast directive sends a message to all tasks waiting at the queue, so they become ready to run. The count of readied tasks is returned to the caller.**

**Messages are received from the message queue using the q\_receive directive. When this directive is called, and a message is in the queue, the message is copied to the task’s message buffer, and the directive is complete. When no message is in the queue, there are several ways to proceed. If the calling task asked to wait, the task will be entered into the queue’s wait list according the queue’s attributes (FIFO or priority). If the calling task asked to wait with timeout, the task will be entered into a timeout list. If the calling task asked not to wait, the task will be returned to with an error code for no message available.**

**Message queues can be deleted by tasks knowing the qid using the q\_delete directive. If any messages are queued, the executive will claim and return the system message buffers to the system message buffer pool. If any tasks are waiting on the queue, then the executive will remove them from the wait list and make them ready. Waiting tasks will return from the q\_receive directive with the message queue deleted error.**

**The message manager defines a message as being fixed length, 16-bytes. The content of the message is user defined. It may be used to carry data, pointers to data, or nothing at all.**

**The directives provided by the message manager are:**

|  |  |
| --- | --- |
| **Directive** | **Function** |
| **q\_create** | **Create queue** |
| **q\_ident** | **Obtain id of a queue** |
| **q\_delete** | **Delete queue** |
| **q\_send** | **Send message** |
| **q\_urgent** | **Urgent message** |
| **q\_broadcast** | **Broadcast message** |
| **q\_receive** | **Receive message** |

**3.2.2 Event Manager**

**Although inter-task synchronization can be accomplished using the message queue, the executive also provides a second, higher performance method of inter-task synchronization, using events.**

**Events are different from messages in that they are directed at other tasks. They are also different from messages in that they carry no information, and they cannot be queued. The final difference is tasks can wait for several events at one time, but cannot wait on multiple message queues at one time.**

**Every task in the system has the ability to send and receive events. Events are simply bits encoded into an event mask. Thirty-two events are available; sixteen will be available as system events and sixteen will be available as user events. A task can send one or more events to another task using the ev\_send directive. The tid of the destination task is required as input, along with the event set.**

**A task can receive events using the ev\_receive directive. The events to receive are input to the directive, along with an option to wait on all of the events, or just one of them. If the events are already pending, then the event mask is cleared before returning to the calling task. If the event condition cannot be satisfied, and the calling task asked to wait, the task will be blocked. If the calling task asked to wait with timeout, the task will be entered into a timeout list. Tasks that do not want to wait for the event condition must specify this as an option. If the event condition was not pending, then an error code for event condition not met is returned.**

**The directives provided by the event manager are:**

|  |  |
| --- | --- |
| **Directive** | **Function** |
| **ev\_send** | **Send event** |
| **ev\_receive** | **Receive event** |

**3.2.3 Signal Manager**

**Asynchronous communication is supported through the use of signals.**

**Signals, like events, are simply bits encoded into a signal mask. Thirty-two signals are available; sixteen will be available as system signals and sixteen will be available as user signals.**

**A task can send one or more signals to another task using the as\_send directive. If the receiving task has set up an asynchronous signal routine (asr) using the as\_catch directive, the task will be dispatched to the signal routine.**

**A task may asynchronously receive signals by establishing an asynchronous signal routine (asr) to catch them using the as\_catch directive. When a signal is caught, the task will be dispatched to the asr address when it becomes the running task. The signal condition will be passed to the task to enable it to determine what signals occurred.**

**The as\_return directive must be executed to return the task to its previous dispatch address.**

**The directives provided by the signal manager are:**

|  |  |
| --- | --- |
| **Directive** | **Function** |
| **as\_catch** | **Catch signal** |
| **as\_send** | **Send signal** |
| **as\_return** | **Return from signal** |

**3.2.4 Data Structures for Message Management**

**Definitions for events and asynchronous signals are as follows:**

**S\_EXEC0 System Software defined**

**S\_EXEC1 System Software defined**

**S\_EXEC2 System Software defined**

**S\_EXEC3 System Software defined**

**S\_EXEC4 System Software defined**

**S\_EXEC5 System Software defined**

**S\_EXEC6 System Software defined**

**S\_EXEC7 System Software defined**

**S\_EXEC8 System Software defined**

**S\_EXEC9 System Software defined**

**S\_EXEC10 System Software defined**

**S\_EXEC11 System Software defined**

**S\_EXEC12 System Software defined**

**S\_EXEC13 System Software defined**

**S\_EXEC14 System Software defined**

**S\_EXEC15 System Software defined**

**S\_USER0 User defined**

**S\_USER1 User defined**

**S\_USER2 User defined**

**S\_USER3 User defined**

**S\_USER4 User defined**

**S\_USER5 User defined**

**S\_USER6 User defined**

**S\_USER7 User defined**

**S\_USER8 User defined**

**S\_USER9 User defined**

**S\_USER10 User defined**

**S\_USER11 User defined**

**S\_USER12 User defined**

**S\_USER13 User defined**

**S\_USER14 User defined**

**S\_USER15 User defined**

**3.2.5 Q\_CREATE**

**NAME**

**q\_create - “Create a Message Queue”**

**SYNOPSIS**

**#include <message.h>**

**uint q\_create (name, count, flags, &qid )**

 **uint name; /\* user defined 4-byte name \*/**

 **uint count; /\* maximum message and reserved buffer count \*/**

**uint flags; /\* process method \*/**

**uint qid; /\* message queue id - returned by this call \*/**

**The flags values are:**

 **PRIOR set to process by priority**

 **clear to process by FIFO**

 **GLOBAL set to indicate the queue is a**

**multiprocessor global resource.**

 **clear to indicate the queue is local**

 **TYPE set to process typed messages**

 **clear to process messages without regard to type**

 **LIMIT set to limit queue entries to number in count field**

 **clear NO limit on queue entries and no reserved buffers**

 **RESVD set to reserve system buffers equal to count when LIMIT is set**

 **clear NO reserved system buffers when LIMIT is set**

**DESCRIPTION**

**The q\_create directive creates a message queue by allocating and initializing a message queue data structure. A message queue is created by name. A message qid is returned. Subsequent sending and receiving calls must reference the message queue with its message qid.**

**By setting the PRIOR value in the flags field, tasks waiting for messages in the queue will be processed by task priority order. Otherwise the tasks waiting for messages will be Processed by first in, first out (FIFO) order.**

**By setting the TYPE value in the flags field, messages sent to this queue may be processed by type.**

**The user may put a limit on the number of messages at the message queue by setting the LIMIT value in the flags field, and placing the count in the count field. The user may additionally reserve a number of system message buffers equal to the count in the count field by setting the RESVD value in the flags field.**

**By setting the GLOBAL value in the flags field, the message qid will be sent to all processors in**

 **the system, to be entered into a global resource table. The system is defined as the collection of interconnected processors. The message queue is always created on the local node.**

**The maximum number of message queues that can be in existence at one time is a configuration parameter.**

**The maximum number of system message buffers is a configuration parameter.**

**RETURN VALUE**

**If the q\_create directive succeeds, the qid is filled in, and 0 is returned.**

**If the call was not successful, an error code is returned.**

**ERROR CONDITIONS**

**Too many message queues.**

**No more system message buffers.**

**NOTES**

**Not callable from ISR.**

**Will not cause a preempt**

**3.2.6 Q\_IDENT**

**NAME**

**q\_ident - ‘Obtain id of a Message Queue"**

**SYNOPSIS**

**#include <message.h>**

**uint q\_ident ( name, node, &qid )**

 **uint name; /\* user defined 4-byte name \*/**

 **uint node; /\* node identifier \*/**

 **/\* 0 indicates any node \*/**

 **uint qid; /\* message queue id - returned by this call \*/**

**DESCRIPTION**

**The q\_ident directive allows a task to identify a previously created message queue by name and receive the message qid to use for send and receive directives for the queue.**

**If the message queue name is not unique, the message qid returned may not correspond to the message queue named in this call.**

**The message queue may exist on the local processor or any remote processor in a multiprocessor configuration, as long as the queue was created with the GLOBAL flags value set (see q\_create). If the message queue name is not unique within the multiprocessor configuration, a non-zero node identifier must be specified in the node field.**

**RETURN VALUE**

**If the q\_ident directive succeeds, the qid is filled in, and 0 is returned.**

**If the call was not successful, an error code is returned.**

**ERROR CONDITIONS**

**Named message queue does not exist.**

**Invalid node identifier.**

**NOTES**

**Can be called from within an ISR.**

**Will not cause a preempt.**

**3.2.7 Q\_DELETE**

**NAME**

**q\_delete - “Delete a Message Queue”**

**SYNOPSIS**

**#include <message.h>**

**uint q\_delete ( qid )**

 **uint qid; /\* message queue id returned from q\_create or q\_ident \*/**

**DESCRIPTION**

**The q\_deIete directive deletes the message queue identified by the qid, freeing the data structure.**

**When a message queue is deleted, it could be in one of three states: empty, tasks waiting for messages, messages waiting for tasks. If empty, the data structure of the message queue is returned to the system. If tasks are waiting, each is made ready and given a return code indicating a deleted message queue. If messages are waiting, then each system message buffer is returned to the system message buffer pool, and the message it is carrying is therefore lost.**

**The message queue must exist on the local processor. If the message queue was created with the GLOBAL flags value set in a multiprocessor configuration, a notification will be sent to all processors in the system, so the qid can be deleted from the global resource table.**

**The requester does not have to be the creator of the message queue. Any task knowing the qid can delete it.**

**RETURN VALUE**

**If the q\_delete directive successfully deleted the message queue, then 0 is returned.**

**If the call was not successful, an error code is returned.**

**ERROR CONDITIONS**

**Message qid is invalid.**

**Message queue not created from local node.**

**NOTES**

**Cannot be called from within an ISR.**

**May cause a preempt if a task waiting at the message queue has a higher priority than the running task, and the preempt mode is in effect. A preempt will not occur if all tasks waiting at the message queue exist on a remote processor in a multiprocessor configuration.**

**3.2.8 Q\_SEND**

**NAME**

**q\_send - “Send a Message to a Message Queue”**

**SYNOPSIS**

**#include <message.h>**

**uint q\_send ( qid, buffer )**

 **uint qid; /\* message queue id returned from q\_create or q\_ident \*/**

 **long (\*buffer)[4]; /\* pointer to message buffer \*/**

**DESCRIPTION**

**The q\_send directive sends a message to the queue identified by the qid.**

**If a task is already waiting at the queue, the message is copied to that task’s indicated receiving buffer. The waiting task is then made ready. If there is no task waiting, the message is copied to a system message buffer which is then placed at the end of the message queue.**

**Once sent, the task’s message buffer may be reused immediately. A message is fixed length, 16-bytes.**

**The message queue may exist on the local processor or any remote processor in a multiprocessor configuration, as long as the queue was created with the GLOBAL flags value set (see q\_create ).**

**RETURN VALUE**

**If the q\_send directive successfully sent a message, then 0 is returned.**

**If the call was not successful, an error code is returned.**

**ERROR CONDITIONS**

**Message qid is invalid.**

**Out of system message buffers.**

**Message queue at maximum count.**

**ISR cannot reference remote node.**

**NOTES**

**Can be called from within an ISR, except when the queue was not created from the local node.**

**May cause a preempt if a task waiting at the message queue has a higher priority than the running task, and the preempt mode is in effect. A preempt will not occur if a task waiting exists on a remote processor in a multiprocessor configuration.**