Event Chaining

Purpose

• Simplify thread activation in situations involving multiple independent events
• Reduce the number of threads and associated resources
• Permits a thread to wait on multiple resources by using notification functions
Notification Functions

- Notification capabilities for:
  - setting flags
  - sending a message to a queue
  - putting a semaphore
  - thread entry and exit
- Notification service registers an application function
- When event occurs, ThreadX invokes that application function

Registering a Function

Example: Queue Send Notification

```c
TX_QUEUE my_queue;

/* Register the "my_queue_send_notify" function for monitoring messages sent to the queue "my_queue." */
status = tx_queue_send_notify(&my_queue, my_queue_send_notify);

/* If status is TX_SUCCESS the queue send notification function was successfully registered. */

void my_queue_send_notify(TX_QUEUE *queue_ptr)
{
    /* A message was just sent to this queue! */
    /* Typically a semaphore put service would be placed here */
}
```
Notification Function Example

Example: Queue Send Notification (continued)

```
  tx_queue_send_notify
```

This service registers a function such as `my_queue_send_notify` for a queue such as `my_queue`. Whenever a message is sent to that queue, the notification function is invoked.

When this notification function is invoked, typically a `semaphore_put` service is executed. The purpose of this semaphore is to inform waiting threads that a message has been placed on the queue.

More Notification Examples

```
  tx_queue_send_notify (&my_queue, my_queue_send_notify);
  (registers the my_queue_send_notify notification function)

  tx_event_flags_set_notify (&my_group, my_event_flags_set_notify);
  (registers the my_event_flags_set_notify notification function)

  tx_semaphore_put_notify (&my_semaphore, my_semaphore_put_notify);
  (registers the my_semaphore_put_notify notification function)

  tx_thread_entry_exit_notify (&my_thread, my_entry_exit_notify);
  (registers the my_entry_exit_notify notification function)

  tx_thread_stack_error_notify (my_stack_error_handler);
  (registers the my_stack_error_handler notification function; note that TX_ENABLE_STACK_CHECKING must be defined)
```
Illustrative Example

A thread needs a message from either `queue_1` or `queue_2` in order to proceed. As indicated in the following diagram, that thread suspends until a message appears on either one of the two queues.

Solution Setup

One solution to this problem is to have the thread suspend on a semaphore that we arbitrarily name `gateway`. When the semaphore count exceeds one, this means that a message appears on one of the queues. As long as the semaphore count is zero, this means that no message appears on either queue.
Solution: Resources Needed

- Two queues: `queue_1` and `queue_2`
- One semaphore: `gateway`
- One thread
- Two queue send notification functions for `queue_1` and `queue_2`
- Register the notification functions
- Thread code that suspends on `gateway` and receives a message from one of the two queues when available

Solution: Code Definitions

```
TX_QUEUE queue_1, queue_2;
TX_SEMAPHORE gateway;

/* Register the notification functions for monitoring messages sent to queues "queue_1" and "queue_2" */
tx_queue_send_notify(&queue_1, queue_1_send_notify);
tx_queue_send_notify(&queue_2, queue_2_send_notify);
```
Solution: Code Functions

/* Notification functions to increment semaphore gateway whenever a message is sent to either queue_1 or queue_2 */

void queue_1_send_notify(TX_QUEUE *queue_ptr)
{
    /* A message was just sent to queue_1 */
    TX_SEMAPHORE_PUT (gateway);
}

void queue_2_send_notify(TX_QUEUE *queue_ptr)
{
    /* A message was just sent to queue_2 */
    TX_SEMAPHORE_PUT (gateway);
}

Solution: Thread Pseudocode

wait forever to get an instance from the semaphore named gatekeeper
/* an instance on this semaphore means that there is at least one message available on one of the two queues */

if a message appears on queue_1, receive it.
else
    receive a message from queue_2.
Solution: Thread Code

/* wait for a message to appear on queue_1 or queue_2 */
tx_semaphore_get (&gatekeeper, TX_WAIT_FOREVER);

/* Determine which queue has a message available */
status = tx_queue_receive (&queue_1, received_message,
    TX_NO_WAIT);

if (status == TX_SUCCESS)
    /* A message has been received from queue_1 */
else
    /* Receive a message from queue_2 */
    tx_queue_receive (&queue_2, received_message,
        TX_NO_WAIT);

Summary

• Event chaining provides an elegant solution to the multiple object suspension problem
• In the illustrative example, a thread suspended on two queues, but it could have suspended on any two or more objects