# mCertikOS Function Dependencies

### Memory Management

1. **container\_alloc(size\_t size):**
   * **Usage:**
     + message.c: Used in message\_create to allocate memory for the message\_t structure.
     + topic.c: Used in topic\_subscribe to allocate memory for the subscriber\_t structure.
     + queue.c (or equivalent): Used within the queue implementation (e.g., in queue\_create or when allocating queue nodes) to allocate memory for the message queue's internal data structures.
   * **Purpose:** Allocates a block of memory of the specified size from the current process's container. This ensures that memory usage is tracked and limited according to mCertikOS's resource quotas, preventing a process from consuming excessive memory.
   * **Importance:** *Critical*. Using container\_alloc is essential for adhering to mCertikOS's memory management policy and preventing memory exhaustion.
2. **container\_free(void \*ptr):**
   * **Usage:**
     + message.c: Used when dequeue the oldest message when the queue is full, or destroy the message if enqueue fails, or when the message is delivered successfully inside trap\_handler.
     + topic.c: Used in topic\_unsubscribe to free the memory allocated for the subscriber\_t structure and message queue.
     + queue.c (or equivalent): Used within the queue implementation (e.g., in queue\_destroy or when dequeuing messages) to free the memory associated with queue nodes.
   * **Purpose:** Releases a previously allocated memory block pointed to by ptr back to the container.
   * **Importance:** *Critical*. Failing to container\_free allocated memory will lead to memory leaks, eventually causing the system to run out of memory.

### Process Management

1. **current\_process\_id():**
   * **Usage:**
     + topic.c: Used in topic\_subscribe to get the process ID of the subscriber and store it in the subscriber\_t structure.
   * **Purpose:** Returns the unique identifier (process ID) of the currently executing process.
   * **Importance:** *Critical*. This is necessary to identify the subscriber process so that traps can be directed to the correct process context for callback execution.
2. **Potentially thread\_create() (If Asynchronous Callbacks are Used):**
   * **Usage:** *Potentially* used within the trap handler or a separate message delivery mechanism if callbacks are executed in separate threads to avoid blocking. This isn't strictly required for the *simplified* version if callbacks are executed directly within the trap handler (though this is less desirable).
   * **Purpose:** Creates a new thread within the current process.
   * **Importance:** *Optional for the simplified version, but highly recommended for a robust implementation*. Asynchronous callbacks prevent the publisher from being blocked by long-running or faulty callback functions.

### Trap Handling

1. **trap\_set\_handler(int trap\_number, void (\*handler)(int)):**
   * **Usage:**
     + trap.c: Used in trap\_init() to register the trap\_handler function with the mCertikOS kernel for a specific trap number.
   * **Purpose:** Registers a function (handler) to be executed when a specific trap (trap\_number) occurs. This allows the kernel to redirect control to the appropriate handler when a trap is triggered.
   * **Importance:** *Critical*. This is the core function that connects the trap mechanism to the message delivery process.
2. **trap\_send(int pid):**
   * **Usage:**
     + syscall.c: Used in sys\_pub to trigger a trap to the subscriber process (identified by pid) after a message has been enqueued.
   * **Purpose:** Sends a trap signal to the process with the specified process ID (pid). This initiates the execution of the registered trap handler in the context of the target process. The specific implementation will vary, but this will likely involve using a system call or kernel-level function to signal the process.
   * **Importance:** *Critical*. This function is responsible for initiating the message delivery to subscribers.

### Synchronization Primitives

1. **spinlock\_init(spinlock\_t \*lock):**
   * **Usage:**
     + topic.c: Used in topic\_init to initialize the spinlock goodbye\_topic.lock, also used inside topic\_subscribe to initialize the spinlock subscriber->lock.
   * **Purpose:** Initializes a spinlock.
   * **Importance:** *Critical*. Initializes the lock before use.
2. **spinlock\_lock(spinlock\_t \*lock):**
   * **Usage:**
     + topic.c: Used in topic\_subscribe before modifying the global goodbye\_topic data, used in topic\_unsubscribe before unsubscribing the current subscriber, used in message\_enqueue before enqueueing the message into the message queue.
     + syscall.c: Used in sys\_pub before sending trap signal
   * **Purpose:** Acquires a spinlock, waiting (spinning) until the lock is free.
   * **Importance:** *Critical*. Prevents race conditions and ensures data consistency when multiple processes or threads access shared data structures.
3. **spinlock\_unlock(spinlock\_t \*lock):**
   * **Usage:**
     + topic.c: Used in topic\_subscribe after modifying the global goodbye\_topic data, used in topic\_unsubscribe after unsubscribing the current subscriber, used in message\_enqueue after enqueueing the message into the message queue.
     + syscall.c: Used in sys\_pub after sending trap signal
   * **Purpose:** Releases a spinlock, allowing other processes or threads to acquire it.
   * **Importance:** *Critical*. Releasing the lock is essential to allow other processes to make progress. Failing to release a lock will lead to deadlocks.