# Implementing Pub/Sub IPC in mCertikOS

## Understanding of Features

* **Topic Management**: Unique identifiers for message channels, with publishers sending messages to specific topics.
* **Subscriber Registration**: Processes can subscribe to topics, with optional message queuing (e.g., buffer size limits).
* **Message Broadcasting**: Publishers send messages to all active subscribers of a topic, ensuring non-blocking delivery.
* **Callback Mechanism**: Subscribers execute user-defined functions upon message receipt.
* **Resource Management**: Efficient allocation of memory for messages and queues, adhering to mCertikOS’s container-based resource quotas

## Implementation Plan

### **Data Structure Design**

* **Topic Registry**: A hashmap mapping topic names to subscriber lists and message queues.
* **Subscriber Structure**: Each subscriber entry contains a process ID, callback function pointer, and a message queue.
* **Message Queue**: A circular buffer or linked list managed per subscriber, with configurable size (e.g., 1000 messages).

### **Layered Architecture**

#### **Layer 1: Message Management**

* **Functions**:
  + message\_create: Allocates memory for a message (using container\_alloc).
  + message\_enqueue: Adds a message to a subscriber’s queue, discarding old messages if the queue is full.
  + message\_dequeue: Retrieves the next message for processing.
* **Dependencies**:
  + **Container System**: Tracks memory usage per process (e.g., container\_can\_consume to prevent overallocation).
  + **Virtual Memory**: Uses page tables (MPTOp layer) for memory mapping.

#### **Layer 2: Topic Management**

* **Functions**:
  + topic\_create: Registers a new topic in the registry.
  + topic\_subscribe: Adds a subscriber to a topic’s list, initializing their message queue.
  + topic\_unsubscribe: Removes a subscriber and frees associated resources.
* **Concurrency Control**: Use spin locks or semaphores from mCertikOS’s synchronization primitives to handle concurrent subscriptions/unsubscriptions.

#### **Layer 3: Syscall Interface**

* **New Syscalls**:
  + sys\_pub: Publishes a message to a topic.
  + sys\_sub: Subscribes to a topic, specifying a callback and queue size.
  + sys\_unsub: Unsubscribes from a topic.
* **Integration**:
  + Modify the syscall.h header and syscall.c to include new syscalls.
  + Implement argument validation (e.g., topic existence, valid callback addresses).

#### **Layer 4: Callback Execution**

* **Trap Handling**: Use mCertikOS’s trap handling infrastructure (Lab 31) to schedule callback execution in user space.
  + When a message is enqueued, trigger a trap to the subscriber’s process, invoking the callback.
  + Ensure callbacks execute in a non-blocking manner (e.g., via separate threads or interrupts).

## Key mCertikOS Functions to Leverage

### Memory Management

* **Container System**: Use container\_alloc/container\_free to manage memory for messages and queues, adhering to resource quotas.
* **Page Tables**: Utilize virtual memory layers (e.g., MPTOp, MPTComm) to map message data structures into user space.

### Process Management

* **Thread Creation**: Use existing process/thread management functions (e.g., thread\_create) to handle concurrent callback execution.
* **Trap Handling**: Implement message delivery via trap handlers (e.g., trap\_init, trap\_set\_handler from Lab 3).

### Synchronization Primitives

* **Locks**: Implement spin locks or semaphores (e.g., using x86 atomic operations) to protect shared data structures like topic registries.

## Roadmap for Implementation

### Phase 1: Core Data Structures and Memory Management

* **Define Topic Registry**: Implement a hash map for topics, storing subscriber lists and queues.
* **Implement Message Queues**: Use circular buffers or linked lists, managed via container\_alloc for memory allocation.
* **Integrate Container System**: Ensure each process’s message queue adheres to its resource quota1.

### Phase 2: Topic and Subscriber Management

* **Implement topic\_create**: Add a new topic to the registry.
* **Implement topic\_subscribe**:
  + Validate topic existence.
  + Allocate a message queue for the subscriber.
  + Store the subscriber’s callback and queue size.
* **Implement topic\_unsubscribe**:
  + Remove the subscriber from the topic’s list.
  + Free the associated message queue and resources.

### Phase 3: Syscall Integration

* **Add Syscall Definitions**: Modify syscall.h to include sys\_pub, sys\_sub, and sys\_unsub.
* **Implement Syscall Handlers**:
  + **sys\_pub**: Retrieve the message, iterate over subscribers, and enqueue the message.
  + **sys\_sub/sys\_unsub**: Manage subscriber registrations.
* **Argument Validation**: Ensure topics exist before allowing operations.

### Phase 4: Message Delivery and Callbacks

* **Implement Message Broadcasting**:
  + For each subscriber, enqueue the message. If the queue is full, discard the oldest message.
* **Trigger Callback Execution**: Use mCertikOS’s trap handling (Lab 31) to schedule the callback in user space.
  + When a message is enqueued, send an interrupt or trigger a trap to the subscriber.
  + Execute the callback in a non-blocking thread or via asynchronous traps.

## 

## Technical Challenges and Mitigations

### Challenge 1: Atomic Operations for Shared Data

* **Solution**: Use x86 atomic instructions or spin locks to protect topic registries and queues during concurrent access.
* **Example**: Use lock xadd to increment subscriber counts atomically.

### Challenge 2: Message Delivery Deadlocks

* **Solution**: Ensure callbacks execute in a non-blocking manner. Use separate threads or prioritize message delivery via traps without blocking the CPU.

### Challenge 3: Memory Overallocation

* **Solution**: Strictly enforce resource quotas using mCertikOS’s container system (container\_can\_consume before allocation1).