Implementation code pub/sub in mCertikOS

1. Core Data Structures (pubsub.h)

```
// pubsub.h
#ifndef PUBSUB_H
#define PUBSUB_H

#include <stddef.h> // For size_t
#include <stdint.h> // For intptr_t

#define TOPIC_NAME "goodbye_topic"
#define MESSAGE_SIZE 64
#define QUEUE_SIZE 100 // Subscriber queue size

// Opaque types to avoid exposing internal structures
typedef struct topic_t topic_t;
typedef struct subscriber_t subscriber_t;

// Function pointer type for the callback
typedef void (*message_callback_t)(char* message);

// Function prototypes
int topic_create(const char *topic_name);
int topic_subscribe(const char *topic_name, message_callback_t callback, size_t
queue_size);
int topic_unsubscribe(const char *topic_name);
int sys_pub(const char *topic_name, char *message);

#endif // PUBSUB_H
```

2. Layer 1: Message Management (message.c)

```
// message.c
#include "pubsub.h"
#include "container.h" // Assuming container_alloc, container_free exist
#include <string.h> // For memcpy
#include <stdlib.h>
```

```
// Message structure
typedef struct message_t {
    char data[MESSAGE_SIZE];
} message_t;

// Function to create a message (allocates memory)
message_t* message_create(const char *message_data) {
    message_t* msg = (message_t*)container_alloc(sizeof(message_t));
    if (msg == NULL) {
        return NULL; // Allocation failed
    }
    strncpy(msg->data, message_data, MESSAGE_SIZE - 1); // Ensure null termination
    msg->data[MESSAGE_SIZE - 1] = '\0';
    return msg;
}

// Function to enqueue a message (implementation depends on queue type)
int message_enqueue(subscriber_t *subscriber, message_t *message);

// Function to dequeue a message (implementation depends on queue type)
message_t* message_dequeue(subscriber_t *subscriber);
```

3. Layer 2: Topic Management (topic.c)

```
// topic.c
#include "pubsub.h"
#include "queue.h"    // Assuming a queue implementation exists
#include "container.h" // For memory management
#include "synch.h"   // For spinlocks/semaphores

#include <stdbool.h> //For bool type
#include <stdlib.h>  //For NULL

// Structure for a subscriber
struct subscriber_t {
    int pid;                    // Process ID of the subscriber
    message_callback_t callback; // Callback function
    queue_t *message_queue;      // Message queue for the subscriber
    spinlock_t lock;        // Spinlock for queue access
};

// Structure for a topic
struct topic_t {
    char name[^32];           // Topic name
    subscriber_t *subscriber; // Single subscriber for simplicity
    bool subscriber_present; // Flag to indicate if there is a subscriber
    spinlock_t lock;        // Spinlock for subscriber management
};

// Global topic
static topic_t goodbye_topic;

// Initialize the topic
int topic_init() {
    strncpy(goodbye_topic.name, TOPIC_NAME, sizeof(goodbye_topic.name) - 1);
    goodbye_topic.name[sizeof(goodbye_topic.name) - 1] = '\0'; // Ensure null
termination
    goodbye_topic.subscriber = NULL;
    goodbye_topic.subscriber_present = false;
    spinlock_init(&goodbye_topic.lock);
    return 0;
}

int topic_create(const char *topic_name) {
    // In this simplified version, topic creation is not allowed.
    return -1;
}

int topic_subscribe(const char *topic_name, message_callback_t callback, size_t
queue_size) {
    if (strcmp(topic_name, TOPIC_NAME) != 0) {
        return -1; // Invalid topic
    }

    spinlock_lock(&goodbye_topic.lock);

    if (goodbye_topic.subscriber_present) {
        spinlock_unlock(&goodbye_topic.lock);
        return -1; // Only one subscriber allowed
    }

    // Allocate memory for the subscriber structure
    goodbye_topic.subscriber = (subscriber_t *)container_alloc(sizeof(subscriber_t));
    if (goodbye_topic.subscriber == NULL) {
        spinlock_unlock(&goodbye_topic.lock);
        return -1; // Memory allocation failure
    }

    // Initialize subscriber fields
    goodbye_topic.subscriber->pid = current_process_id(); // Get current process ID
    goodbye_topic.subscriber->callback = callback;
    spinlock_init(&goodbye_topic.subscriber->lock);

    //Allocate memory for the message queue
    goodbye_topic.subscriber->message_queue = queue_create(queue_size);
    if (goodbye_topic.subscriber->message_queue == NULL) {
        container_free(goodbye_topic.subscriber); // Free allocated memory
        goodbye_topic.subscriber = NULL;
        spinlock_unlock(&goodbye_topic.lock);
        return -1; //Queue creation fails
    }

    goodbye_topic.subscriber_present = true;

    spinlock_unlock(&goodbye_topic.lock);
    return 0;
}

int topic_unsubscribe(const char *topic_name) {
    if (strcmp(topic_name, TOPIC_NAME) != 0) {
        return -1; // Invalid topic
    }

    spinlock_lock(&goodbye_topic.lock);

    if (!goodbye_topic.subscriber_present) {
        spinlock_unlock(&goodbye_topic.lock);
        return -1; // No subscriber
```

```c
    }

    // Free resources
    queue_destroy(goodbye_topic.subscriber->message_queue);
    container_free(goodbye_topic.subscriber);
    goodbye_topic.subscriber = NULL;
    goodbye_topic.subscriber_present = false;

    spinlock_unlock(&goodbye_topic.lock);
    return 0;
}

// Enqueue message to the subscriber, returns 0 on success and -1 on failure
int message_enqueue(subscriber_t *subscriber, message_t *message) {
    if (subscriber == NULL || message == NULL) {
        return -1;
    }

    spinlock_lock(&subscriber->lock);
    if (queue_is_full(subscriber->message_queue)) {
        // Dequeue the oldest message
        message_t *old_message = queue_dequeue(subscriber->message_queue);
        container_free(old_message);
    }

    if (queue_enqueue(subscriber->message_queue, message) != 0) {
        spinlock_unlock(&subscriber->lock);
        return -1; // Enqueue failed
    }
    spinlock_unlock(&subscriber->lock);

    return 0;
}

// Dequeue message to the subscriber, returns the pointer to message, or NULL if empty
message_t *message_dequeue(subscriber_t *subscriber) {
    spinlock_lock(&subscriber->lock);
    message_t *message = queue_dequeue(subscriber->message_queue);
    spinlock_unlock(&subscriber->lock);
    return message;
```

```
}
```

4. Layer 3: Syscall Interface (syscall.c)

```
// syscall.c
#include "pubsub.h"
#include "topic.h"      // Topic management functions
#include "message.h"    // Message management functions
#include "trap.h"       // Trap handling functions
#include "container.h"  // Container functions
#include <string.h>

// Syscall implementations

long sys_sub(const char *topic_name, message_callback_t callback, size_t queue_size) {
    return topic_subscribe(topic_name, callback, queue_size);
}

long sys_unsub(const char *topic_name) {
    return topic_unsubscribe(topic_name);
}

long sys_pub(const char *topic_name, char *message_data) {
    if (strcmp(topic_name, TOPIC_NAME) != 0) {
        return -1; // Invalid topic
    }

    // Check if there is a subscriber and get the subscriber
    spinlock_lock(&goodbye_topic.lock);
    if (!goodbye_topic.subscriber_present) {
        spinlock_unlock(&goodbye_topic.lock);
        return -1; // No subscriber
    }
    subscriber_t *subscriber = goodbye_topic.subscriber;
    spinlock_unlock(&goodbye_topic.lock);

    // Create message
    message_t *message = message_create(message_data);
    if (message == NULL) {
        return -1; // Message creation failed
    }

    // Enqueue the message
    if (message_enqueue(subscriber, message) != 0) {
        container_free(message); // Free the message
        return -1;              // Enqueue failed
    }

    // Trigger callback execution (using trap handling)
    trap_send(subscriber->pid); // Assuming trap_send takes the process ID

    return 0;
}
```

5. Layer 4: Callback Execution (trap.c, and modifications to subscriber.c)

This part is the most complex and requires careful integration with mCertikOS's trap handling mechanism.

```
// trap.c (Example - adjust based on mCertikOS's trap API)
#include "trap.h"
#include "pubsub.h"
#include "message.h"
#include "topic.h"

// Trap handler function
void trap_handler(int trap_number) {
    // Get the current process ID
    int pid = current_process_id();

    spinlock_lock(&goodbye_topic.lock);
    if (!goodbye_topic.subscriber_present) {
        spinlock_unlock(&goodbye_topic.lock);
        return; // No subscriber
    }

    subscriber_t *subscriber = goodbye_topic.subscriber;
    if (subscriber->pid != pid) {
        spinlock_unlock(&goodbye_topic.lock);
        return; // Not the subscriber's trap
    }
    spinlock_unlock(&goodbye_topic.lock);

    // Dequeue the message
    message_t *message = message_dequeue(subscriber);
    if (message != NULL) {
        // Call the callback function
        (subscriber->callback)(message->data);

        // Free the message
        container_free(message);
    }
}

//Initialize trap
void trap_init(){
    trap_set_handler(TRAP_NUMBER, trap_handler); //TRAP_NUMBER is the trap number
}

// Send trap signal to the specific pid
void trap_send(int pid){
    //Send signal to the specified pid
}
```

And in `subscriber.c`, you'll need to initialize the trap handler:

```
// Subscriber (subscriber.c)
#include "pubsub.h"
#include <stdio.h> // For demonstration; replace with mCertikOS's print function
#include <stdlib.h>

// mCertikOS system call definitions (replace with actual syscall numbers)
#define SYS_SUB 101
#define SYS_UNSUB 102

void message_callback(char* message) {
    printf("Received: %s\n", message);
}
```

```
int main() {
    // mCertikOS system call to subscribe
    long result = syscall(SYS_SUB, TOPIC_NAME, message_callback, QUEUE_SIZE);

    if (result < 0) {
        printf("Error subscribing.\n");
        return 1;
    }

    // Initialize the trap handler
    trap_init();

    // Keep the subscriber running to receive messages
    while (1) {
        sleep(1); // Or any other necessary wait
    }

    //Unsubscribe
    syscall(SYS_UNSUB, TOPIC_NAME);
    return 0;
}
```

6. Publisher Implementation (publisher.c)

```
// Publisher (publisher.c)
#include "pubsub.h"
#include <stdio.h> // For demonstration; replace with mCertikOS's print function

// mCertikOS system call definitions (replace with actual syscall numbers)
#define SYS_PUB 100

int main() {
    int i = 0;
    char message[MESSAGE_SIZE];

    //Initialize the topic
    topic_init();
    while (1) {
        i++;
        snprintf(message, MESSAGE_SIZE, "Goodbye World %d", i);

        // mCertikOS system call to publish
        long result = syscall(SYS_PUB, TOPIC_NAME, message);

        if (result < 0) {
            // Handle error (e.g., no subscriber)
            printf("Error publishing message.\n");
        }

        // Delay (replace with mCertikOS's sleep function)
        sleep(1); // Sleep for 1 second
    }

    return 0;
}
```