# 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  
 }  
  
 // 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;  
}