WSL & mCertikOS

Project 12: Pub/Sub IPC

Created @May 4, 2025

Setup WSL for mCertikOS

Once you have your WSL2 & Ubuntu 24.04 installed on your windows machine, we need to setup the WSL to be able to run mCertikOS.

- `sudo apt-get install -y build-essential gcc-multilib gdb`
- `apt-get install qemu-system`
- `sudo apt-get update && sudo apt-get install -y parted`

File Permissions

Open the `mcertikOs` folder(download and unzip form the google drive link provided by `Sani Sir`) inside `VSCode` and then run the following commands in a terminal -

- `chmod +x make_image.py`
- `chmod +x scripts/set_auto_load.sh`

Build & Run

- `make clean`
- `make all`
- `make qemu` or `make qemu-nox`

Interacting with mCertikOs

After the `make qemu` or `make qemu-nox` is successful, it should show a QEMU emulator window open, as well as the ability to type commands in the VS code terminal. We can type in `help` to see the available Shell commands.

- `help`
- `kerninfo`
- `startuser`
- `pubsub`
- `test_pubsub`

`startuser`

This command starts the user idle process, which is essentially the first user-space process in the operating system. It:

- Creates a new process for the idle program
- Sets up the process state as running
- Switches the context to run the user process
- This is typically used to transition from kernel mode to user mode

test_pubsub

This is a test command that automatically:

- Subscribes to a test topic
- Publishes a test message
- Demonstrates the basic functionality of the Pub/Sub system

Usage

```
$> test_pubsub
```

This will automatically:

- Subscribe to "`test_topic`"
- Publish a message "Hello, this is a test message!"
- Show you the received message

pubsub

This is a command to interact with the Pub/Sub IPC system. It has two subcommands:

- `pubsub publish <topic> <message>`: Publishes a message to a specific topic
- `pubsub subscribe <topic>`: Subscribes to a specific topic to receive messages

Usage

```
$> pubsub subscribe test_topic
```

- This will subscribe to "test_topic" and set up a callback to receive messages.

Then in another terminal or after subscribing run the `make qemu` again and then →

`$> pubsub publish test_topic "Your message here”`

- This will publish a message to the topic, and all subscribers will receive it.
- `NOTE -` This will show an error, as we can NOT run `make qemu` or `make qemu-nox` for the same `certikos.img` image, while it is being used in the previous subscribe terminal. The error encountered is a safety feature of QEMU to prevent multiple instances from corrupting the disk image.
- So we need to use the same qemu terminal instance to run both the commands, which is already handled as - the Pub/Sub system is implemented within the kernel itself (as we can see from the code in `kern/ipc/`), and all communication happens within the same kernel instance. We don't need separate QEMU instances to test the Pub/Sub functionality.
- So, basically, run the subscribe command first, and then run the publish command, the terminal will show the received message.

Important notes

1. The Pub/Sub system is an inter-process communication mechanism that allows processes to communicate through topics
2. Messages are limited to 256 characters (as defined in `pubsub.h`)
3. The system supports up to 10 topics and 100 subscribers per topic
4. The `startuser` command is typically used once to transition from kernel mode to user mode
5. All commands are case-sensitive
6. The system will show debug messages (`dprintf`) to indicate the success or failure of operations

Later Updates

Complete `monitor.c` file

```
// Simple command-line kernel monitor useful for
// controlling the kernel and exploring the system interactively.

#include <lib/debug.h>
#include <lib/elf.h>
#include <lib/types.h>
#include <lib/gcc.h>
#include <lib/string.h>
#include <lib/x86.h>
#include <lib/thread.h>
#include <lib/monitor.h>
#include <dev/console.h>
#include <vmm/MPTIntro/export.h>
#include <vmm/MPTNew/export.h>
#include <kern/ipc/pubsub.h>
#include <kern/ipc/publisher.c>
#include <kern/ipc/subscriber.c>

#define CMDBUF_SIZE 80  // enough for one VGA text line

struct Command {
    const char *name;
    const char *desc;

    int (*func)(int argc, char **argv, struct Trapframe *tf);
};


/*****  Subscriber callback for displaying messages *****/
void topic_callback(struct Message *msg) {
    dprintf("Message received: %s\n", msg->data);
}
/*****  Subscriber callback for displaying messages *****/


/***** Implementations of basic kernel monitor commands *****/
int mon_kerninfo(int argc, char **argv, struct Trapframe *tf)
{
```

```
    extern uint8_t start[], etext[], edata[], end[];

    dprintf("Special kernel symbols:\n");
    dprintf("  start  %08x\n", start);
    dprintf("  etext  %08x\n", etext);
    dprintf("  edata  %08x\n", edata);
    dprintf("  end    %08x\n", end);
    dprintf("Kernel executable memory footprint: %dKB\n",
            ROUNDUP(end - start, 1024) / 1024);
    return 0;
}

// Command to subscribe to a topic
int mon_topic_sub(int argc, char **argv, struct Trapframe *tf) {
    if (argc < 2) {
        dprintf("Usage: topic_sub <topic_name>\n");
        return 0;
    }

    // Check if already subscribed
    struct Topic *topic = find_topic(argv[1]);
    bool already_subscribed = FALSE;
    if (topic != NULL) {
        for (int i = 0; i < topic->num_subscribers; i++) {
            if (topic->subscribers[i].callback == topic_callback) {
                already_subscribed = TRUE;
                break;
            }
        }
    }

    if (already_subscribed) {
        dprintf("Already subscribed to topic '%s'\n", argv[1]);
        return 0;
    }

    // Subscribe to the topic
    subscribe(argv[1], topic_callback);
```

```
    dprintf("Subscribed to topic '%s'\n", argv[1]);

    // Show existing messages if any
    if (topic != NULL && topic->count > 0) {
        dprintf("Existing messages in topic '%s':\n", argv[1]);
        int idx = topic->head;
        for (int i = 0; i < topic->count; i++) {
            dprintf("[%d] %s\n", i+1, topic->queue[idx].data);
            idx = (idx + 1) % MAX_QUEUE_SIZE;
        }
    } else {
        dprintf("No messages in topic '%s'\n", argv[1]);
    }

    return 0;
}

// Command to unsubscribe from a topic
int mon_topic_unsub(int argc, char **argv, struct Trapframe *tf) {
    if (argc < 2) {
        dprintf("Usage: topic_unsub <topic_name>\n");
        return 0;
    }

    // Check if subscribed
    struct Topic *topic = find_topic(argv[1]);
    if (topic == NULL) {
        dprintf("Topic '%s' does not exist\n", argv[1]);
        return 0;
    }

    bool was_subscribed = FALSE;
    for (int i = 0; i < topic->num_subscribers; i++) {
        if (topic->subscribers[i].callback == topic_callback) {
            was_subscribed = TRUE;
            break;
        }
    }
```

```
    if (!was_subscribed) {
        dprintf("Not subscribed to topic '%s'\n", argv[1]);
        return 0;
    }

    // Unsubscribe from the topic
    unsubscribe(argv[1], topic_callback);
    dprintf("Unsubscribed from topic '%s'\n", argv[1]);

    return 0;
}

// Command to publish to a topic
int mon_topic_pub(int argc, char **argv, struct Trapframe *tf) {
    if (argc < 3) {
        dprintf("Usage: topic_pub <topic_name> <message>\n");
        return 0;
    }

    publish(argv[1], argv[2]);
    dprintf("Published to topic '%s': %s\n", argv[1], argv[2]);
    return 0;
}

// Command to list topics
int mon_topic_ps(int argc, char **argv, struct Trapframe *tf) {
    dprintf("Available topics:\n");
    for (int i = 0; i < num_topics; i++) {
        dprintf("- %s (%d subscribers, %d messages in queue)\n",
                topics[i].name,
                topics[i].num_subscribers,
                topics[i].count);
    }

    if (num_topics == 0) {
        dprintf("No topics available\n");
    }
```

```
    return 0;
}

// Command to list publishers
int mon_publisher_ps(int argc, char **argv, struct Trapframe *tf) {
    // In this simple implementation, we don't track publishers separately
    // So we'll just show topics with messages
    dprintf("Active publishers (topics with messages):\n");
    int count = 0;

    for (int i = 0; i < num_topics; i++) {
        if (topics[i].count > 0) {
            dprintf("- %s (%d messages)\n", topics[i].name, topics[i].count);
            count++;
        }
    }

    if (count == 0) {
        dprintf("No active publishers\n");
    }
    return 0;
}

// Command to list messages in a topic
int mon_message_ps(int argc, char **argv, struct Trapframe *tf) {
    if (argc != 2 || argv[1][0] == '\0') {
        dprintf("Usage: message_ps <topic>\n");
        return 0;
    }

    struct Topic *topic = find_topic(argv[1]);
    if (!topic) {
        dprintf("Error: Topic '%s' not found\n", argv[1]);
        return 0;
    }

    if (topic->count == 0) {
        dprintf("Topic '%s' has no messages\n", argv[1]);
```

```
        return 0;
    }

    dprintf("Messages in '%s' (%d messages, %d subscribers):\n",
            argv[1], topic->count, topic->num_subscribers);

    int idx = topic->head;
    for (int i = 0; i < topic->count; i++) {
        dprintf("[%04d] %s\n", i+1, topic->queue[idx].data);
        idx = (idx + 1) % MAX_QUEUE_SIZE;
    }

    return 0;
}



// Command to exit the monitor
int mon_exit(int argc, char **argv, struct Trapframe *tf) {
    dprintf("Exiting kernel monitor.\n");
    return -1;
}
/***** Implementations of basic kernel monitor commands *****/


/***** Monitor commands *****/
static struct Command commands[] = {
    {"help", "Display this list of commands", mon_help},
    {"kerninfo", "Display information about the kernel", mon_kerninfo},
    {"topic_pub", "Publish to a topic", mon_topic_pub},
    {"topic_sub", "Subscribe to a topic", mon_topic_sub},
    {"topic_unsub", "Unsubscribe from a topic", mon_topic_unsub},
    {"topic_ps", "List available topics", mon_topic_ps},
    {"publisher_ps", "List active publishers", mon_publisher_ps},
    {"message_ps", "List messages in a topic", mon_message_ps},
    {"exit", "Exit the kernel monitor", mon_exit},
};
```

```
#define NCOMMANDS (sizeof(commands) / sizeof(commands[0]))


int mon_help(int argc, char **argv, struct Trapframe *tf)
{
    int i;

    for (i = 0; i < NCOMMANDS; i++)
        dprintf("%s - %s\n", commands[i].name, commands[i].desc);
    return 0;
}
/***** Monitor commands *****/


extern uint8_t _binary___obj_user_idle_idle_start[];
extern unsigned int proc_create(void *elf_addr, unsigned int quota);
extern void tqueue_remove(unsigned int chid, unsigned int pid);
extern void tcb_set_state(unsigned int pid, unsigned int state);
extern void set_curid(unsigned int curid);
extern void kctx_switch(unsigned int from_pid, unsigned int to_pid);

/***** Kernel monitor command interpreter *****/
#define WHITESPACE "\t\r\n "
#define MAXARGS    16

/***** Kernel monitor command interpreter *****/
static int runcmd(char *buf, struct Trapframe *tf) {
    int argc;
    char *argv[MAXARGS];
    int i;

    // Parse the command buffer into whitespace-separated arguments
    argc = 0;
    argv[argc] = 0;
    while (1) {
        // gobble whitespace
        while (*buf && strchr(WHITESPACE, *buf))
```

```
            *buf++ = 0;
        if (*buf == 0)
            break;

        // Handle quoted arguments
        if (*buf == '"') {
            *buf++ = 0; // Remove opening quote
            argv[argc++] = buf;
            while (*buf && *buf != '"')
                buf++;
            if (*buf == '"')
                *buf++ = 0; // Remove closing quote
        } else {
            // Normal argument
            argv[argc++] = buf;
            while (*buf && !strchr(WHITESPACE, *buf))
                buf++;
        }

        if (argc == MAXARGS - 1) {
            dprintf("Too many arguments (max %d)\n", MAXARGS);
            return 0;
        }
    }
    argv[argc] = 0;

    // Rest of the function remains the same
    if (argc == 0)
        return 0;
    for (i = 0; i < NCOMMANDS; i++) {
        if (strcmp(argv[0], commands[i].name) == 0)
            return commands[i].func(argc, argv, tf);
    }
    dprintf("Unknown command '%s'\n", argv[0]);
    return 0;
}
```

```
/***** Kernel monitor command interpreter *****/


/***** Monitor Startup *****/
void monitor(struct Trapframe *tf)
{
   char *buf;

   dprintf("\n****************************************\n\n");
   dprintf("Welcome to the mCertiKOS kernel monitor!\n");
   dprintf("\n****************************************\n\n");
   dprintf("Type 'help' for a list of commands.\n");

   while (1) {
      buf = (char *) readline("$>> ");
      if (buf != NULL)
         if (runcmd(buf, tf) < 0)
            break;
   }
}
/***** Monitor Startup *****/
```

Complete `queue.c` file

```
#include <kern/ipc/pubsub.h>
#include <lib/string.h>
// Initialize the message queue
void init_message_queue(struct Topic *topic) {
   topic->head = 0;
   topic->tail = 0;
   topic->count = 0;
}
```

```
// Add a message to the queue
void enqueue_message(struct Topic *topic, const char *data) {
    if (topic->count == MAX_QUEUE_SIZE) {
        // Discard the oldest message
        topic->head = (topic->head + 1) % MAX_QUEUE_SIZE;
        topic->count--;
    }

    // Add the new message
    strncpy(topic->queue[topic->tail].data, data, MESSAGE_SIZE);
    topic->tail = (topic->tail + 1) % MAX_QUEUE_SIZE;
    topic->count++;
}

// Remove a message from the queue
void dequeue_message(struct Topic *topic) {
    if (topic->count > 0) {
        topic->head = (topic->head + 1) % MAX_QUEUE_SIZE;
        topic->count--;
    }
}

// List all messages in the queue for a given topic
int list_messages(const char *topic_name) {
    struct Topic *topic = find_topic(topic_name);
    if (topic == NULL) {
        dprintf("Topic '%s' not found\n", topic_name);
        return -1;
    }

    if (topic->count == 0) {
        dprintf("Topic '%s' has no messages\n", topic_name);
        return 0;
    }

    dprintf("Messages in topic '%s':\n", topic_name);
    int index = topic->head;
    for (int i = 0; i < topic->count; i++) {
```

```
        dprintf("  %d. %s\n", i+1, topic->queue[index].data);
        index = (index + 1) % MAX_QUEUE_SIZE;
    }
    return 0;
}
```

Complete `topic.c` file

```
#include <lib/string.h>
#include <kern/ipc/pubsub.h>

struct Topic topics[MAX_TOPICS];
int num_topics = 0;

// Find a topic by name
struct Topic *find_topic(const char *name) {
    for (int i = 0; i < num_topics; i++) {
        if (strcmp(topics[i].name, name) == 0) {
            return &topics[i];
        }
    }
    return NULL; // Topic not found
}

// Create a new topic
struct Topic *create_topic(const char *name) {
    if (num_topics >= MAX_TOPICS) {
        return NULL;
    }

    struct Topic *topic = &topics[num_topics++];
    strncpy(topic->name, name, TOPIC_NAME_LEN);
    topic->head = 0;
    topic->tail = 0;
    topic->count = 0;
    topic->num_subscribers = 0;
```

```
    return topic;
}



int list_topics(int argc, char **argv, struct Trapframe *tf) {
    if (num_topics == 0) {
        dprintf("No topics available.\n");
        return 0;
    }

    dprintf("Available topics (%d):\n", num_topics);
    for (int i = 0; i < num_topics; i++) {
        dprintf("  %d. %s (subscribers: %d, messages: %d)\n",
                i+1,
                topics[i].name,
                topics[i].num_subscribers,
                topics[i].count);
    }
    return 0;
}
```

Complete `subscriber.c` file

```
#include <lib/string.h>
#include <kern/ipc/pubsub.h>


// Subscribe to a topic
void subscribe(const char *topic_name, void (*callback)(struct Message *)) {
    struct Topic *topic = find_topic(topic_name);
    if (topic == NULL) {
        // Create the topic if it doesn't exist
        topic = create_topic(topic_name);
        if (topic == NULL) {
            return; // Failed to create topic
```

```
        }
    }

    // Add the subscriber to the list
    if (topic->num_subscribers >= MAX_SUBSCRIBERS) {
        return; // Too many subscribers
    }

    topic->subscribers[topic->num_subscribers++].callback = callback;
}

// Unsubscribe from a topic
void unsubscribe(const char *topic_name, void (*callback)(struct Message *)) {
    struct Topic *topic = find_topic(topic_name);
    if (topic == NULL) {
        return;
    }

    // Find and remove the subscriber
    for (int i = 0; i < topic->num_subscribers; i++) {
        if (topic->subscribers[i].callback == callback) {
            // Shift remaining subscribers down
            for (int j = i; j < topic->num_subscribers - 1; j++) {
                topic->subscribers[j] = topic->subscribers[j+1];
            }
            topic->num_subscribers--;
            break;
        }
    }

    // If no more subscribers, consider deleting the topic
    if (topic->num_subscribers == 0 && topic->count == 0) {
        // In a real implementation, we would remove the topic
    }
}
```

Complete `publisher.c` file

```
#include <kern/ipc/pubsub.h>
#include <lib/string.h>
#include <kern/ipc/topic.c>
#include <kern/ipc/queue.c>

// Publish a message to a topic
void publish(const char *topic_name, const char *message_data) {
    struct Topic *topic = find_topic(topic_name);
    if (topic == NULL) {
        // Create the topic if it doesn't exist
        topic = create_topic(topic_name);
        if (topic == NULL) {
            return; // Failed to create topic
        }
    }

    // Ensure the message is properly null-terminated
    char safe_message[MESSAGE_SIZE];
    strncpy(safe_message, message_data, MESSAGE_SIZE - 1);
    safe_message[MESSAGE_SIZE - 1] = '\0';

    // Add the message to the queue
    enqueue_message(topic, safe_message);

    // Notify all subscribers
    for (int i = 0; i < topic->num_subscribers; i++) {
        if (topic->subscribers[i].subscribed) {
            topic->subscribers[i].callback(&topic->queue[topic->head]);
        }
    }
}
```

Complete `pubsub.h` file

```
#ifndef KERN_IPC_PUBSUB_H
#define KERN_IPC_PUBSUB_H
```

```
#include <lib/types.h>

#define MAX_TOPICS 10
#define MAX_SUBSCRIBERS 100
#define MAX_QUEUE_SIZE 1000
#define TOPIC_NAME_LEN 32
#define MESSAGE_SIZE 256

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

// Subscriber structure
struct Subscriber {
    void (*callback)(struct Message *msg); // Callback function for message de
    char subscribed;                       // Flag to indicate if subscribed
};

// Topic structure
struct Topic {
    char name[TOPIC_NAME_LEN];             // Topic name
    struct Message queue[MAX_QUEUE_SIZE];  // Message queue
    int head;                              // Index of the oldest message
    int tail;                              // Index of the next free slot
    int count;                             // Number of messages in the queue
    struct Subscriber subscribers[MAX_SUBSCRIBERS]; // List of subscribers
    int num_subscribers;                   // Number of subscribers
};


// Global variables
extern struct Topic topics[MAX_TOPICS];
extern int num_topics;

// Function declarations
struct Topic *find_topic(const char *name);
```

```
struct Topic *create_topic(const char *name);
void enqueue_message(struct Topic *topic, const char *data);
void dequeue_message(struct Topic *topic);
void publish(const char *topic_name, const char *message_data);
void subscribe(const char *topic_name, void (*callback)(struct Message *));
void unsubscribe(const char *topic_name, void (*callback)(struct Message *))
int list_topics(int argc, char **argv, struct Trapframe *tf);
void list_publishers(void);
int list_messages(const char *topic_name);

#endif // KERN_IPC_PUBSUB_H
```

Commands for Demo - In this Sequence

- `topic_pub <topicname> “topic message”`
- `topic_ps`
- `publisher_ps`
- `message_ps <topicname>`
- `topic_sub <topicname>`
- `topic_unsub <topicname>`
- `exit`

COMMAND OUTPUT (cryinggggg 😭😭)

```
**************************************

Welcome to the mCertiKOS kernel monitor!

**************************************

Type 'help' for a list of commands.
$>>
```

```
$>>
$>>
$>>
$>> topic_pub test "Hello World"
Published to topic 'test': Hello World
$>>
$>>
$>> topic_pub test "Bye World"
Published to topic 'test': Bye World
$>>
$>>
$>> topic_pub test2 "Hi Hi"
Published to topic 'test2': Hi Hi
$>>
$>>
$>> topic_sub test
Subscribed to topic 'test'
Existing messages in topic 'test':
[1] Hello World
[2] Bye World
$>>
$>>
$>>
$>> topic_ps
Available topics:
- test (1 subscribers, 2 messages in queue)
- test2 (0 subscribers, 1 messages in queue)
$>>
$>>
$>> publisher_ps
Active publishers (topics with messages):
- test (2 messages)
- test2 (1 messages)
$>>
$>>
$>>
$>> message_ps test
Messages in 'test' (2 messages, 1 subscribers):
```

```
[0001] Hello World
[0002] Bye World
$>>
$>>
$>> topic_unsub test
Unsubscribed from topic 'test'
$>>
$>>
$>> topic_ps
Available topics:
- test (0 subscribers, 2 messages in queue)
- test2 (0 subscribers, 1 messages in queue)
$>>
$>>
$>> exit
$>>
$>> exit
Exiting kernel monitor.
```