kucomms kernel programmers guide

Table of Contents

Introduction....1
Defining and registering callbacks....1
Sending a message....2
Sending a message and locking....3
Writing work handlers....3

Introduction

In order for a kernel module to communicate with a userspace application, it is necessary to define three callback functions and then register those callback functions with the kucomms module.

Defining and registering callbacks

The first step is to declare the callback functions. The functions shown below have no implementation and are examples only.

```
static bool
my_message_hlr(const struct Message * message,
               MessageQueueHeaderPtr tx_msgq,
               const __u64 rx_msgq_queueLength,
               const __u64 tx_msgq_queueLength,
               void * userData)
{
        return true;
}

static bool
my_work_hlr(void * userData)
{
        return false;
}

static void
my_timer_hlr(const __u64 time, void * userData)
{
}
```

The next step is to register the callbacks with the kucomms module.

The user must register in the module init function and must unregister in the module exit function.

```
const char * devname = "kucomms_myname";

static int __init init_mymodule(void)
{
        bool ok = kucomms_register(
                devname,
                strlen(devname),
                my_message_hlr,
                my_work_hlr,
                my_timer_hlr,
                0);

        if (!ok) return -ENODEV;

        return 0;
}

static void __exit exit_mymodule(void)
{
        kucomms_unregister_wait(devname, strlen(devname));
}

module_init(init_mymodule);
module_exit(exit_mymodule);
```

Sending a message

The message_queue_add_tx0() function is used to send a message.

Below are some examples of sending a message.

```
static bool
my_message_hlr(const struct Message * message,
               MessageQueueHeaderPtr tx_msgq,
               const __u64 rx_msgq_queueLength,
               const __u64 tx_msgq_queueLength,
               void * userData)
{
        struct kucomms_file_data * pfd = (struct kucomms_file_data *)userData;

        // Send message received back to the sender.
        bool ok = message_queue_add_tx0(pfd, message);

        return true;
}

static bool
my_work_hlr(void * userData)
{
        struct kucomms_file_data * pfd = (struct kucomms_file_data *)userData;
        struct Message * message;
        __u64 dataLength;
        bool ok;

        dataLength = 10; // build a message
        message = vmalloc(message_get_message_length(dataLength));

        message->m_length = dataLength;
        message->m_type = 0;
        message->m_id = 0;
        message->m_userValue = 0;
        for (__u32 u0=0; u0<dataLength; u0++) message->m_data[u0] = 0;

        ok = message_queue_add_tx0(pfd, message); // send message

        vfree(message);

        return false;
}
```

Sending a message and locking

The function to send a message has two variants, a locked version and a unlocked version. If messages are to be sent only from callback handlers then the unlocked version can be called. If messages are to be sent from multiple thread contexts then the locked version must be used.

The unlocked version is called *message_queue_add_tx0()* and the locked version is called *message_queue_add_tx0_locked()*.

Writing work handlers

Work handlers are called at a rate of approximately 100 times a second. Work handlers should execute as quickly as possible. The average execution time of a work handler should not exceed 1/100 of a second. If a work handler does not sleep then it should return false. If a work handler sleeps for >= 1/100 of a second then it should return true.