

USING A PUBLIC MESSAGE QUEUE TO PASS MESSAGES ACROSS PROCESS BOUNDARIES

The typical use case has one task (the "Receiver") in some memory space (RTP or the kernel) that pends, waiting for a message from another task. A task in some other memory space (the "Sender") then places a message onto the queue, to be read by the Receiver.

TASK GUIDE

| Step | Action | Example |
|------|---|---|
| 1 | Create a project for each participant in the sharing of the queue. | |
| 2 | For the Receiver, the first task to execute, write code that does the following: a) Creates the queue and captures its ID b) Loops, pending on a message being placed onto the queue, then printing a message when it is received | The code will look something like this: <pre>MSG_Q_ID msgQId; int length; /*message length */ char *buffer; /* receive buffer */ buffer = malloc (100); msgQId = msgQOpen("/m ", 10, 100, MSG_Q_FIFO, OM_CREATE, 0); . . FOREVER{ length = msgQReceive(msgQId, buffer, 100, WAIT_FOREVER); printf ("%s\n", buffer); }</pre> |
| 3 | For the Sender, write code that does the following: a) Attaches to (gains access to) the queue and captures its ID b) Loops, sending a message every 5 seconds, for example | The code will look something like this: <pre>SEM_ID semId; char message[] = { "Hello World!\n" }; msgQId = msgQOpen("/msgQ", 0, 0, 0, 0, 0); . . FOREVER{ msgQSend (msgQId, (char *) message, strlen(message)+1, 0, 0) taskDelay (sysClkRateGet() * 5); }</pre> |

Key Points

- The msgQOpen () call is used either to create the public task initially or to gain access to it. The only difference is the OM_CREATE option.
- The example has been simplified to handle only string messages; the extension to other message types is straightforward.
- The basic dialog could also be extended to have multiple Senders (very common) and/or Receivers (less so). That extension is not covered here.

education.windriver.com – training@windriver.com