# Experiment #4
# Message Queues

## Introduction

In VxWorks, the primary intertask communication mechanism within a single CPU is *message queues*. Message queues allow a variable number of messages, each of variable length, to be queued(FIFO or priority based). Any task can send a message to a message queue and any task can receive a message from a message queue. Multiple tasks can send to and receive from the same message queue. Two way communication between two tasks generally requires two message queues, one for each direction.

## Objectives

The following are the primary objectives of this experiment:

- To demonstrate the use of VxWorks message queues.

## Description

Wind message queues are created and deleted with the following routines:

- *msgQCreate(int maxMsgs, int maxMsgLength, int options)*: Allocate and initialize a message queue.

- *msgQDelete(MSG_Q_ID msgQId)*: Terminate and free a message queue.

- *msgQSend(MSG_Q_ID msgQId, char *Buffer, UINT nBytes, int timeout, int priority)*: Send a message to a message queue.

- *msgQReceive(MSG_Q_ID msgQId, char *Buffer, UINT nBytes, int timeout)* Send a message to a message queue.

This library provides messages that are queued in FIFO order, with a single exception: there are two priority levels, and messages marked as high priority are attached to the head of the queue.
A message queue is created with *msgQCreate()*. Its parameters specify the maximum number of messages that can be queued in the message queue and the maximum length in bytes of each message.
A task sends a message to a message queue with *msgQSend()*. If no tasks are waiting for messages on that queue, the message is added to the queue's buffer of messages. If any tasks are waiting for a message from that message queue, the message is immediately delivered to the first waiting task.
A task receives a message from a message queue with *msgQReceive()*. If messages are already available in the queue's buffer, the first message is immediately dequeued and returned to the caller. If no messages are available, then the calling task blocks and is added to a queue of tasks waiting for messages. The queue of waiting tasks can be ordered either by task priority or FIFO, as specified when the queue is created.

- *Timeouts*: Both *msgQSend()* and *msgQReceive()* take timeout parameters. The timeout parameter specifies how many ticks(clock ticks per second) to wait for space to be available when sending a message and to wait for a message to be available when receiving a message.

- *Urgent Messages*: The *msgQSend()* function can specify the priority of a message either as normal MSG_PRI_NORMAL or urgent MSG_PRI_URGENT. Normal priority messages are added to the tail of the message queue, while urgent priority messages are added to the head of the message queue.

## 1. Example

```
-------------------------------------------------------------------------------------
/* includes */
#include "vxWorks.h"
#include "msgQLib.h"

/* function prototypes */
void taskOne(void);
void taskTwo(void);

/* defines */
#define MAX_MESSAGES 100
```

```
#define MAX_MESSAGE_LENGTH 50

/* globals */
MSG_Q_ID mesgQueueId;

void message(void) /* function to create the message queue and two tasks */
{
int taskIdOne, taskIdTwo;

/* create message queue */
if ((mesgQueueId = msgQCreate(MAX_MESSAGES,MAX_MESSAGE_LENGTH,MSG_Q_FIFO))
         == NULL)
         printf("msgQCreate in failed\n");


/* spawn the two tasks that will use the message queue */
if((taskIdOne = taskSpawn("t1",90,0x100,2000,(FUNCPTR)taskOne,0,0,0,0,0,0,0,
         0,0,0)) == ERROR)
         printf("taskSpawn taskOne failed\n");
if((taskIdTwo = taskSpawn("t2",90,0x100,2000,(FUNCPTR)taskTwo,0,0,0,0,0,0,0,
         0,0,0)) == ERROR)
         printf("taskSpawn taskTwo failed\n");
}


void taskOne(void) /* task that writes to the message queue */
{
char message[] = "Received message from taskOne";

/* send message */
if((msgQSend(mesgQueueId,message,MAX_MESSAGE_LENGTH, WAIT_FOREVER,
         MSG_PRI_NORMAL)) == ERROR)
         printf("msgQSend in taskOne failed\n");
}

void taskTwo(void) /* tasks that reads from the message queue */
{
char msgBuf[MAX_MESSAGE_LENGTH];

/* receive message */
if(msgQReceive(mesgQueueId,msgBuf,MAX_MESSAGE_LENGTH, WAIT_FOREVER)
                   == ERROR)
         printf("msgQReceive in taskTwo failed\n");
else
                   printf("taskTwo %s\n",msgBuf);
msgQDelete(mesgQueueId); /* delete message queue */
}
-------------------------------------------------------------------------------------
```

## Procedures

1. Copy the source code in the example and compile it.
2. Load the object file onto the target machine.
3. Run the examples by executing the main routine("message") of the example on WindSh terminal.

Note: Make sure you have redirected I/O, otherwise you won't see the results of the printf commands.

---

## Follow On Experiment

Experiment 1. Modify the above program so that taskTwo can send a string message ("Received message from taskTwo") to taskOne. Have taskOne print the message sent from taskTwo.

Experiment 2. Modify the above program so that taskOne can send a string message ("Received message from taskOne") to taskTwo and taskTwo can send a string message ("Received message from taskTwo") to taskOne. Have taskOne print the message sent from taskTwo and taskTwo print the message sent from taskOne. The printout from the program looks like the following:

```
taskTwo Received message from taskOne
taskOne Received message from taskTwo
```

---

## Additional Information

Refer to VxWorks User's Manual and Reference Manual.

---