Experiment Flow1
Data Flow Control

Description

7.7.2 Interlocked, One-Way Data Communication

In some designs, a sending task might require a handshake (acknowledgement) that the receiving task has been successful in receiving the message. This process is called interlocked communication, in which the sending task sends a message and waits to see if the message is received.

This requirement can be useful for reliable communications or task synchronization. For example, if the message for some reason is not received correctly, the sending task can resend it. Using interlocked communication can close a synchronization loop. To do so, you can construct a continuous loop in which sending and receiving tasks operate in lockstep with each other. An example of one-way, interlocked data communication is illustrated in Figure 7.7.

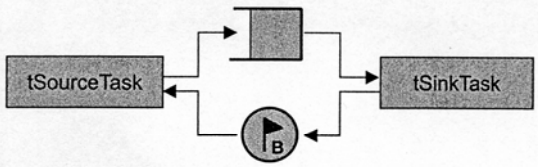

Figure 7.7 Interlocked, one-way data communication.

In this case, `tSourceTask` and `tSinkTask` use a binary semaphore initially set to `0` and a message queue with a length of `1` (also called a mailbox). `tSourceTask` sends the message to the message queue and blocks on the binary semaphore. `tSinkTask` receives the message and increments the binary semaphore. The semaphore that has just been made available wakes up `tSourceTask`. `tSourceTask`, which executes and posts another message into the message queue, blocking again afterward on the binary semaphore.

The pseudo code for interlocked, one-way data communication is provided in Listing 7.2.

The semaphore in this case acts as a simple synchronization object that ensures that `tSourceTask` and `tSinkTask` are in lockstep. This synchronization mechanism also acts as a simple acknowledgement to `tSourceTask` that it's okay to send the next message.

7.7.3 Interlocked, Two-Way Data Communication

Sometimes data must flow bidirectionally between tasks, which is called interlocked, two-way data communication (also called full-duplex or tightly coupled communication). This form of communication can be useful when designing a client/server-based system. A diagram is provided in Figure 7.8.

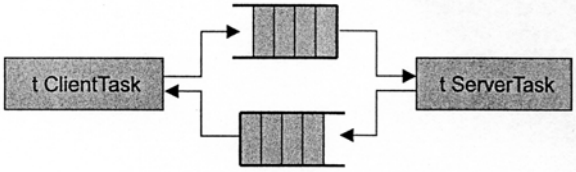

Figure 7.8 Interlocked, two-way data communication.

Listing 7.2 Pseudo code for interlocked, one-way data communication.

```
tSourceTask ()
{
                :
                Send message to message queue
                Acquire binary semaphore
                :
}

tSinkTask ()
{
                :
                Receive message from message queue
                Give binary semaphore
                :
}
```

In this case, `tClientTask` sends a request to `tServerTask` via a message queue. `tServerTask` fulfills that request by sending a message back to `tClientTask`.

The pseudo code is provided in Listing 7.3.

Listing 7.3 Pseudo code for interlocked, two-way data communication.

```
tClientTask ()
{
                :
                Send a message to the requests queue
                Wait for message from the server queue
                :
}

tServerTask ()
{
                :
                Receive a message from the requests queue
                Send a message to the client queue
                :
}
```

Experiment

Experiment 1. Write and test a sample program sending and receiving 10 messages using Interlocked, One-Way Data Communication.

Experiment 2. Write and test a sample program sending and receiving 10 messages

using Interlocked, Two-Way Data Communication.

Additional Information

Refer to VxWorks User's Manual and Reference Manual.
