DSP II: ELEC 4523

QUE Module

Objectives

- Become familiar with QUE objects and their use in sharing data between TSK objects syncronized with SEM objects.

Reading

- SPRU423 TMS320 DSP/BIOS Users Guide: Queues (section)
- Code Composer Studio Online Help: QUE Module

Lab Module Prerequisites

Be sure to do the TSK and SEM modules before the QUE module.

Description

The QUE module provides a way to manage linked lists of data objects. A linked list is a group of objects where an object in the list has information about the previous and next elements in the list as demonstrated in Figure 1. A linked list can be used to implement buffers that operate in a FIFO, LIFO, LILO and FILO manner, to name a few. Many times it is used in a FIFO manner.

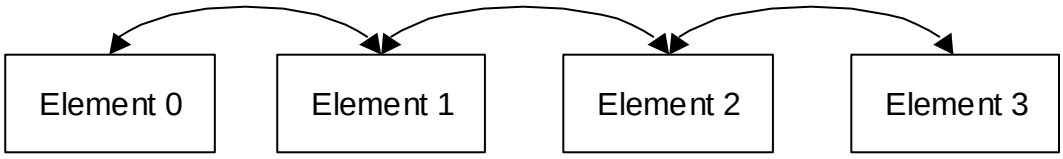

Figure 1: Diagram of a linked list

The basic element in a queue structure is a `QUE_Elem`:

```
typedef struct QUE_Elem {
      struct QUE_Elem *next;
      struct QUE_Elem *prev;
} QUE_Elem;
```

A message or object put on a queue is a structure that has as its first element a `QUE_Elem`. An example would be:

```
typedef struct MsgObj {
    QUE_Elem   elem;      /* first field for QUE */
    Int        val;       /* message value */
} MsgObj, *Msg;
```

To put a message at the end of a queue the function use `QUE_put` and to remove a message from the beginning of the queue use `QUE_get`. These two functions allow the queue to be used as a FIFO buffer.

The following code can be used to allocate memory for a message and put it on a queue.

```
Msg         msg;

msg = MEM_alloc(0, sizeof(MsgObj), 0);
if (msg == MEM_ILLEGAL) {
      SYS_abort("Memory allocation failed!\n");
}
/* fill the message with data here */
QUE_put(&queue, msg);
```

Before trying to get a message from a queue the code should check to see if there is a message available on the queue. The following code makes this check and then gets the message from the queue.

```
if (QUE_empty(&queue)) {
      SYS_abort("queue error\n");
}
/* dequeue message */
msg = QUE_get(&queue);
/* use the message here */
/* free msg */
MEM_free(0, msg, sizeof(MsgObj));
```

Functions `QUE_put` and `QUE_get` are atomic in that they add and remove elements from the queue with interrupts turned off. Therefore there should not be a problem of more than one task trying to access the queue at the same time. The function `QUE_get` is also non-blocking so the tasks should determine if there are any elements on the queue before calling `QUE_get`. A semaphore can be used to count the number of elements on a queue and used to block a task that needs access to a queue.

Laboratory

Part 1

- In this part you will be creating two TSKs were one will generate some data and send it to the other TSK in a queue. The transmitting TSK will generate 5 messages to send to the receiving TSK which will print out the content of the message.
- Create a new project called `quelab`.
- Create a new DSP/BIOS Configuration file and use the `C6xxx.cdb` template for use with the simulator.
- Save the file as `quelab.cdb` and add it in your project. Also add the`quelabcfg.cmd` file.
- If using the simulator then change the RTDX interface to Simulator by right clicking on Input/Output->RTDX and bringing up the properties. Change the RTDX mode to Simulator. If you do not do this then when you load your program you will see the error "RTDX

application does not match emulation protocol." If you are loading onto an EVM or DSK you shouldn't need to change this.

- Create a LOG object by right clicking on Instrumentation->LOG and selecting Insert LOG. Change the name to `trace`. Set its properties to have a length of 512 and be a circular buffer.
- Change the `LOG_system` object to have a length of 512.
- Create a QUE object by right clicking on Syncronization->QUE and selecting Insert QUE. This will create a queue called `QUE0`. There are no properties to set (unless you want to change the comment).
- Create two TSKs with the following properties
  - Name: `TSK0`, priority: 1, function: `_funTSK0`.
  - Name: `TSK1`, priority: 1, function: `_funTSK1`.
- On the priority list make sure that `TSK0` is first.
- Create a main.c file and include a `main` function that does nothing. Include this file in your project.
- At the top of `main.c` make a global structure for your message as follows:

```
typedef struct MsgObj {
    QUE_Elem      elem;      /* first field for QUE */
    Int           val;       /* message value */
} MsgObj, *Msg;
```

- In the `main.c` file make functions for your TSKs, `funTSK0` and `funTSK1`.
- In fun`TSK0`, which will generate the data, do the following
  - Put in a loop that will loop 5 times
  - Inside the loop:
    - Allocate memory for a new message
    - Fill the message value with the message number
    - Print the the `trace` LOG which message is being generated
    - Put the message on the queue `QUE0`
- In fun`TSK1`, which will receive the data, do the following
  - Put in a loop that will loop 5 times
  - Inside the loop:
    - Check to see if there is a message on the queue and if not then print an error message to the `trace` LOG and return. This will cause the task to be done running.
    - Get the message from the queue `QUE0`
    - Print which message number was read
    - Free the buffer for the message
- Build and load your project.
- Use the Execution Graph and the LOG manager to examine the processing. Open the RTA Control Plannel by selecting it under the DSP/BIOS menu. Uncheck the Enable SWI logging. This will cause the KNL_swi to not be logged since we are not interested in when it runs.
- Run the program and record the results.
- Describe in detail the processing that is occurring.

Part 2

- Change the priority of `TSK1` to 2.
- Build and load your project.
- Run the program and record the results.
- How does the execution change? Describe in detail.

Part 3

- In this part you will be adding a semaphore to syncronize the two tasks using the queue.
- Copy the `main.c` file from above to a new file `main3.c`. Remove the `main.c` from your project and add `main3.c`.
- Add a semaphore to your configuration file and call it `SEM0`. It should be initialized to zero.
- In main3.c change the following:
  - In `funTSK0` after the `QUE_put` add a `SEM_post` for `SEM0`.
  - In `funTSK1` delete the code that checks to see if there is a message on the queue and replace it with as `SEM_pend` for `SEM0`. Use `SYS_FOREVER` for the timeout on the pend.
- Build and load your project.
- Run the program and record the results.
- How does the execution change? Describe in detail.

Part 4

- Notice that in Part 3 that if the program ran fro a long time that the code would have to continually allocate memory for a message and then deallocate it when it was done using the message. This could take up a substantial amount of time and could cause fragmentation of the memory space. A better method is to have two queues where one queue holds messages that are free and one holds messages that contain data being transmitted from one task to another. In this part you will be adding another queue which will be initialized by adding some free messages to it. Also, another semaphore will be added to track the number of free messages on the free queue.
- Copy the `main3.c` file from above to a new file `main4.c`. Remove the `main3.c` from your project and add `main4.c`.
- Add a semaphore to your configuration file and call it `freeSEM`. It should be initialized to zero.
- Create a QUE object and call it `freeQUE`.
- Change the priority of `TSK1` to 1 and make sure `TSK0` is first in the list.
- To the main function add code that loops 3 times and adds 3 messages to the `freeQUE`. Be sure to post to `freeSEM` after adding the message to the queue.
- In the function `funTSK0` delete the code that allocates a message and replace it will code that pends on `freeSEM` and then gets a message from `freeQUE`.
- In the function `funTSK1` delete the code that frees the message buffer and replace it will code that puts the message on `freeQUE` and then posts to `freeSEM`.
- Build and load your project.
- Run the program and record the results.

- How does the execution change? Describe in detail.