CG2271 Real Time Operating Systems
Tutorial 9

Question 1

Assume that messages in your RTOS consist of void pointers, that sendmsg places the void pointer passed to it on a queue, and that rcvmesg returns the void pointer it retrieved from the queue. What is wrong with the following code?

```
void vLookForInputTask(void)
{
      ….
      if(!! A key has been pressed on the keyboard)
            vGetKey();
}

void vGetKey(void)
{
      char ch;
      ch = !! Get key from the keyboard;
      /* Send key to keyboard command handler task */
      sndmsg(KEY_MBOX, &ch, PRIORITY_NORMAL);
}

void vHandleKeyCommandsTask(void)
{
      char *p_chLine;
      char ch;
      while(1)
      {
            /* Wait for key to be received */
            p_chLine = rcvmsg(KEY_MBOX, WAIT_FOREVER);
            ch = *p_chLine;
            !! Do stuff with ch
      } /* while */
}
```

Question 2

This code uses the following *AMX* functions to create and use events:

```
ajevcre(AMXID *p_amxidGroup, unsigned int uValueInit,
               char *p_chTag);
```

- Creates a group of 16 events, and returns a handle to the events in *p_amxidGroup*. The events group is also given a four-character tag given in *p_chTag*. The events are initialized to "set" or "reset" depending on the value of *uValueInit*.

```
ajevsig(AMXID amxidGroup, unsigned int uMask,
         unsigned int uValueNew);
```

- Sets and resets the events in the group indicated by *amxidGroup*. The *uMask* parameter indicates which of the 16 events in the group should be set or reset, depending on the parameter *uValueNew*.

```
ajevwat(AMXID amxidGroup, unsigned int uMask, unsigned int uValue,
               int iMatch, long lTimeOut);
```

- Causes the task to wait for one or more events in the group indicated by *amxidGroup*. The *umask* parameter indicates which of the 16 events in the group the task should wait for. *uValue* indicates whether the task wants to wait for the selected events to be "set" or "reset", while *iMatch* specifies whether the task should unblock when ALL or AT LEAST ONE of the indicated events is set or reset. *lTimeOut* indicates how long the task is willing to wait.

Given the information above, rewrite the following code with semaphores.

```
/* Handle for trigger group of events */
AMXID amxidTrigger;

/* Constants for use in the group */
#define TRIGGER_MASK       0x0001
#define TRIGGER_SET              0x0001
#define TRIGGER_RESET            0x0000

void main(void)
{
   ……
   /* Create an event group with the trigger and keyboard events set */
   ajevcre(&amxidTrigger, 0, “EVTR”);
   ……
}

void interruptvTriggerISR(void)
{
         /* User pulled trigger. Set the event */
         ajevsig(amxidTrigger, TRIGGER_MASK, TRIGGER_SET);
}

void vScanTask(void)
{
         ……
   while(1)
   {
         /* Wait for user to pull trigger */
         ajevwat(amxidTrigger, TRIGGER_MASK, TRIGGER_SET,
               WAIT_FOR_ANY, WAIT_FOREVER);

         /* Reset the trigger event */
         ajevsig(amxidTrigger, TRIGGER_MASK, TRIGGER_RESET);
         !! turn on hardware scanner
   } /* while */
}
```

Question 3

What does the term “re-entrancy” mean? In particular, what does it mean when we say that a routine is “re-entrant”? Demonstrate, with an example, the problems associated with a non re-entrant routine, and how the routine can be made re-entrant. GIYF. ☺

Question 4

Is this function re-entrant?

```
int strlen(char *p_sz)
{
   int iLength;

   iLength = 0;

   while(*p_sz != '\0')
   {
        ++iLength;
        ++p_sz;
   }

   return iLength;
}
```

Question 5

Consider the statement: "In a non-preemptive RTOS, tasks cannot interrupt each other. Therefore there are no data sharing problems amongst task." Would you agree with this?