Real-Time Operating Systems (Part II)

Embedded Software Design

熊博安

國立中正大學資訊工程研究所

pahsiung@cs.ccu.edu.tw

Textbooks: Embedded Software Primer, David E. Simon, Addison Wesley

Programming Embedded Systems with C and GNU Development Tools, Michael Barr & Anthony Massa

Contents

- Intertask Communication
- Timer Services
- Memory Management
- Events
- RTOS and ISR

Intertask Communication

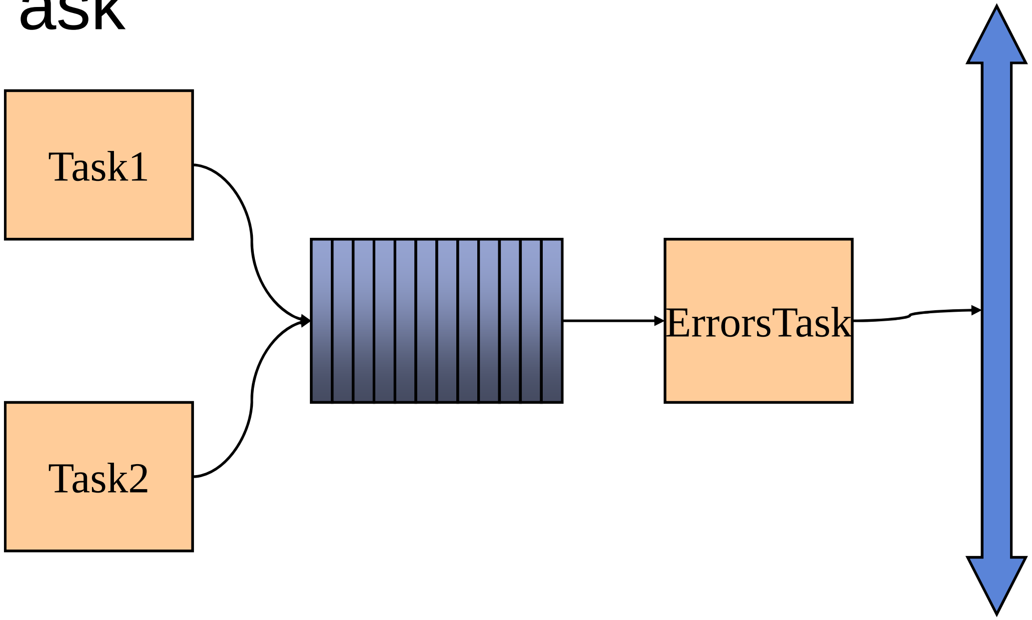
- Shared data
  - atomically-accessed variables
  - reentrant functions
- Semaphores
  - binary, counting semaphores
  - monitors
- Message Queues
- Mailboxes
- Pipes

Message Queues

- Simple Example:
  - 2 tasks discover error conditions that must be reported on the network (time consuming!)
  - 1 more task handles the error reporting
    - Task1 and Task2 report errors to ErrorsTask
- Qs: How to implement this in an RTOS?
  - Ans: Use an RTOS queue!

Message Queue Example

- Three tasks
  - Task1
  - Task2
  - ErrorsTask



Message Queue Example: Code

- Task1 and Task2

```
/* RTOS queue function prototypes */
void AddToQueue (int iData);
void ReadFromQueue (int *p_iData);
```

```
void Task1 (void)
{
    .
    .
    .
    if (!!problem arises)
        vLogError (ERROR_TYPE_X);

    !! Other things that need to be do
    .
    .
    .
}
```

```
void Task2 (void)
{
    .
    .
    .
    if (!!problem arises)
        vLogError (ERROR_TYPE_Y);

    !! Other things that need to be done soon.
    .
    .
    .
}
```

Message Queue Example: Code

- ErrorsTask

```
void vLogError (int iErrorType)
{
   AddToQueue (iErrorType);
}

static int cErrors;

void ErrorsTask (void)
{
   int iErrorType;

   while (FOREVER)
   {
      ReadFromQueue (&iErrorType);
      ++cErrors;
      !! Send cErrors and iErrorType out on network
   }
}
```

If the queue is empty, this function will block the calling task.

Message Queue Example

- AddtoQueue
  - add an integer to a queue in RTOS
- ReadFromQueue
  - read value from head of queue
- Both functions must be reentrant!

Message Queue Details

- Queues must be initialized before using
  - call an initialization function
  - must initialize before any task tries to use them
  - may have to allocate memory for queue
- RTOSs allow multiple queues
  - must identify queue in function calls
- Condition handler for full queue:
  - write operation failed error, OR
  - block until space available after some read

Message Queue Details

- Condition handler for empty queue:
  - queue empty error return, AND
  - block until data available in queue
- Write block size allowed by RTOS < Write block size desired by task
  - Qs: How to handle this situation?
  - Ans:
    - Write actual data to a buffer
    - Write buffer pointer, a (void *)-sized block, into queue

Message Queue in μCOS

- Queue data structures and operations

```
/* RTOS queue function prototypes */
OS_EVENT *OSQCreate (void **ppStart, BYTE bySize);
unsigned char OSQPost (OS_EVENT *pOse, void *pvMsg);
void *OSQPend (OS_EVENT *pOse, WORD wTimeout, BYTE *pByErr);
#define WAIT_FOREVER 0

/* Our message queue */
static OS_EVENT *pOseQueue;

/* The data space for our queue.  The RTOS will manage this. */
#define SIZEOF_QUEUE 25
void *apvQueue[SIZEOF_QUEUE];
```

Message Queue in µCOS

- main, Task1 and Task2

```
void main (void)
{
    .
    .
    .
    /* The queue gets initialized before the tasks are started */
    pOseQueue = OSQCreate (apvQueue, SIZEOF_QUEUE);
    .
    .
    .
    !! Start Task1
    !! Start Task2
    .
    .
    .
}
```

```
void Task1 (void)
{
    .
    .
    .
    if (!!problem arises)
        vLogError (ERROR_TYPE_X);

    !! Other things that need to be done soon.
    .
    .
    .
```

```
void Task2 (void)
{
    .
    .
    .
    if (!!problem arises)
        vLogError (ERROR_TYPE_Y);

    !! Other things that need to be done soon.
    .
    .
    .
}
```

Message Queue in µCOS

- vLogError

```
void vLogError (int iErrorType)
{
   BYTE byReturn;      /* Return code from writing to queue */

   /* Write to the queue.  Cast the error type as a void pointer
      to keep the compiler happy. */
   byReturn = OSQPost (pOseQueue, (void *) iErrorType);

   if (byReturn != OS_NO_ERR)
      !! Handle the situation that arises when the queue is full
}
```

Message Queue in μCOS

- ErrorsTasks

```
static int cErrors;

void ErrorsTask (void)
{
   int iErrorType;
   BYTE byErr;

   while (FOREVER)
   {
      /* Cast the value received from the queue back to an int.
         (Note that there is no possible error from this, so
         we ignore byErr.) */
      iErrorType =
         (int) OSQPend (pOseQueue, WAIT_FOREVER, &byErr);

      ++cErrors;

      !! Send cErrors and iErrorType out on network
   }
}
```

Message Queues & Pointers

```
/* Queue function prototypes */
OS_EVENT *OSQCreate (void **ppStart, BYTE bySize);
unsigned char OSQPost (OS_EVENT *pOse, void *pvMsg);
void *OSQPend (OS_EVENT *pOse, WORD wTimeout, BYTE *pByErr);
#define WAIT_FOREVER 0

static OS_EVENT *pOseQueueTemp;

void vReadTemperaturesTask (void)
{
   int *pTemperatures;

   while (TRUE)
   {
      !! Wait until it's time to read the next temperature

      /* Get a new buffer for the new set of temperatures. */
      pTemperatures = (int *) malloc (2 * sizeof *pTemperatures);

      pTemperatures[0] = !! read in value from hardware;
      pTemperatures[1] = !! read in value from hardware;

      /* Add a pointer to the new temperatures to the queue */
      OSQPost (pOseQueueTemp, (void *) pTemperatures);
   }
}
```

Message Queues & Pointers

```
void vMainTask (void)
{
   int *pTemperatures;
   BYTE byErr;

   while (TRUE)
   {
      pTemperatures =
         (int *) OSQPend (pOseQueueTemp, WAIT_FOREVER, &byErr);
      if (pTemperatures[0] != pTemperatures[1])
         !! Set off howling alarm;

      free (pTemperatures);
   }
}
```

Message Passing in Linux (main program)

```
#include <pthread.h>
#include <mqueue.h>
#include "led.h"
int8_t messageQueuePath[] = "message queue";

int main(void) {
    mqd_t messageQueueDescr;

    /* Configure the green LED control pin. */
    ledInit( );
    /* Create the message queue for sending information between tasks. */
    messageQueueDescr = mq_open(messageQueuePath, (O_CREAT | O_EXCL | O_RDWR));
    /* Create the producer task using the default task attributes. */
    pthread_create(&producerTaskObj, NULL, (void *)producerTask, NULL);
    /* Create the consumer task using the default task attributes. */
    pthread_create(&consumerTaskObj, NULL, (void *)consumerTask, NULL);
    /* Allow the tasks to run. */
    pthread_join(producerTaskObj, NULL);
    pthread_join(consumerTaskObj, NULL);
    return 0;
}
```

Message Passing in Linux (producerTask)

```
#include <unistd.h>
#include "button.h"
typedef union { uint32_t count; uint8_t buf[4]; } msgbuf_t;
void producerTask(void *param) {
    uint32_t buttonPressCount = 0;
    mqd_t messageQueueDescr;
    uint8_t button;
    msgbuf_t msg;
    messageQueueDescr = mq_open(messageQueuePath, O_WRONLY);
    while (1) {
        /* Delay for 10 milliseconds. */
        usleep(10000);
        /* Check whether SW0 button has been pressed. */
        button = buttonDebounce( );
        /* If button SW0 was pressed, send a message to consumer. */
        if (button & BUTTON_SW0) {
            buttonPressCount++; msg.count = buttonPressCount;
            mq_send(messageQueueDescr, &msg.buf[0],
    sizeof(buttonPressCount), 0);
} } }
```

Message Passing in Linux (consumerTask)

```
void consumerTask(void *param) {
   mqd_t messageQueueDescr;
   msgbuf_t rcvMsg;

   /* Open the existing message queue */
   messageQueueDescr = mq_open(messageQueuePath, O_RDONLY);

   while (1) {
        /* Wait for a new message. */
        mq_receive(messageQueueDescr, &rcvMsg.buf[0], 4, NULL);

        printf("Button SW0 pressed %d times.\\n", rcvMsg.count);
        ledToggle( );
   }
}
```

Mailboxes

- Like queues
- Mailbox functions:
  - Create a mailbox
  - Write to a mailbox
  - Read from a mailbox
  - Check if a mailbox has any message
  - Destroy an unused mailbox

Mailboxes

- Variations
  - Mailbox size
    - One message: write only once, must read before next write
    - User-defined: size parameter in create mailbox function
    - Unlimited: each mailbox is not limited, but total size of all mailboxes is limited
  - Priority
    - higher-priority messages read before low-priority ones, regardless of write order

Mailbox Example in MultiTask!

- Each message is a void pointer
- Must create all mailboxes you need when you configure the system
- Three functions:
- int **sndmsg** (unsigned int *uMbId*, void **p_vMsg*, unsigned int *uPriority*);
- void ***rcvmsg** (unsigned int *uMbId*, unsigned int *uTimeout*);
- void ***chkmsg** (unsigned int *uMbId*);

Mailbox ID

Message

Msg Priority

Block Time

Return 1st Msg, or NULL

Mailbox in eCos (main program)

```
#include <cyg/kernel/kapi.h>
#include <cyg/infra/diag.h>
#include "led.h"
cyg_handle_t mailboxHdl;
cyg_mbox mailbox;
void cyg_user_start(void) {
    /* Configure the green LED control pin. */
    ledInit( );
    /* Create the mailbox for sending messages between tasks. */
    cyg_mbox_create(&mailboxHdl, &mailbox);
    /* Create the producer and consumer tasks. */
    cyg_thread_create(PRODUCER_TASK_PRIORITY, producerTask,
          (cyg_addrword_t)0, "Producer Task", (void *)producerTaskStack,
          PRODUCER_TASK_STACK_SIZE, &producerTaskHdl, &producerTaskObj);
    cyg_thread_create(CONSUMER_TASK_PRIORITY, consumerTask,
          (cyg_addrword_t)0, "Consumer Task", (void *)consumerTaskStack,
          CONSUMER_TASK_STACK_SIZE, &consumerTaskHdl, &consumerTaskObj);
    /* Notify the scheduler to start running the tasks. */
    cyg_thread_resume(producerTaskHdl);
    cyg_thread_resume(consumerTaskHdl);
    diag_printf("eCos mailbox example - press button SW0.\\n");
}
```

Mailbox in eCos (producerTask)

```
#include "button.h“
void producerTask(cyg_addrword_t data) {
   uint32_t buttonPressCount = 0;
   int buttonOn;
   while (1) {
        /* Delay for 10 milliseconds. */
        cyg_thread_delay(TICKS_PER_SECOND / 100);

        /* Check if the SW0 button has been pressed. */
        buttonOn = buttonDebounce( );

        /* If button SW0 was pressed, send a message to consumer. */
        if (buttonOn) {
                buttonPressCount++;
                cyg_mbox_put(mailboxHdl, (void *)buttonPressCount);
        }
   }
}
```

Mailbox in eCos (consumerTask)

```
void consumerTask(cyg_addrword_t data) {
   uint32_t rcvMsg;

   while (1) {
       /* Wait for a new message. */
       rcvMsg = (uint32_t)cyg_mbox_get(&mailboxHdl);

       diag_printf("Button SW0 pressed %d times.\\n",
              rcvMsg);

       ledToggle( );
   }
}
```

Pipes

- Like queues
- Functions:
  - Create
  - Write to
  - Read from
  - …

Pipes

- Variations:
  - Varying lengths of write block size
  - Byte-oriented
    - Task 1 writes 11 bytes to pipe
    - Task 2 writes 19 bytes to pipe
    - Task 3 reads 14 bytes from pipe
      - 11 bytes from task 1
      - 3 bytes from task 2
    - 16 bytes remaining in pipe
  - Some RTOSs use C functions: fread, fwrite

Which Should I Use?

- Depends on RTOS
- Each RTOS has different implementations for message queues, mailboxes, and pipes.
- Trade-off among flexibility, speed, memory space, length of interrupt disabled time, …
- Read RTOS documents and decide which best meets your requirements

Pitfalls

- No restrictions on reader/writer of queue, mailbox, or pipe
  - temperature data written to a queue read by task expecting error codes → system failure
- Data type mismatch between write and read
  - wrong interpretation of data
  - e.g.: write int, read pointer
  - compilers can find obvious errors, BUT
  - compilers cannot find interpretation errors

Pitfalls (Data type mismatch: caught by compiler)

```
/* Declare a function that takes a pointer parameter */
void vFunc (char *p_ch);

void main (void)
{
    int i;
    .
    .
    .
    /* Call it with an int, and get a compiler error */
    vFunc (i);
    .
    .
    .
}
```

Pitfalls (Data type mismatch: uncaught by compiler)

```
static OS_EVENT *pOseQueue;

void TaskA (void)
{
    int i;
    .
    .
    .
    /* Put an integer on the queue. */
    OSQPost (pOseQueue, (void *) i);
    .
    .
    .
}

void TaskB (void)
{
    char *p_ch;
    BYTE byErr;
    .
    .
    .
    /* Expect to get a character pointer. */
    p_ch = (char *) OSQPend (pOseQueue, FOREVER, byErr);
    .
    .
    .
}
```

Pitfalls

- Running out of space in queues, mailboxes, or pipes is a disaster
  - Communication is not optional
  - Good solution: make it large enough in the first place
- Passing pointers through queues, mailboxes, or pipes ➔ shared data!

Pointer passing = shared data bug

- No malloc and free

```
/* Queue function prototypes */
OS_EVENT *OSQCreate (void **ppStart, BYTE bySize);
unsigned char OSQPost (OS_EVENT *pOse, void *pvMsg);
void *OSQPend (OS_EVENT *pOse, WORD wTimeout, BYTE *pByErr);
#define WAIT_FOREVER 0

static OS_EVENT *pOseQueueTemp;

void vReadTemperaturesTask (void)
{
   int *pTemperatures;

   while (TRUE)
   {
      !! Wait until it's time to read the next temperature

      /* Get a new buffer for the new set of temperatures. */
      pTemperatures = (int *) malloc (2 * sizeof *pTemperatures);

      pTemperatures[0] = !! read in value from hardware;
      pTemperatures[1] = !! read in value from hardware;

      /* Add a pointer to the new temperatures to the queue */
      OSQPost (pOseQueueTemp, (void *) pTemperatures);
   }
}
```

Before

```
/* Queue function prototypes */
OS_EVENT *OSQCreate (void **ppStart, BYTE bySize);
unsigned char OSQPost (OS_EVENT *pOse, void *pvMsg);
void *OSQPend (OS_EVENT *pOse, WORD wTimeout,
      BYTE *pByErr);
#define WAIT_FOREVER 0
static OS_EVENT *pOseQueueTemp;

void vReadTemperaturesTask (void)
{
   int iTemperatures[2];

   while (TRUE)
   {
      !! Wait until it's time to read the next temperature

      iTemperatures[0] = !! read in value from hardware;
      iTemperatures[1] = !! read in value from hardware;

      /* Add to the queue a pointer to the temperatures
         we just read */
      OSQPost (pOseQueueTemp, (void *) iTemperatures);
   }
}
```

After

An Example

- No malloc and free

Before

```
void vMainTask (void)
{
   int *pTemperatures;
   BYTE byErr;

   while (TRUE)
   {
      pTemperatures =
         (int *) OSQPend (pOseQueueTemp, WAIT_FOREVER, &byErr);
      if (pTemperatures[0] != pTemperatures[1])
         !! Set off howling alarm;

      free (pTemperatures);
   }
}
```

After

```
void vMainTask (void)
{
   int *pTemperatures;
   BYTE byErr;

   while (TRUE)
   {
      pTemperatures = (int *)
         OSQPend (pOseQueueTemp, WAIT_FOREVER, &byErr);
      if (pTemperatures[0] != pTemperatures[1])
         !! Set off howling alarm;
   }
}
```

Timer Functions

- Most embedded systems must keep track of the passage of time
  - cordless bar-code scanner turns itself off after a certain number of seconds
  - wait for ack, re-transmit data on network
  - wait for robot arms to move
  - wait for motors to come up to speed
- RTOS provides a delay function

VxWorks RTOS Delay Function

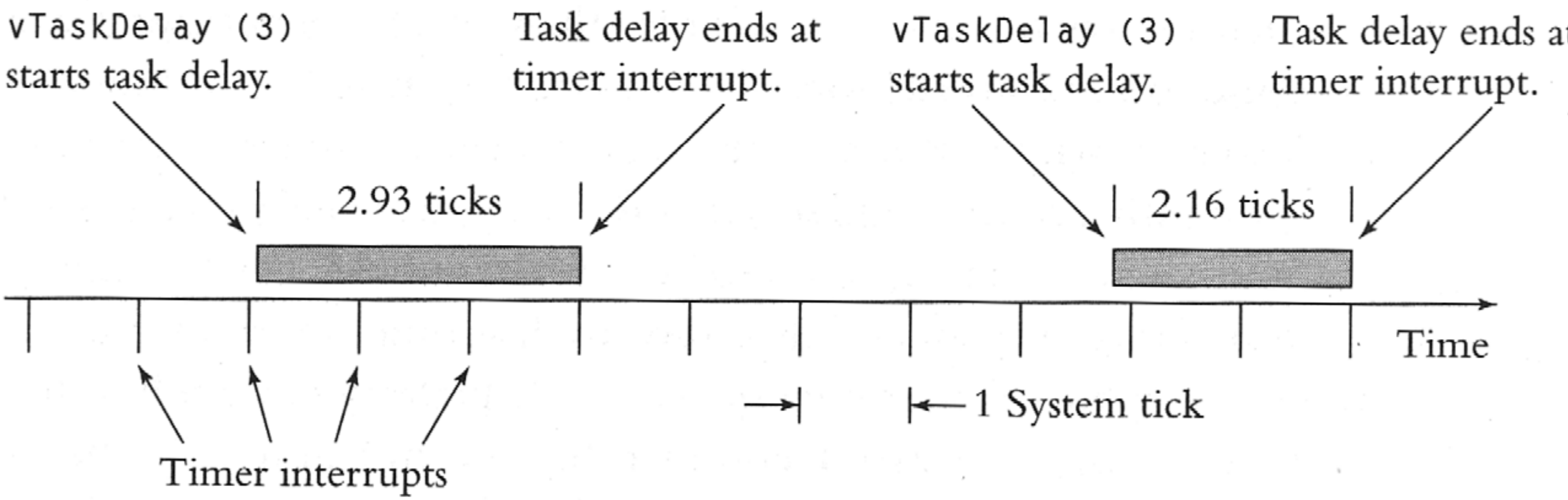
```
/* Message queue for phone numbers to dial. */
extern MSG_Q_ID queuePhoneCall;

void vMakePhoneCallTask (void)
{
   #define MAX_PHONE_NUMBER  11
   char a_chPhoneNumber[MAX_PHONE_NUMBER];
         /* Buffer for null-terminated ASCII number */
   char *p_chPhoneNumber;
         /* Pointer into a_chPhoneNumber */
   .
   .
   .
   while (TRUE)
   {
      msgQreceive (queuePhoneCall, a_chPhoneNumber,
         MAX_PHONE_NUMBER, WAIT_FOREVER);

      /* Dial each of the digits */
      p_chPhoneNumber = a_chPhoneNumber;
      while (*p_chPhoneNumber)
      {
         taskDelay (100);    /* 1/10th of a second silence */
         vDialingToneOn (*p_chPhoneNumber -'0');
         taskDelay (100);    /* 1/10th of a second with tone */
         vDialingToneOff ();

         /* Go to the next digit in the phone number */
         ++p_chPhoneNumber;
      }
      .
      .
      .
   }
}
```

Questions

- What is the unit of time for the taskDelay function? milliseconds?
  - No! It is the number of system ticks!
  - Length of one system tick can be controlled when you set up the system
- How accurate are the delays?
  - Accurate to the nearest system tick
  - Heartbeat timer: a hardware timer
  - taskDelay(n) ➔ n-1 < TimerExpires < n

Timer Function Accuracy

- vTaskDelay(3)



Questions

- How does RTOS know how to setup hardware timer?
  - RTOS is microprocessor dependent and so is the hardware timer, thus RTOS engineers know a priori how to setup the microprocessor hardware timer
  - Many RTOS vendors provide “board support packages” (BSPs), which contain driver software for common hardware components, such as timers, etc.

Questions

- What is the “normal length” for the system tick?
  - None!
  - Short ➔
    - accurate timings
    - frequent execution of timer interrupt routines ➔ decreased system performance
- What if I need extremely accurate timing?
  - Make system tick short enough
  - Use a separate hardware timer for those timings

Other Timing Services

- To limit waiting time of a task for a message from a queue or a mailbox or for a semaphore
- **Issue**:
  - High-priority task attempts to get a semaphore
  - Time limit expires, task does not have the semaphore, task cannot access shared data
  - Need to write recovery code
- **Solution**:
  - send instructions about using shared data through a mailbox to a low-priority task

Other Timing Services

- To call a function after a given number of system ticks
- An Example:
- Handle radio hardware: turn on & off from time to time
- Turn radio off: cut the power
- Turn radio on:
  - turn on power to basic radio hardware
  - after waiting 12 ms, set radio frequency
  - after waiting 3 ms, turn on transmitter / receiver
  - radio is ready to function

Timer Callback Functions

Figure 7.7 Using Timer Callback Functions

```
/* Message queue for radio task. */
extern MSG_Q_ID queueRadio;

/* Timer for turning the radio on. */
static WDOG_ID wdRadio;

static int iFrequency;      /* Frequency to use. */

void vSetFrequency (int i);
void vTurnOnTxorRx (int i);

void vRadioControlTask (void)
{
   #define MAX_MSG 20
   char a_chMsg[MAX_MSG + 1];  /* Message sent to this task */

   enum
   {
      RADIO_OFF,
      RADIO_STARTING,
```

(continued)

Timer Callback Functions

Figure 7.7 *(continued)*

```
      RADIO_TX_ON,
      RADIO_RX_ON,
   } eRadioState;   /* State of the radio */
   eRadioState = RADIO_OFF;

   /* Create the radio timer */
   wdRadio = wdCreate ();

   while (TRUE)
   {
      /* Find out what to do next */
      msgQReceive (queueRadio, a_chMsg, MAX_MSG, WAIT_FOREVER);

      /* The first character of the message tells this task what
         the message is. */
      switch (a_chMsg[0])
      {
         case 'T':
         case 'R':
            /* Someone wants to turn on the transmitter */
            if (eRadioState == RADIO_OFF)
            {
               !! Turn on power to the radio hardware.

               eRadioState = RADIO_STARTING;
```

(continued)

Timer Callback Functions

Figure 7.7 *(continued)*

```
            /* Get the frequency from the message */
            iFrequency = * (int *) a_chMsg[1];

            !! Store what needs doing when the radio is on.
            /* Make the next step 12 milliseconds from now. */
            wdStart (wdRadio, 12, vSetFrequency,
               (int) a_chMsg[0]);
         }
         else
            !! Handle error.  Can't turn radio on if not off
         break;

      case 'K':
         /* The radio is ready. */
         eRadioState = RADIO_TX_ON;
         !! Do whatever we want to do with the radio
         break;

      case 'L':
         /* The radio is ready. */
         eRadioState = RADIO_RX_ON;
         !! Do whatever we want to do with the radio
         break;
```

Timer Callback Functions

```
        case 'X':
           /* Someone wants to turn off the radio. */
           if (eRadioState == RADIO_TX_ON ||
                 eRadioState == RADIO_RX_ON)
           {
              !! Turn off power to the radio hardware.
              eRadioState = RADIO_OFF;
           }
           else
              !! Handle error.  Can't turn radio off if not on
           break;
        .
        .
        .
        default:
           !! Deal with the error of a bad message
           break;
     }
  }
}
```

(continued)

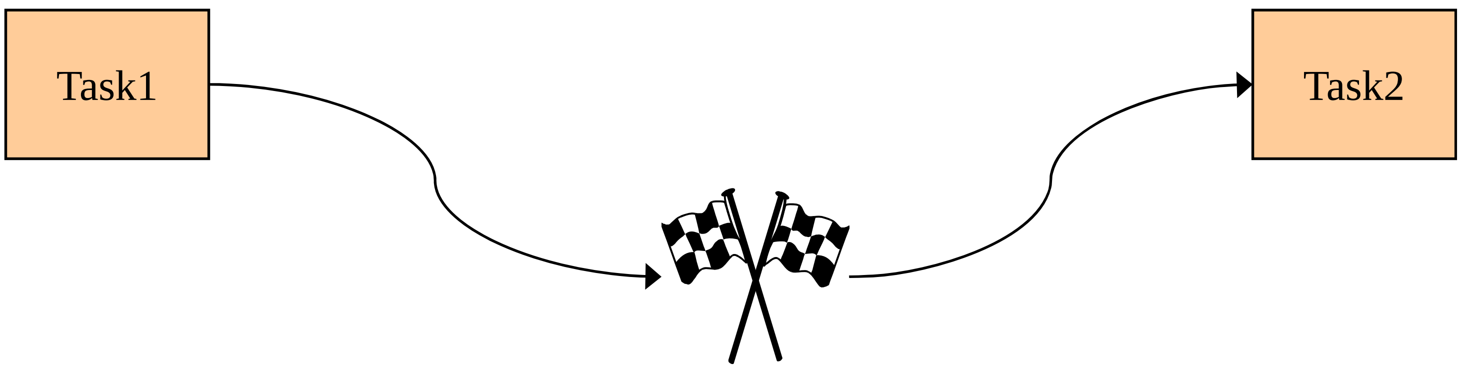
Timer Callback Functions

Figure 7.7 *(continued)*

```
void vSetFrequency (int i)
{
   !! Set radio frequency to iFrequency;

   /* Turn on the transmitter 3 milliseconds from now. */
   wdStart (wdRadio, 3, vTurnOnTxorRx, i);
}

void vTurnOnTxorRx (int i)
{
   if (i == (int) 'T')
   {
      !! Turn on the transmitter

      /* Tell the task that the radio is ready to go. */
      msgQSend (queueRadio, "K", 1,
         WAIT_FOREVER, MSG_PRI_NORMAL);
   }
   else
   {
      !! Turn on the receiver

      /* Tell the task that the radio is ready to go. */
      msgQSend (queueRadio, "L", 1,
         WAIT_FOREVER, MSG_PRI_NORMAL);
   }
}
```

Events

- Event = a Boolean flag that tasks can
  - set, reset, and wait for



- Example: cordless bar-code scanner
  - user pulls trigger ➔
  - laser scanning mechanism must start

Features of Events

- More than one task can block wait for an event, when event occurs
  - all blocked tasks are unblocked, and
  - RTOS executes them in priority order
- RTOS forms groups of events:
  - tasks can wait for any subset of an event group
  - Example: {key-press on scanner keypad, trigger-pull} ➔ start scanning
- Resetting of events:
  - automatically done by RTOS
  - done by user task software

Using Event in AMX

Figure 7.8 Using Events

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

/* Constants for use in the group. */

#define TRIGGER_MASK  0x0001
#define TRIGGER_SET   0x0001
#define TRIGGER_RESET 0x0000
#define KEY_MASK      0x0002
#define KEY_SET       0x0002
#define KEY_RESET     0x0000

void main (void)
{
   .
   .
   .
   /* Create an event group with
      the trigger and keyboard events reset */
   ajevcre (&amxidTrigger, 0, ''EVTR'');
   .
   .
   .
}
```

Using Events in AMX

```
void interrupt vTriggerISR (void)
{
   /* The user pulled the trigger.  Set the event. */
   ajevsig (amxidTrigger, TRIGGER_MASK, TRIGGER_SET);
}

void interrupt vKeyISR (void)
{
   /* The user pressed a key.  Set the event. */
   ajevsig (amxidTrigger, KEY_MASK, KEY_SET);

   !! Figure out which key the user pressed and store that value
}
```

(continued)

Using Events in AMX

Figure 7.8 *(continued)*

```
void vScanTask (void)
{
    .
    .
    .
    while (TRUE)
    {
        /* Wait for the user to pull the trigger. */
        ajevwat (amxidTrigger, TRIGGER_MASK, TRIGGER_SET,
            WAIT_FOR_ANY, WAIT_FOREVER);

        /* Reset the trigger event. */
        ajevsig (amxidTrigger, TRIGGER_MASK, TRIGGER_RESET);

        !! Turn on the scanner hardware and look for a scan.
        .
        .
        .
        !! When the scan has been found, turn off the scanner.
    }
}
```

Using Events in AMX

```
void vRadioTask (void)
{
   .
   .
   .
   while (TRUE)
   {
      /* Wait for the user to pull the trigger or press a key. */
      ajevwat (amxidTrigger, TRIGGER_MASK | KEY_MASK,
         TRIGGER_SET | KEY_SET, WAIT_FOR_ANY,
         WAIT_FOREVER);

      /* Reset the key event.  (The trigger event will be reset
         by the ScanTask.) */
      ajevsig (amxidTrigger, KEY_MASK, KEY_RESET);

      !! Turn on the radio.
      .
      .
      .
      !! When data has been sent, turn off the radio.
   }
}
```

AMX Event Functions

Figure 7.9 *AMX* Event Functions

The *AMX* functions used in Figure 7.8 are the following:

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

The `ajevcre` function creates a group of 16 events, the handle for which is written into the location pointed to by `p_amxidGroup`. The initial values of those events—set and reset—are contained in the `uValueInit` parameter. *AMX* assigns the group a four-character name pointed to by `p_chTag`; this is a special feature of *AMX*, which allows a task to find system objects by name if it does not have access to the handle.

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

The `ajevsig` function sets and resets the events in the group indicated by `amxidGroup`. The `uMask` parameter indicates which events should be set or reset, and the `uValueNew` parameter indicates the new values that the events should have.

AMX Event Functions

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

The `ajevwat` function causes the task to wait for one or more events within the group indicated by `amxidGroup`. The `uMask` parameter indicates which events the task wants to wait for, and `uValue` indicates whether the task wishes to wait for those events to be set or reset. The `iMatch` parameter indicates whether the task wishes to unblock when *all* of the events specified by `uMask` have reached the values specified by `uValue` or when *any one* of the events has reached the specified value. The `lTimeout` parameter indicates how long the task is willing to wait for the events.

AMX also includes functions to delete a group of events that are no longer needed, to read the current values of all the events in a group and to read the values of all the events in a group as of the moment that a task unblocked because an event occurred for which it was waiting.

Comparison: Semaphores, Events

- Semaphores are usually the fastest and simplest methods.
  - However, not much information can pass through a semaphore.
- Events are a little more complicated than semaphores and take up just a hair more microprocessor time than semaphores.
  - ADV: A task can wait for any one of several events at the same time, whereas it can only wait for one semaphore.
  - ADV: Some RTOSs make it convenient to use events and make it inconvenient to use semaphores.

Comparison: Queues

- Queues allow you to send a lot of information from one task to another.
  - The drawbacks
    - putting messages into and taking messages out of queues is more microprocessor-intensive
    - that queues offer you many more opportunities to insert bugs into your code.

Memory Management

- RTOS have memory management
- But, not malloc() and free():
  - slow
  - unpredictable execution times
- Solution:
  - allocate and free **fixed-size buffers**
  - fast and predictable functions
- Example:
  - MultiTask! RTOS

MultiTask! Memory Management

- Pool = a collection of memory buffers of the same size
- Users can declare and use different pools
- void *reqbuf(unsigned int uPoolId);
  - request a buffer from pool uPoolId,
  - return NULL if no buffer available
- void *getbuf(unsigned int uPoolId, unsigned int uTimeout);
  - get a buffer from pool uPoolId
  - block wait for uTimeout

MultiTask! Memory Management

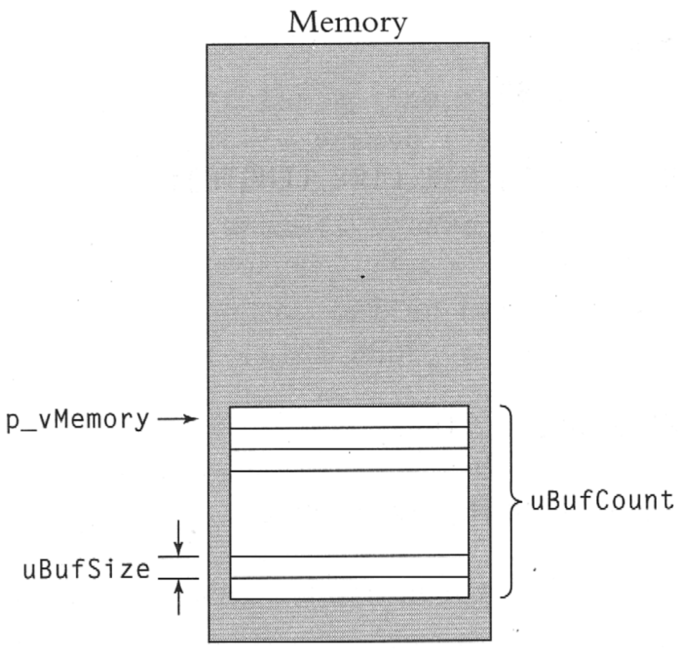
- void relbuf (unsigned int uPoolId,
  void *p_vBuffer);
  - release buffer *p_vBuffer into pool uPoolId
  - RTOS does not check if buffer belongs to pool
    - for efficiency,
    - drastic consequence on error
- RTOS does not know where is free memory
- Application needs to tell RTOS!
- Initialize memory pool init_mem_pool()

MultiTask! Memory Management

```
int init_mem_pool (
    unsigned int uPoolId,
    void *p_vMemory,
    unsigned int uBufSize,
    unsigned int uBufCount,
    unsigned int uPoolType
);
```

used by functions

used by task or ISR



Example of Memory Management in MultiTask!

- Underground tank monitoring system
- Slow thermal printer prints a few lines per second
- 2 tasks
  - high priority task: formats report
  - low priority task: feeds lines to printer
    - one line at a time
    - 40-character line ➔ buffer-size = 40 bytes ➔ waste of memory ➔ solution: use diff pools

Example of Memory Management in MultiTask!

Figure 7.11 Using Memory Management Functions

```
#define LINE_POOL           1
#define MAX_LINE_LENGTH     40
#define MAX_LINES           80

static char a_lines[MAX_LINES][MAX_LINE_LENGTH];

void main (void)
{
   .
   .
   .
   init_mem_pool (LINE_POOL, a_lines,

      MAX_LINES, MAX_LINE_LENGTH, TASK_POOL);
   .
   .
   .
}
```

Example of Memory Management in MultiTask!

```
void vPrintFormatTask (void)
{
   char *p_chLine;      /* Pointer to current line */
   .
   .
   .
   /* Format lines and send them to the vPrintOutputTask */
   p_chLine = getbuf (LINE_POOL, WAIT_FOREVER);
   sprintf (p_chLine, "INVENTORY REPORT");
   sndmsg (PRINT_MBOX, p_chLine, PRIORITY_NORMAL);
   p_chLine = getbuf (LINE_POOL, WAIT_FOREVER);
   sprintf (p_chLine, "Date: %02/%02/%02",
      iMonth, iDay, iYear % 100);
   sndmsg (PRINT_MBOX, p_chLine, PRIORITY_NORMAL);
   p_chLine = getbuf (LINE_POOL, WAIT_FOREVER);
   sprintf (p_chLine, "Time: %02:%02", iHour, iMinute);
   sndmsg (PRINT_MBOX, p_chLine, PRIORITY_NORMAL);
   .
   .
   .
}
```

Example of Memory Management in MultiTask!

```
void vPrintOutputTask (void)
{
   char *p_chLine;
   while (TRUE)
   {
      /* Wait for a line to come in. */
      p_chLine = rcvmsg (PRINT_MBOX, WAIT_FOREVER);

      !! Do what is needed to send the line to the printer

      /* Free the buffer back to the pool */
      relbuf (LINE_POOL, p_chLine);
   }
}
```

Interrupt Routines in an RTOS

- Rules for ISR (not for tasks):
- Rule 1: ISR must not call any RTOS function that might block the caller
  - Must NOT:
    - get semaphores
    - read from empty queues, mailboxes, etc.
    - wait for event, …
  - Must run to completion to reset hardware to be ready for next interrupt

Interrupt Routines in an RTOS

- Rule 2: ISR may not call any RTOS function that cause task switching, unless RTOS knows that it is an ISR (& thus will not switch task)
  - May not
    - write to mailboxes, queues on which tasks may be waiting
    - set events
    - release semaphores, …

Rule 1: No Blocking

Figure 7.12 Interrupt Routines *Cannot* Use Semaphores

ISR should not get semaphore!!!

```
static int iTemperatures[2];

void interrupt vReadTemperatures (void)
{
    GetSemaphore (SEMAPHORE_TEMPERATURE);   /***NOT ALLOWED***/
    iTemperatures[0] = !! read in value from hardware;
    iTemperatures[1] = !! read in value from hardware;
    GiveSemaphore (SEMAPHORE_TEMPERATURE);
}

void vTaskTestTemperatures (void)
{
    int iTemp0, iTemp1;

    while (TRUE)
    {
        GetSemaphore (SEMAPHORE_TEMPERATURE);
        iTemp0 = iTemperatures[0];
        iTemp1 = iTemperatures[1];
        GiveSemaphore (SEMAPHORE_TEMPERATURE);
        if (iTemp0 != iTemp1)
            !! Set off howling alarm;
    }
}
```

Rule 1: No Blocking

- Task is running (semaphore held)
- Interrupt occurs, ISR tries to get semaphore
- ISR is blocked, task is blocked
- Semaphore is never released
- All lower-priority tasks are also blocked
- One-armed deadly embrace
  - ISR vReadTemperatures() interrupts task vTaskTestTemperatures()

Post Queue in ISR? (Eg in VRTX)

Figure 7.13 Legal Uses of RTOS Functions in Interrupt Routines

```
/* Queue for temperatures. */
int iQueueTemp;

void interrupt vReadTemperatures (void)
{
   int aTemperatures[2];      /* 16-bit temperatures. */
   int iError;

   /* Get a new set of temperatures. */
   aTemperatures[0] = !! read in value from hardware;
   aTemperatures[1] = !! read in value from hardware;

   /* Add the temperatures to a queue. */
   sc_qpost (iQueueTemp,
      (char *) ((aTemperatures[0] << 16) | aTemperatures[1]),
      &iError);
}
```

Post Queue in ISR? (Eg in VRTX)

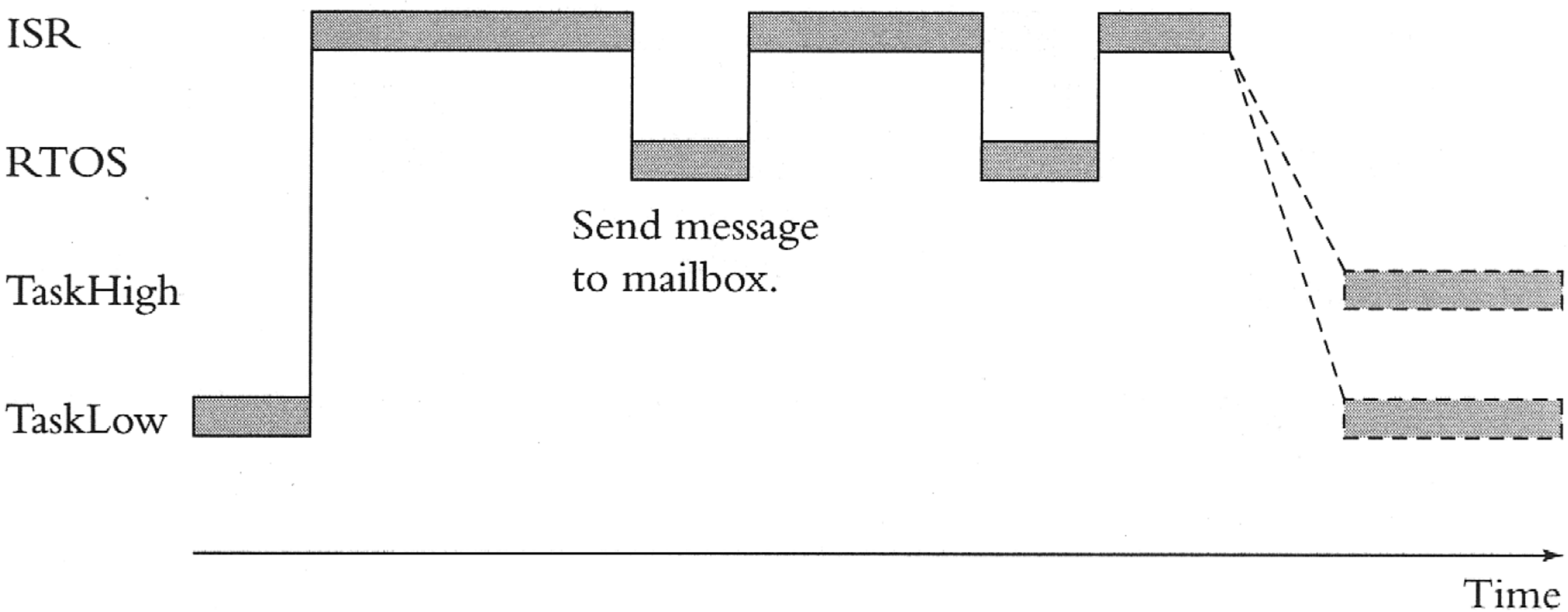
```
void vMainTask (void)
{
   long int lTemps;   /* 32 bits; the same size as a pointer. */
   int aTemperatures[2];
   int iError;

   while (TRUE)
   {
      lTemps = (long) sc_qpend (iQueueTemp, WAIT_FOREVER,
         sizeof(int), &iError);
      aTemperatures[0] = (int) (lTemps >> 16);
      aTemperatures[1] = (int) (lTemps & 0x0000ffff);
      if (aTemperatures[0] != aTemperatures[1])
         !! Set off howling alarm;
   }
}
```

Post Queue in ISR?

- No problem!
- Posting to a queue is non-blocking!
- ISR can post a queue

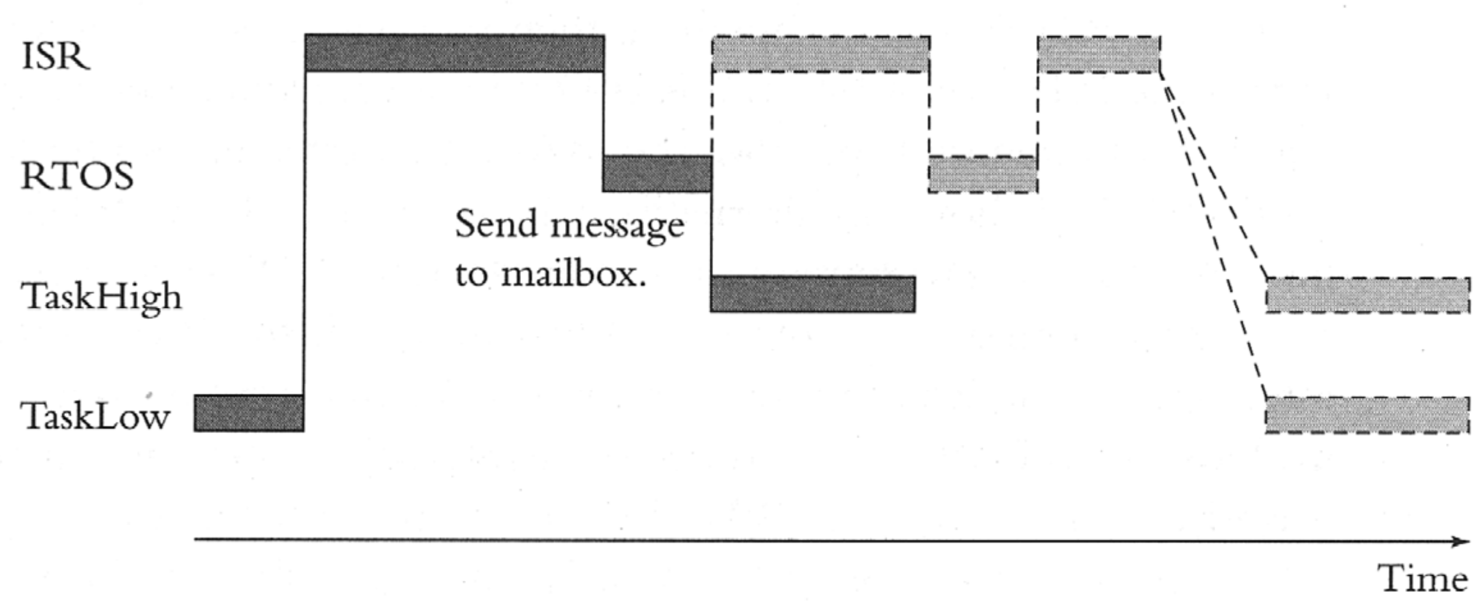
Rule 2: No RTOS Calls without Fair Warning

- A naïve view



Rule 2: No RTOS Calls without Fair Warning

What would really happen!

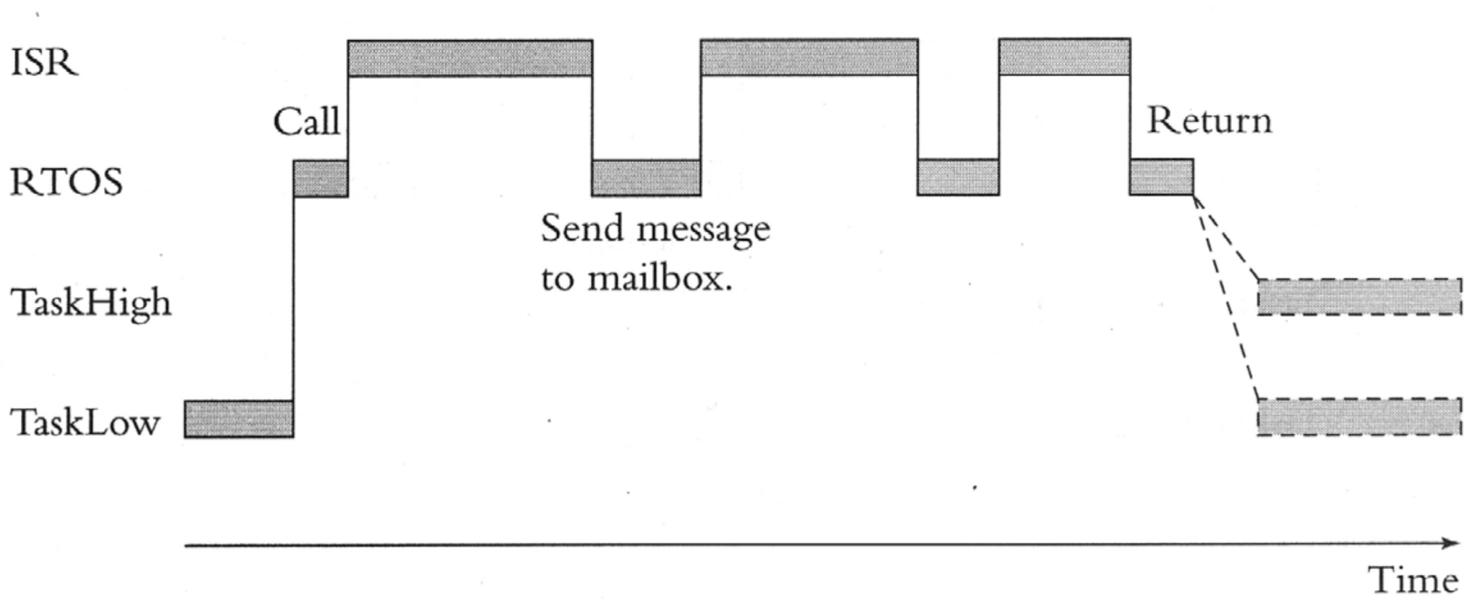


RTOS unblocks a high priority task, is unaware of ISRs, switches to high priority task, ISR is delayed!

Rule 2: No RTOS Calls without Fair Warning

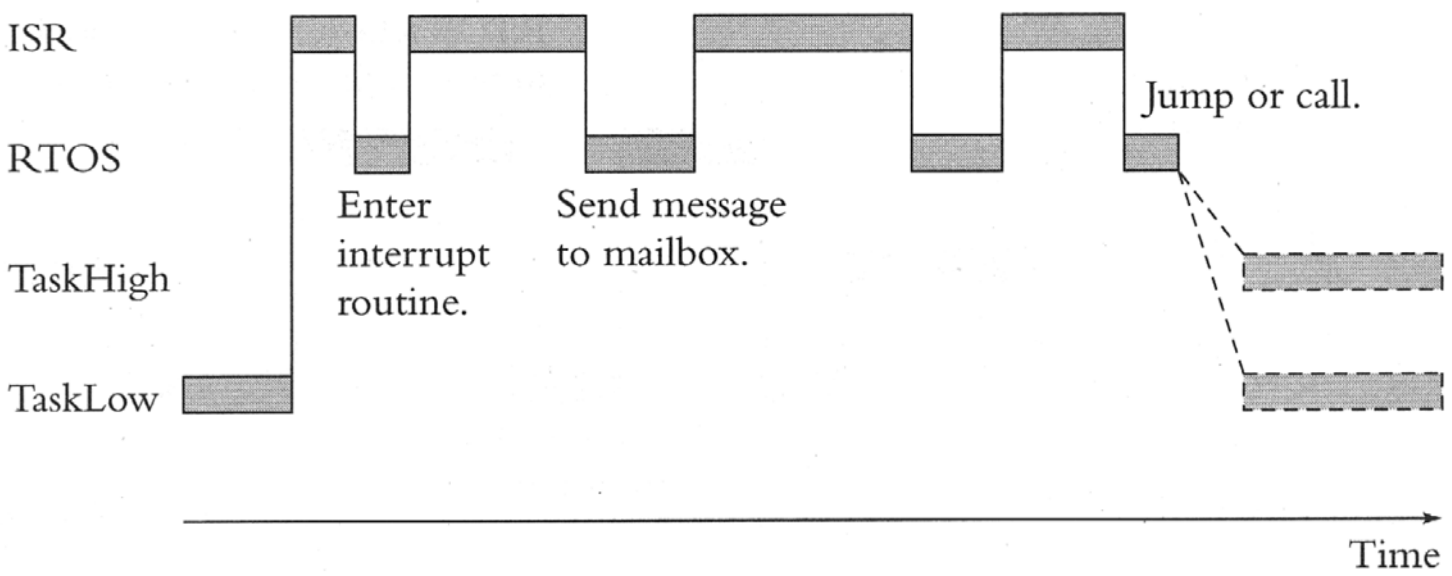
- Solutions:
- **Plan A**: Let RTOS know those functions are ISRs, need to register ISRs and which hardware interrupt corresponds to which ISR
- **Plan B**: In ISR, call a function to let RTOS know that we're in ISR (suspend task switching temporarily), jump or call at end of ISR (to switch to tasks)
- **Plan C**: Separate set of functions to ISR, which always return to ISR

Rule 2: No RTOS Calls without Fair Warning



Plan A: let RTOS know about ISRs, hardware interrupts

Rule 2: No RTOS Calls without Fair Warning



Plan B: suspend scheduler in ISR

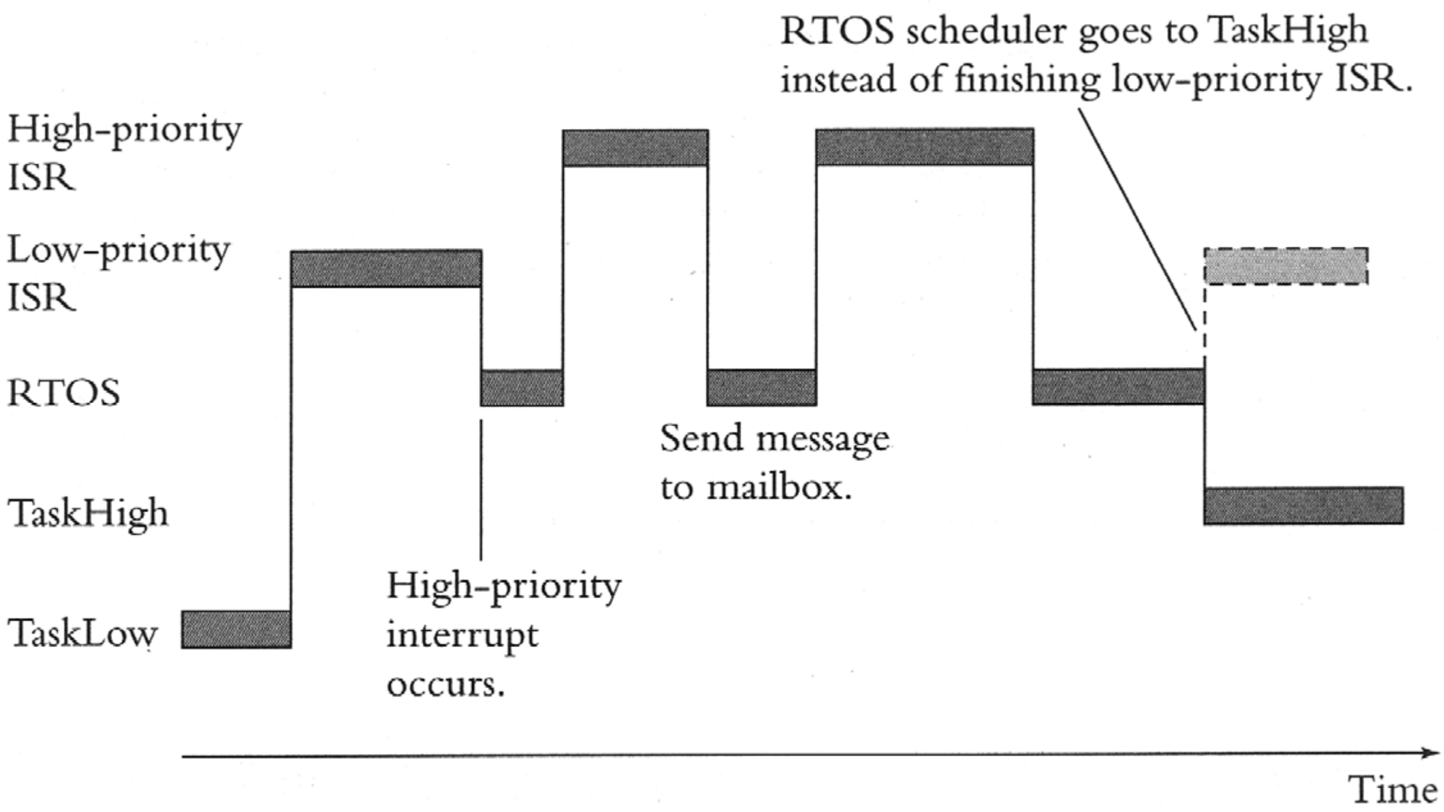
Rule 2: No RTOS Calls without Fair Warning

- Plan C:
- Call **OSISRSemPost()** instead of **OSSemPost()**
- OSISRSemPost() always return to ISR

Rule 2 and Nested Interrupts

- **Nest Interrupts**: Higher priority interrupt interrupts low-priority ISR
- When higher priority ISR finishes, it must return to low-priority ISR and not to another ready task (otherwise low priority ISR will be delayed!)
- Must suspend scheduler until all nested ISRs have finished execution

Rule 2 and Nested Interrupts



Interrupt Handling in eCos

- Interrupt handling in eCos is divided into
  - ISR: Interrupt Service Routine
    - Only the most urgent handling
    - Cannot signal a semaphore via a non-blocking call
    - Higher priority than DSR
    - Eg: character saving
  - DSR: Deferred Service Routine
    - More computing intensive handling
    - Can signal semaphores
    - Higher priority than all tasks
    - Eg: command construction

Interrupt-Driven Blinking LED in eCos (main program)

```
#include <cyg/kernel/kapi.h>
#include "timer.h"
#include "led.h"
/* Declare the ISR variables. */
cyg_handle_t timerInterruptHdl;
cyg_interrupt timerInterruptObj;
cyg_vector_t timerInterruptVector = TIMER1_INT;
cyg_sem_t ledToggleSemaphore;
void cyg_user_start(void) {
    /* Configure the green LED control pin. */    ledInit( );
    /* Create semaphore for task signaling, initialized as 0 */
    cyg_semaphore_init(&ledToggleSemaphore, 0);
    /* Create the LED task. */
    cyg_thread_create(LED_TASK_PRIORITY, blinkLedTask, (cyg_addrword_t)0,
          "LED Task", (void *)ledTaskStack, LED_TASK_STACK_SIZE, &ledTaskHdl, &ledTaskObj);
    /* Notify the scheduler to start running the task. */   cyg_thread_resume(ledTaskHdl);
    /* Initialize the interrupt for the timer. */
    cyg_interrupt_create(timerInterrup  tVector, 0, 0, timerIsr, timerDsr,
          &timerInterruptHdl, &timerInterruptObj);
    cyg_interrupt_attach(timerInterruptHdl);
    cyg_interrupt_acknowledge(timerInterruptVector);
    cyg_interrupt_unmask(timerInterruptVector);
    /* Initialize the timer registers. */         timerInit( ); }
```

Interrupt-Driven Blinking LED in eCos (timerIsr)

```
#include <cyg/hal/hal_intr.h>
uint32_t timerIsr(cyg_vector_t vector, cyg_addrword_t data) {

   /* Block timer interrupt from occurring until DSR runs. */
   cyg_interrupt_mask(timerInterruptVector);

   /* Acknowledge the interrupt in interrupt controller & in
   timer peripheral. */
   cyg_interrupt_acknowledge(timerInterruptVector);
   TIMER_STATUS_REG = TIMER_1_MATCH;

   /* Inform OS that the interrupt is handled by this ISR and
   that the DSR needs to run. */
   return (CYG_ISR_HANDLED | CYG_ISR_CALL_DSR);
}
```

Interrupt-Driven Blinking LED in eCos (timerDsr)

```
void timerDsr(cyg_vector_t vector, cyg_ucount32 count,
  cyg_addrword_t data) {
  /* Signal the task to toggle the LED. */
  cyg_semaphore_post(&ledToggleSemaphore);

  /* Set the new timer interval. */
  TIMER_1_MATCH_REG = (TIMER_COUNT_REG +
  TIMER_INTERVAL_500MS);

  /* Enable processing of incoming timer interrupts. */
  cyg_interrupt_unmask(timerInterruptVector);
}
```

Interrupt-Driven Blinking LED in eCos (blinkLedTask)

```
void blinkLedTask(cyg_addrword_t data) {
  while (1) {
      /* Wait for signal to toggle the LED. */
      cyg_semaphore_wait(&ledToggleSemaphore);

      /* Change the state of the green LED. */
      ledToggle( ); }
}
```