Operating systems and concurrency (B11)

David Kendall

Northumbria University

Introduction

- Brief look at some remaining features of uC/OS-II API
  - Soft timers
  - Message mailboxes
  - Message queues

Create soft timer

```
OS_TMR *OSTmrCreate(INT32U dly,
                    INT32U period,
                    INT8U opt,
                    OS_TMR_CALLBACK callback,
                    void *callback_arg,
                    INT8U *pname,
                    INT8U *perr);
```

- `OSTmrCreate()` allows us to create a soft timer
- `dly` : in ONE_SHOT mode this specifies when the timer expires; PERIODIC mode it specifies the initial delay until the timer enters its periodic behaviour
- `period` : specifies the period; set to 0 if one-shot
- Units of `dly` and `period` are defined by the value of `OS_TMR_CFG_TICKS_PER_SEC` in the OS configuration file.
  - e.g. if `OS_TMR_CFG_TICKS_PER_SEC` is 10 and `period` is 20, then the timer will expire periodically every 2 seconds.
- `opt` : `OS_TMR_OPT_PERIODIC` or `OS_TMR_OPT_ONE_SHOT`

Create soft timer

```
OS_TMR *OSTmrCreate(INT32U dly,
                    INT32U period,
                    INT8U opt,
                    OS_TMR_CALLBACK callback,
                    void *callback_arg,
                    INT8U *pname,
                    INT8U *perr);
```

- `callback` : specifies the address of a callback function, defined by `void MyCallback (void *ptmr, void *callback_arg);`
- `pname` : allows you to name your timer for debugging
- `perr` : pointer to an error code

Start timer

```
BOOLEAN OSTmrStart(OS_TMR *ptmr,
                   INT8U *perr);
```

- `ptmr` : pointer to the timer that you want to start
- `perr`: pointer to an error code
- returned value: true if the timer was started; false if an error occurred
- If `OSTmrStart` called when timer is already running, timer is simply restarted.
- Can use `OSTmrStateGet` to discover current state of timer (see example later)

Stop timer

```
BOOLEAN  OSTmrStop(OS_TMR  *ptmr,
                   INT8U  opt,
                   void  *callback_arg,
                   INT8U  *perr);
```

- `ptmr` Is a pointer to the timer you want to stop. This 'handle' was returned to your application when you called `OSTmrStart()` and uniquely identifies the timer.
- `opt` specifies whether you want the timer to:
  1. `OS_TMR_OPT_NONE`: Do NOT call the callback function.
  2. `OS_TMR_OPT_CALLBACK`: Call the callback function and pass it the callback argument specified when you started the timer (see `OSTmrCreate()`).
  3. `OS_TMR_OPT_CALLBACK_ARG`: Call the callback function BUT pass it the callback argument specified in the `OSTmrStop()` function INSTEAD of the one defined in `OSTmrCreate()`.

Stop timer ctd.

- `callback_arg` If you set `opt` to `OS_TMR_OPT_CALLBACK_ARG` then this is the argument passed to the callback function when it's executed.
- `perr` a pointer to an error code

Timer example (softtimer.c)

```
OS_TMR    *ledTimer;
INT8U     osStatus;
...
ledTimer = OSTmrCreate(50, 0, OS_TMR_OPT_ONE_SHOT,
                       toggleFlashing, (void *)0,
                       (uint8_t *)0, &osStatus);
...
if (isButtonPressed(BUT_1) &&
    !(OSTmrStateGet(ledTimer, &osStatus) ==
      OS_TMR_STATE_RUNNING)) {
  (void)OSTmrStart(ledTimer, &osStatus);
}
...
OSTmrStop(ledTimer, OS_TMR_OPT_NONE,
          (void *)0, &osStatus);
...
void toggleFlashing(void *ledTimer, void *pdata) {
  flashing = !flashing;
}
```

Message mailboxes: Create

- A message mailbox allows tasks or ISRs to send a pointer-sized variable (message) to a task.
- Create mailbox

```
OS_EVENT *OSMboxCreate(void *pmsg);
```

- `pmsg` is used to initialize the contents of the mailbox. The mailbox is empty when `pmsg` is a `NULL` pointer. The mailbox initially contains a message when `pmsg` is non-NULL.

Message mailboxes: Pend

- Pend

```
void *OSMboxPend(OS_EVENT *pevent,
                 INT32U timeout,
                 INT8U *perr);
```

- `OSMboxPend()` is used when a task expects to receive a message from another task or an ISR.
- The message received is a pointer-sized variable, and its use is application specific.
- If a message is present in the mailbox when `OSMboxPend()` is called, the message is retrieved, the mailbox is emptied, and the retrieved message is returned to the caller.
- If no message is present in the mailbox, `OSMboxPend()` suspends the current task until either a message is received or a user-specified timeout expires.

Message mailboxes: Post

- Post

```
INT8U  OSMboxPost(OS_EVENT  *pevent,
                  void  *pmsg);
```

- `OSMboxPost()` sends a message to a task through a mailbox.
- If a message is already in the mailbox, an error code is returned indicating that the mailbox is full. `OSMboxPost()` then immediately returns to its caller, and the message is not placed in the mailbox.
- If any task is waiting for a message at the mailbox, the highest priority task waiting receives the message.
- If the task waiting for the message has a higher priority than the task sending the message, the higher priority task is resumed, and the task sending the message is suspended – context switch.

Message mailboxes: Broadcast

- Post with options

```
INT8U  OSMboxPostOpt(OS_EVENT  *pevent,
                     void  *pmsg,
                     INT8U  opt);
```

- Behaves just like `OSMboxPost()` except `opt` can be one or more of:
  - `OS_POST_OPT_NONE` – behave like `OSMboxPost()`
  - `OS_POST_OPT_BROADCAST` – send message to *all* waiting tasks
  - `OS_POST_OPT_NO_SCHED` – don't call the scheduler after posting the message
- The execution time of `OSMboxPostOpt()` depends on the number of tasks waiting on the mailbox if you set opt to `OS_POST_OPT_BROADCAST`.

Message queues: Create

- Create message queue

```
OS_EVENT *OSQCreate(void **start,
                    INT8U size);
```

- `start` is the base address of the message storage area. A message storage area is declared as an array of pointers to voids.
- `size` is the size (in number of entries) of the message storage area.

Message queues: Pend

- Pend on message queue

```
void *OSQPend(OS_EVENT *pevent,
              INT32U timeout,
              INT8U *perr);
```

- `OSQPend()` is used when a task wants to receive messages from a queue. The messages are sent to the task either by an ISR or by another task.
- If at least one message is present at the queue when `OSQPend()` is called, the message is retrieved and returned to the caller.
- If no message is present at the queue, `OSQPend()` suspends the current task until either a message is received or a user-specified timeout expires.

Message queues: Post

- Post a message to a message queue

```
INT8U  OSQPost(OS_EVENT  *pevent,
               void  *pmsg);
```

- `OSQPost()` sends a message to a task through a queue.
- If the message queue is full, an error code is returned to the caller. In this case, OSQPost() immediately returns to its caller, and the message is not placed in the queue.
- If any task is waiting for a message at the queue, the highest priority task receives the message. If the task waiting for the message has a higher priority than the task sending the message, the higher priority task resumes, and the task sending the message is suspended; that is, a context switch occurs.
- Message queues are first-in first-out (FIFO), which means that the first message sent is the first message received.

Acknowledgements

- Labrosse, J., MicroC/OS-II: The Real-time Kernel, CMP, 2002