

Event Multiplexing

INTRODUCTION

Multiplexing generally describes the usage of a common carrier for several signals. This usually involves a multiplexer which takes several inputs and combines them into a single unit and a demultiplexer which takes the combined signals and separates them into the original form.

In the case of event multiplexing, a task may use an event multiplex object (multiplexer) to wait for the occurrence of one of several different events (signals). The task code can then process the occurred events (demultiplexer).

FUNCTIONALITY

The following functionality is usually provided by the RTOS to use event multiplexing:

1. Create an event multiplex object which is associated to the task which uses it to wait on events.
2. Add/register events of certain objects to the event multiplex object the task is interested in.
3. Change events that were added to the event multiplex object.
4. Remove events, if an event is no longer of interest for the task.
5. Wait for the occurrence of events. While waiting for the registered events to occur, the task can be blocked and has therefore not consumed any CPU time.
6. Demultiplex the occurred events when the task is notified about the occurrence of a registered event. The event multiplex object provides the information in which events occurred, so the task may process only those events.

AN EXAMPLE OF EVENT MULTIPLEXING

A use case of event multiplexing could be processing user inputs from different devices. Let us assume we have a task K which takes the input of the keyboard, does some initial processing like debouncing, and puts the characters into a queue KeyIn. A second task M takes the input of the mouse, does some initial processing like debouncing or double click detection and puts the commands into another queue MouseIn. A third task P should process the inputs of the several devices and therefore must wait until either of those inputs happen again. Task P could now use an event multiplex object to register the message waiting events of queue KeyIn and queue MouseIn. The task code would then wait for one of the events to happen and process them as they occur. Figure 1 shows a possible flowchart.

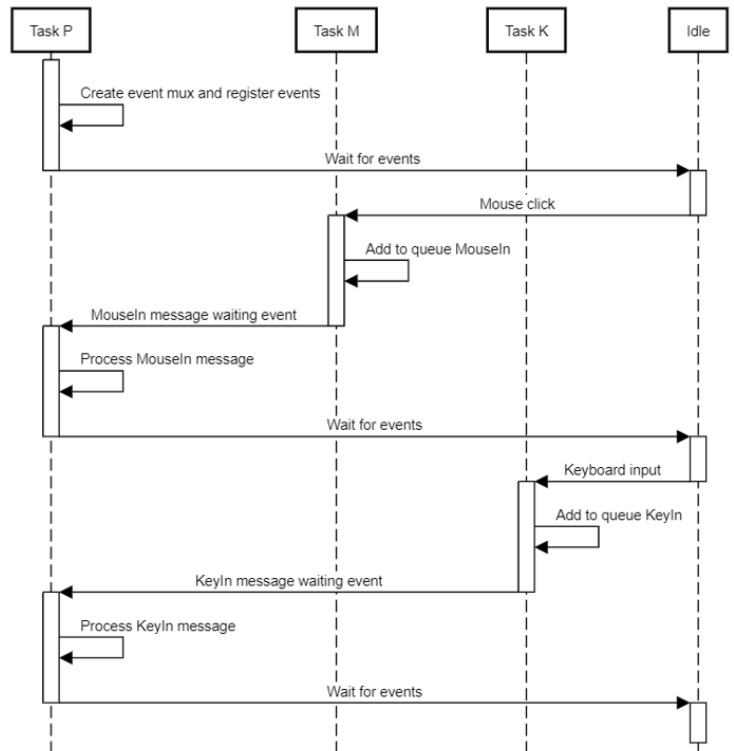


Figure 1: A Case Study of Event Multiplexing

EVENT MULTIPLEXING & SAFERTOS®

The SAFERTOS API has the following functions associated with event multiplexing to provide the mentioned functionality.

- **xEvtMplxCreate**

Create an event multiplex object. At creation the event multiplex object needs a buffer to store the occurred events, as `evtMplxObjectEvents` and an owner task.

- **xEvtMplxAddObjectEvents**

Add an object-event pair to the event multiplex (registers the event).

- **xEvtMplxModifyObjectEvents**

Modify the event of an object-event pair in the event multiplex.

- **xEvtMplxRemoveObjectEvents**

Delete an object-event pair from the event multiplex.

- **xEvtMplxWait**

Block the task until at least one of the events registered in the event multiplex object occurred. A timeout time can be specified to prevent infinite blocking. Upon return from the `xEvtMplxWait` call, the task code can read the `evtMplxObjectEvents` buffer to know which events occurred.

SAFERTOS implements 6 event types which can be registered with an event multiplex object. Each event must be associated with a target object. The possible events are:

- `evtmplxQUEUE_MESSAGE_WAITING`
- `evtmplxQUEUE_SPACE_AVAILABLE`
- `evtmplxSEMAPHORE_AVAILABLE`
- `evtmplxMUTEX_AVAILABLE`
- `evtmplxTASK_NOTIFICATION_RECEIVED`
- `evtmplxEVENT_GROUP_BITS_SET`

WITTENSTEIN high **integrity** systems

Worldwide Sales and Support

Americas: +1 408 625 4712

ROTW: +44 1275 395 600

Email: sales@highintegritysystems.com

Web: www.highintegritysystems.com



WITTENSTEIN