11

Signal Manager

11.1 Introduction

The **signal manager** provides the capabilities required for asynchronous communication. The directives provided by the signal manager are:

Name	Directive Description
signal_catch	Establish an ASR
signal_send	Send signal set to a task

11.2 Background

11.2.1 Definitions

The signal manager allows a task to optionally define an **asynchronous signal routine** (ASR). An ASR is to a task what an ISR is to an application's set of tasks. When the processor is interrupted, the execution of an application is also interrupted and an ISR is given control. Similarly, when a signal is sent to a task, that task's execution path will be "interrupted" by the ASR. Sending a signal to a task has no effect on the receiving task's current execution state.

A **signal flag** is used by a task (or ISR) to inform another task of the occurrence of a significant situation. Thirty-two signal flags are associated with each task. A collection of one or more signals is referred to as a **signal set**. A signal set is **posted** when it is directed (or sent) to a task. A **pending signal** is a signal that has

been sent to a task with a valid ASR, but has not been processed by that task's ASR.

11.2.2 A Comparison of ASRs and ISRs

The format of an ASR is similar to that of an ISR with the following exceptions:

- *ISRs are scheduled by the processor hardware. ASRs are scheduled by **RTEMS**.*
- *ISRs do not execute in the context of a task and may invoke only a subset of directives. ASRs execute in the context of a task and may execute any directive.*
- *When an ISR is invoked, it is passed the vector number as its argument. When an ASR is invoked, it is passed the signal set as its argument.*
- *An ASR has a task mode which can be different from that of the task. An ISR does not execute as a task and, as a result, does not have a task mode.*

11.2.3 Building a Signal Set

A signal set is built by a bitwise OR of the desired signals. The set of valid signals is **SIGNAL_0** through **SIGNAL_31**. If a signal is not explicitly specified in the signal set, then it is not present. Signal values are specifically designed to be mutually exclusive, therefore bitwise OR and addition operations are equivalent as long as each signal appears exactly once in the component list.

This example demonstrates the **signal** parameter used when sending the signal set consisting of **SIGNAL_6**, **SIGNAL_15**, and **SIGNAL_31**. The **signal** parameter provided to the **signal_send** directive should be **SIGNAL_6 + SIGNAL_15 + SIGNAL_31**.

11.2.4 Building an ASR's Mode

In general, an ASR's mode is built by a bitwise OR of the desired mode components. The set of valid mode components is the same as those allowed with the **task_create** and **task_mode** directives. A complete list of mode options is provided in the following table:

MODE CONSTANT	MASK CONSTANT	DESCRIPTION
PREEMPT	PREEMPT_MASK	enable preemption
NO_PREEMPT	PREEMPT_MASK	disable preemption
NO_TIMESLICE	TIMESLICE_MASK	disable timeslicing
TIMESLICE	TIMESLICE_MASK	enable timeslicing
ASR	ASR_MASK	enable ASR processing
NO_ASR	ASR_MASK	disable ASR processing
INTERRUPT_LEVEL(0)	INTERRUPT_MASK	enable all interrupts
INTERRUPT_LEVEL(n)	INTERRUPT_MASK	execute at interrupt level n

Mode values are specifically designed to be mutually exclusive, therefore bitwise OR and addition operations are equivalent as long as each mode appears exactly once in the component list. A mode component listed as a default is not required to appear in the mode list, although it is a good programming practice to specify default components. If all defaults are desired, the mode **DEFAULT_MODES** should be specified on this call.

This example demonstrates the **mode** parameter used with the **signal_catch** to establish an ASR which executes at interrupt level three and is non-preemptible. The **mode** should be set to **INTERRUPT_LEVEL(3) + NO_PREEMPT** to indicate the desired processor mode and interrupt level.

11.3 Operations

11.3.1 Establishing an ASR

The **signal_catch** directive establishes an ASR for the calling task. The address of the ASR and its execution mode are specified to this directive. The ASR's mode is distinct from the task's mode. For example, the task may allow preemption, while that task's ASR may have preemption disabled. Until a task calls **signal_catch** the first time, its ASR is invalid, and no signal sets can be sent to the task.

A task may invalidate its ASR and discard all pending signals by calling **signal_catch** with a value of **NULL** for the ASR's address. When a task's ASR is invalid, new signal sets sent to this task are discarded.

A task may disable ASR processing (**NO_ASR**) via the **task_mode** directive. When a task's ASR is disabled, the signals sent to it are left pending to be processed later when the ASR is enabled.

Any directive that can be called from a task can also be called from an ASR. A task is only allowed one active ASR. Thus, each call to **signal_catch** replaces the previous one.

Normally, signal processing is disabled for the ASR's execution mode, but if signal processing is enabled for the ASR, the ASR must be reentrant.

11.3.2 Sending a Signal Set

The **signal_send** directive allows both tasks and ISRs to send signals to a target task. The target task and a set of signals are specified to the **signal_send** directive. The sending of a signal to a task has no effect on the execution state of that task. If the task is not the currently running task, then the signals are left pending and processed by the task's ASR the next time the task is dispatched to run. The ASR is executed immediately before the task is dispatched. If the currently running task sends a signal to itself or is sent a signal from an ISR, its ASR is immediately dispatched to run provided signal processing is enabled.

If an ASR with signals enabled is preempted by another task or an ISR and a new signal set is sent, then a new copy of the ASR will be invoked, nesting the preempted ASR. Upon completion of processing the new signal set, control will return to the preempted ASR. In this situation, the ASR must be reentrant.

Like events, identical signals sent to a task are not queued. In other words, sending the same signal multiple times to a task (without any intermediate signal processing occurring for the task), has the same result as sending that signal to that task once.

11.3.3 Processing an ASR

Asynchronous signals were designed to provide the capability to generate software interrupts. The processing of software interrupts parallels that of hardware interrupts. As a result, the differences between the formats of ASRs and ISRs is limited to the meaning of the single argument passed to an ASR. The ASR should have the following calling sequence and adhere to **Ada calling conventions:**

```
procedure user_routine(
  signals : in      rtems.signal_set
);
```

When the ASR returns to **RTEMS** the mode and execution path of the interrupted task (or ASR) is restored to the context prior to entering the ASR.

11.4 Directives

This section details the signal manager's directives. A subsection is dedicated to each of this manager's directives and describes the calling sequence, related constants, usage, and status codes.

11.4.1 SIGNAL_CATCH - Establish an ASR

CALLING SEQUENCE:

```
procedure signal_catch (
   asr_handler : in     rtems.address;
   mode_set    : in     rtems.mode;
   result      :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL always successful

DESCRIPTION:

This directive establishes an asynchronous signal routine (ASR) for the calling task. The **asr_handler** parameter specifies the entry point of the ASR. If **asr_handler** is **NULL**, the ASR for the calling task is invalidated and all pending signals are cleared. Any signals sent to a task with an invalid ASR are discarded. The **mode** parameter specifies the execution mode for the ASR. This execution mode supersedes the task's execution mode while the ASR is executing.

NOTES:

This directive will not cause the calling task to be preempted.

The following task mode constants are defined by **RTEMS**:

MODE CONSTANT	MASK CONSTANT	DESCRIPTION
PREEMPT	PREEMPT_MASK	enable preemption
NO_PREEMPT	PREEMPT_MASK	disable preemption
NO_TIMESLICE	TIMESLICE_MASK	disable timeslicing
TIMESLICE	TIMESLICE_MASK	enable timeslicing
ASR	ASR_MASK	enable ASR processing
NO_ASR	ASR_MASK	disable ASR processing
INTERRUPT_LEVEL(0)	INTERRUPT_MASK	enable all interrupts
INTERRUPT_LEVEL(n)	INTERRUPT_MASK	execute at interrupt level n

11.4.2 SIGNAL_SEND - Send signal set to a task

CALLING SEQUENCE:

```
procedure signal_send (
   id         : in     rtems.id;
   signal_set : in     rtems.signal_set;
   result     :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	signal sent successfully
INVALID_ID	task id invalid
NOT_DEFINED	ASR invalid

DESCRIPTION:

This directive sends a signal set to the task specified in **id**. The **signal_set** parameter contains the signal set to be sent to the task.

If a caller sends a signal set to a task with an invalid ASR, then an error code is returned to the caller. If a caller sends a signal set to a task whose ASR is valid but disabled, then the signal set will be caught and left pending for the ASR to process when it is enabled. If a caller sends a signal set to a task with an ASR that is both valid and enabled, then the signal set is caught and the ASR will execute the next time the task is dispatched to run.

NOTES:

Sending a signal set to a task has no effect on that task's state. If a signal set is sent to a blocked task, then the task will remain blocked and the signals will be processed when the task becomes the running task.

Sending a signal set to a global task which does not reside on the local node will generate a request telling the remote node to send the signal set to the specified task.