OS Awareness Manual RealTimeCraft

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents ..... 

OS Awareness Manuals ..... 

OS Awareness Manual RealTimeCraft **1**

History **2**

Brief Overview of Documents for New Users 2

Configuration **3**

Quick Configuration 3

Hooks in XEC 68 4

Features **5**

Display of Kernel Resources 5

Function Runtime Statistics 5

Task Runtime Analysis 5

Task State Analysis 6

System Call Trace 6

XEC 68 Commands **7**

TASK.DeLaY Delay table 7

TASK.MailBoX Mailbox table 7

TASK.SEMaphore Semaphore table 8

TASK.SysCall Execute XEC 68 system call 8

TASK.TASK Task table 8

Frequently-Asked Questions **8**

OS Awareness Manual RealTimeCraft

Version 18-Nov-2020

History

28-Aug-18	The title of the manual was changed from “RTOS Debugger for *<x>*” to “OS Awareness Manual *<x>*”.

Brief Overview of Documents for New Users

Architecture-independent information:

- **“Debugger Basics - Training”** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **“T32Start”** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **“General Commands”** (general_ref_*<x>*.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- **“Processor Architecture Manuals”**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.
- **“OS Awareness Manuals”** (rtos_*<os>*.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

The OS Awareness for RealTime Craft supports the following features:

Configuration

The PRACTICE file 'prtc.cmm' patches XEC 68 and configures the OS Awareness. The macros, defined at the beginning of the file, specify the address of XEC 68, the address of the current-tcb pointer, and the vectors which are used to enter the kernel. Be sure tat emulation memory has been mapped to the address space of the system tables. Otherwise the display functions will not work with dual-port access.

Format: **TASK.CONFIG rtc** *<magic_address> <sleep> <data_area> <config_table> <system_call_gate>*

<magic_address>	Specifies a memory location that contains the current running task.
<sleep>	The argument for *<sleep>* is currently not used. Specify '0'.
<data_area>	Address of the XEC data area (label "`XecDataAd`")
<config_table>	Kernel configuration table address (label "`ConfigTable`")
<system_call_gate>	Address of the system call patch routine

If the task selective debugging features are not used, the patching of the kernel is not required. The PRACTICE script 'prtc.cmm' can make the required patches to XEC 68 and configure the display command:

```
DO prtc nopatch       ; configures only display functions
                      ; no patches are made

DO prtc               ; patched XEC 68 for task selective debugging
```

When patching is required the patch area in the PRACTICE script file must be modified to point to an unused memory area.

The demo script 'rtc.cmm' in the '~~/demo/m68k/kernel/rtc' directory can be started with the same parameters.

Quick Configuration

To access all features of the OS Awareness you should follow the following roadmap:

1. Run the demo script (~~/demo/m68k/kernel/rtc/rtc.cmm) without any patching. Start the demo with 'do rtc nopatch'. The result should be a list of tasks, which change continuously their state.
2. Run the demo script with patching.
3. Try the analyzer demo programs (tasksc, taskstat and taskfunc).
4. Make a copy of the 'prtc.cmm' PRACTICE file. Modify the file according to your application. This can be changing XEC Entry Trap or choosing a different memory area for the patches.
5. Run the modified version in your application **without patching** (with 'nopatch' argument). This should allow you to display the kernel resources and use most of the analyzer features (except the system call display).
6. Run your application with patching.

Hooks in XEC 68

To determine the entry of a task, the patching of XEC 68 is required. All returns to the task context (usually RTE instructions) are patched to pass control to the multitask monitor. The patch writes the current executing tcb address to the magic-word of the OS Awareness.

The entries to XEC 68 are patched directly in the vector table. The patches write the value 1 to the magic-word and run to a breakpoint.

Features

Display of Kernel Resources

The resources are usually read by dual port memory. For correct operation memory must be mapped at all places where XEC 68 holds its tables. The following information can be displayed:

- Tasks (TASK)
- Mailboxes (MAILBOX)
- Semaphores (SEMAPHORE)
- Delays (DELAY)

Function Runtime Statistics

All function related statistic and time chart functions can be used with or without patching the kernel. The difference is whether the kernel will be seen like another task or as part of the task who called the kernel. Task selective debugging should not be used when statistics are made, as this would cause an error in the measurements. The task switch can be displayed in the analyzer list with the **List.TASK** keyword. The example script 'taskfunc.cmm' makes a task-selective performance analysis for the demo application.

Analyzer.STATistic.TASKFunc	Display function runtime statistic
Analyzer.STATistic.TASKTREE	Display functions as tree
Analyzer.Chart.TASKFunc	Display function time chart
Analyzer.List	List.TASK FUNC Display function nesting in analyzer

Task Runtime Analysis

The time spend in a task can be analyzed by marking the access to a word holding a pointer to the current tasks tcb. This can either be in the kernel or in the patch programs. In the first case the runtime in the kernel will be added to the last task which called the kernel. If the 'magic' word in the patch program is marked, the kernel is treated like another task. Task selective debugging should not be used when statistics are made, as this would cause an error in the measurements. The example script 'taskfunc.cmm' can be used to make the measurement for this analysis.

Analyzer.STATistic.TASK	Display task runtime statistic
Analyzer.Chart.TASK	Display task runtime time chart

Task State Analysis

The time different tasks are is a certain state (running, ready, suspended or waiting) can be displayed as a statistic or in graphical form. This feature is implemented by recording all accesses to the status words of all tasks. Additionally the accesses to the current tcb pointer or the magic word are traced. This is required as the status of a task makes no difference between 'running' and 'ready'. The breakpoints to the tcb status words are set by the **TASK.TASKState** command. The example script 'taskstat.cmm' makes a task state analysis with the demo application.

Analyzer.STATistic.TASKState	Display task state statistic
Analyzer.Chart.TASKState	Display task state time chart

System Call Trace

System calls with parameters can be traced and used as trigger criteria by the state analyzer. This feature requires patching the kernel. The example script 'tasksc.cmm' makes a trace of system calls and task switches in the demo application.

Analyzer.List	List.TASK	Display system calls

TASK.DeLaY

Delay table

Format: **TASK.DeLaY**

Displays the delay table of XEC 68.

TASK.MailBoX

Mailbox table

Format: **TASK.MailBoX** [<*mailbox_id*>]

Displays the mailbox table of XEC 68. With a mailbox id as an argument it displays one mailbox in detail, i.e. with it's associated message queue.

TASK.SEMaphore — Semaphore table

Format: **TASK.SEMaphore**

Displays the semaphore table of XEC 68.

TASK.SysCall — Execute XEC 68 system call

Format: **TASK.SysCall** *<function> <d1.w> <d2.l> <d3.l>*

<function>: **StopTask** | **CurrTask** | **TaskState** | **TaskPrio** | **ChgPrio**
SigEvent | **WaitEvent** | **EventsOcc** | **ClrEvents** | **V**
TESTP | **SEND** | **TestRec** | **ClearSem** | **ClearMBx**

Executes a system call. The function can only be executed, when the emulation is stopped in a regular task. The function runs under the current task ID. Take care that a preempted task switch (e.g. in waiting conditions) can hang the emulation. So no task switch condition should be given in a system call.

```
TASK.SysCall stoptask 3

TASK.SysCall send 1 0 12345678
```

TASK.TASK — Task table

Format: **TASK.TASK** [*<task_id>*]

Displays the task-table of XEC 68. With a task ID as an argument it displays one task in detail.

Frequently-Asked Questions

No information available