OS Awareness Manual RTX51 tiny

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents ..... 

OS Awareness Manuals ..... 

OS Awareness Manual RTX51 tiny **1**

History **2**

Brief Overview of Documents for New Users 2

Configuration **3**

Quick Configuration 3

Hooks in RTX51 tiny 4

Features **5**

Display of Kernel Resources 5

Function Runtime Statistics 5

Task Runtime Analysis 5

Task State Analysis 6

System Call Trace 6

RTX51 tiny Commands **7**

TASK.StateTab Task state table 7

TASK.SysCall Execute RTX51 tiny system call 7

TASK.TaskInfo Task information table 8

Frequently-Asked Questions **8**

OS Awareness Manual RTX51 tiny

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 RTX51 tiny supports the following features:

Configuration

The PRACTICE file 'prtx51t.cmm' configures the OS Awareness for RTX51 tiny. Be sure that 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 rtx51t** *<magic_address> <sleep> <max_task> <status_table> <system_call_gate> <system_call_parameter_area>*

<magic_address>	When patched, *<magic_address>* is the address of magic location in external memory. Without patch, specify “0”.
<sleep>	The argument for *<sleep>* is currently not used. Specify “0”.
<max_task>	The first argument passes the highest task number to the configuration.
<status_table>	The second argument specifies the address of the internal task status table (label “`?RTX_TASKSTATUS`”).
<system_call_gate>	This argument is the address of the system call patch routine (in program memory) and the address (in external data memory), where the system call parameters are exchanged. Without patch specify “0”.
<system_call_parameter_area>	The last argument is the address in external memory, at which the system call parameters are exchanged.

The PRACTICE script 'prtx51t.cmm' can make the required configurations:

```
do prtx51t        ; configure the OS Awareness for full support
```

Quick Configuration

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

1. Run the PRACTICE demo script (~~/demo/i51/kernel/rtxtiny/rtx51t.cmm). Start the demo with 'do rtx51t' and 'go'. The result should be a list of tasks, which continuously change their state.
2. Try the analyzer demo scripts (taskstat and taskfunc).
3. Make a copy of the 'prtx51t.cmm' PRACTICE file. Modify the file according to your application. This can be using a different memory area for the patches.
4. Run your application with patching.

Hooks in RTX51 tiny

To determine the entry of a task, patching of RTX51 tiny is required. The patching routines are called in the task switch routine (not system calls!) of RTX51 tiny, to detect the task switches correctly. This is done with absolute address distances, therefore you have to care about using different RTX51 tiny versions!. The patch writes the current running task ID to the magic word of the OS Awareness. The manual system call does a direct function call to the desired routine.

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 RTX51 tiny holds its tables. The following information can be displayed:

- task information (TI)
- task state table (ST)

Function Runtime Statistics

All function related statistic and time chart functions can be used with task specific information. 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 the current task ID. This is normally held in the internal memory. No emulator access to the internal memory is possible while the application is running. Therefore a patch is needed, which copies the internal task ID ("magic") to an external location. The kernel is treated like another task. The example PRACTICE 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. The breakpoints to the task status words, which are copied by the patch from internal to external memory, can be set with the command 'TASK.TASKState'. 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

No System Call Trace is available yet. As all system calls are direct function calls, a function parameter trace will do quite the same job.

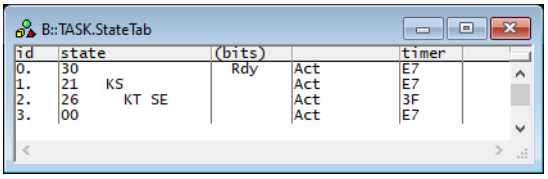
Data.TAGFunc	Tag code for analysis
Analyzer.List	FUNC Var List.TASK Display function parameters

TASK.StateTab — Task state table

Format: **TASK.StateTab**

Displays a detailed state table of the tasks.

NOTE: See 'TASK.TaskInfo'.



TASK.SysCall — Execute RTX51 tiny system call

Format: **TASK.SysCall** *<function> <parameters>*

<function>: **R_Task_Id** | **Create_T** | **Delete_T** | **Send_Sig** | **Isr_S_Sig** **Clr_Sig** | **Wait**

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.

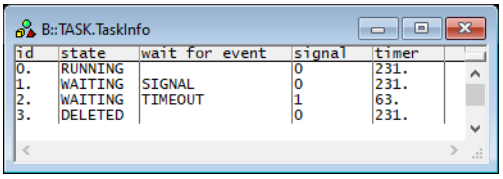
```
task.sc r_task_id

task.sc send_sig 2
```

Format: **TASK.TaskInfo**

Displays the Task table of RTX51 tiny.

NOTE: The task states and timers are calculated from a copy of the internal task table. This copy can slightly differ from the original table. Extend the 'TASK.TI' window, so that it shows at least one empty line at the end. If the application is then stopped, the internal table will be used for calculation.



Frequently-Asked Questions

No information available