LAUTERBACH
DEVELOPMENT TOOLS

OS Awareness Manual Cmicro

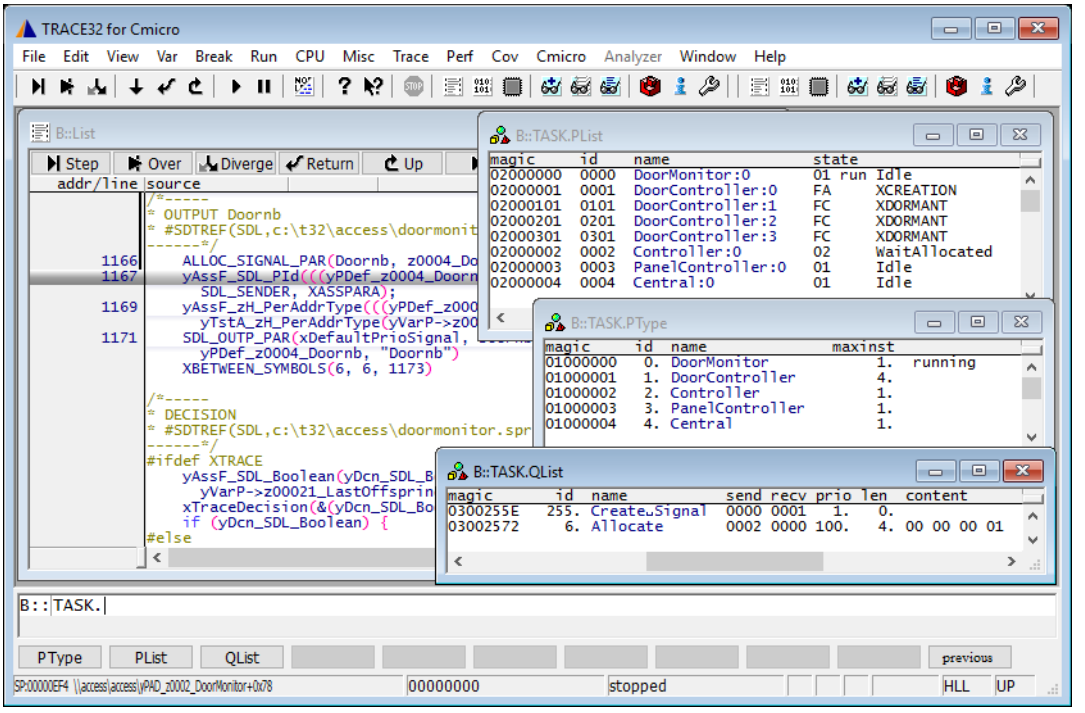
MANUAL

OS Awareness Manual Cmicro

OS Awareness Manual Cmicro

Version 06-Jun-2024

Overview



The OS Awareness for Cmicro contains special extensions to the TRACE32 Debugger. This chapter describes the additional features, such as additional commands and statistic evaluations.

Brief Overview of Documents for New Users

Architecture-independent information:

- **"Training Basic Debugging"** (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.

Supported Versions

Currently SDT-Cmicro is supported for the following versions:

- SDT 3.1 (Cmicro V2.1) on 68k and MELPS7700
- Tau 3.2 (Cmicro V2.2) on 68k and C167

Configuration

The **TASK.CONFIG** command loads an extension definition file called “cmicro.t32” (directory “~~/demo/*<processor>*/kernel/cmicro”). It contains all necessary extensions.

Automatic configuration tries to locate the Cmicro internals automatically. For this purpose all symbol tables must be loaded and accessible at any time the OS Awareness is used.

If a system symbol is not available or if another address should be used for a specific system variable then the corresponding argument must be set manually with the appropriate address. In this case, use the manual configuration, which can require some additional arguments.

If you want do use the display functions “On The Fly”, i.e. displaying the OS objects, while the target is running, you need to have access to memory while running. In case of ICD, you have to enable **SYStem.MemAccess** or **SYStem.CpuAccess** (CPU dependent).

Manual Configuration

Format: **TASK.CONFIG cmicro** *<magic_address> <args>*

<magic_address> Specifies a memory location that contains the current running task. This address can be found at at the label “`xRunPID`”.

<args> The configuration requires two additional argument, that specify Cmicro internal tables. Give the labels “`xPDTBL`” and “`xmk_Queue`”.

This command configures the TRACE32 OS Awareness for Cmicro with manual setup.

The **TASK.CONFIG** command loads an extension definition file called “cmicro.t32” (directory “~~/demo/*<processor>*/kernel/cmicro”). It contains all necessary extensions.

Example:

```
; manual configuration for SDT-Cmicro support
 task.config cmicro xRunPID xPDTBL xmk_Queue
```

If you want to have dual port access for the display functions (display ’On The Fly’), you have to map emulation memory to the address space of all used system tables.

See also the example “~~/demo/*<processor>*/kernel/cmicro/cmicro.cmm”

Format:	**TASK.CONFIG cmicro**

This command configures the TRACE32 OS Awareness for Cmicro with automatic setup.

The **TASK.CONFIG** command loads an extension definition file called “cmicro.t32” (directory “~~/demo/<*processor*>/kernel/cmicro”). It contains all necessary extensions.

This configuration tries to locate the Cmicro internals automatically. For this purpose the symbols ’xRunPID’, ’xPDTBL’ and ’xmk_Queue’ must be loaded and accessible at any time, the OS Awareness is used.

Each **TASK.CONFIG** argument can be substituted by ’0’, which means, that this argument will be searched and configurated automatically. For a full automatic configuration omit all arguments to the command:

```
; full automatic configuration for SDT-Cmicro support
 task.config cmicro
```

If a system symbol is not available, or if another address should be used for a specific system variable, then the corresponding argument must be set manually with the appropriate address.

If you want to have dual port access for the display functions (display ’On The Fly’), you have to map emulation memory to the address space of all used system tables.

See also the example “~~/demo/<*processor*>/kernel/cmicro/cmicro.cmm”

Hooks & Internals of SDT-Cmicro

To detect the current running process, the system variable “xRunPID” is used. The loaded process types and instances are accesses through “xPDTBL”, while all signals are accessed through “xmk_Queue”.

Features

The OS Awareness for Cmicro supports the following features.

Display of Kernel Resources

The extension defines new PRACTICE commands to display various kernel resources. The following information can be displayed:

- process types (TASK.PType)
- process instances (TASK.PList)
- signal queue (TASK.QList)

For a detailed description of each command refer to the chapter “SDT-Cmicro PRACTICE Commands”.

When working with emulation memory or shadow memory, these resources can be displayed “On The Fly”, i.e. while the target application is running, without any intrusion to the application. If using this dual port memory feature, be sure that emulation memory is mapped to all places, where Cmicro holds its tables.

When working only with target memory, the information will only be displayed, if the target application is stopped.

Process Runtime Statistics

The time spent in a process can be evaluated statistically and displayed graphically. To do this, a memory location, holding the current running process, must be recorded (that means, recording all process switches).

To do a selective recording on process switches, the following PRACTICE commands can be used:

```
; Mark the magic location with an Alpha breakpoint
Break.Set task.config(magic)++(task.config(magicsize)-1) /Alpha
; Program the Analyzer to record only process switches
Analyzer.ReProgram
(
     Sample.Enable if AlphaBreak&&Write
)
```

To evaluate the contents of the trace buffer, use these commands:

Analyzer.List List.TASK DEFault	Display trace buffer and process switches
Analyzer.STATistic.TASK	Display process runtime statistic evaluation
Analyzer.Chart.TASK	Display process runtime time chart

The kernel activities up to the process switch are added to the calling process. The start of the recording time, when the calculation doesn't know, which process is running, is calculated as "(root)".

Function Runtime Statistics

All function related statistic and time chart evaluations can be used with process specific information. The function timings will be calculated dependent on the process, that called this funciton. To do this, additionally to the function entries and exits, the process switches must be recorded.

To do a selective recording on process related function runtimes, the following PRACTICE commands can be used:

```
; Mark the magic location with an Alpha breakpointBreak.Set
task.config(magic)++(task.config(magicsize)-1) /Alpha
; Mark the function entries/exits with Alpha/Beta breakpoints
Break.SetFunc
; Program the Analyzer to record function entries/exits and task switches
Analyzer.ReProgram
(
     Sample.Enable if AlphaBreak||BetaBreak
     Mark.A        if AlphaBreak
     Mark.B        if BetaBreak
)
```

To evaluate the contents of the trace buffer, use these commands:

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

The kernel activities up to the process switch are added to the calling process. The start of the recording time, when the calculation doesn't know, which process is running, is calculated as "(root)".

SDT-Cmicro specific Menu

The file “cmicro.men” contains an alternate menu with Cmicro specific topics. Load this menu with the MENU.ReProgram command.

You will find a new pull down menu called “Cmicro”.

The “Process Types”, “Process List” and “Queue List” topics launch the kernel resource display windows.

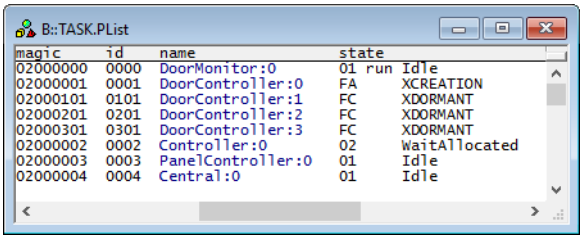
The Analyzer->List pull-down menu is changed. You can additionally choose for an analyzer list window showing only process switches (if any) or process switches and defaults.

The “Perf” menu contains an additional submenu for process runtime statistics.

TASK.PList — Display process instances

Format: **TASK.PList**

Displays a table with all created process instances.

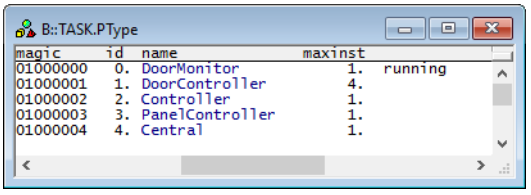


The magic number is a unique ID to the OS Awareness to specify a specific process. The process instance name is only available, if the according process type name, or the process instance name itself is set via the TASK.NAME.SET command. “run” specifies the process instance, which is currently running. The “X…” states are system states. All other states are user states. To display the state name, it must be set with the TASK.NAME.SET command.

TASK.PType — Display process types

Format: **TASK.PType**

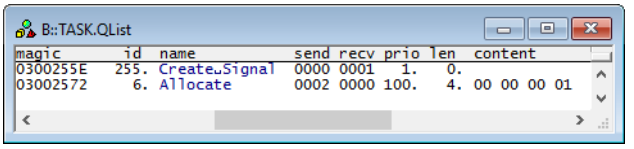
Displays a table with all defined process types.



The magic number is a unique ID to the OS Awareness to specify a specific process. The process type name is only available, if set via the TASK.NAME.SET command.
“running” specifies the process type, which is currently active.

Format: **TASK.QList**

Displays a table with all signals currently queued.



The magic number is a unique ID to the OS Awareness to specify a specific signal.
The signal name is only available, if set via the TASK.NAME.SET command.
“send” and “recv” are the sending rsp. receiving process ID (if available).

There are special definitions for Cmicro specific PRACTICE functions.

TASK.CONFIG() OS Awareness configuration information

Syntax: **TASK.CONFIG(magic | magicsize)**

Parameter and Description:

magic	**Parameter Type**: String (***without*** quotation marks). Returns the magic address, which is the location that contains the currently running task (i.e. its task magic number).
magicsize	**Parameter Type**: String (***without*** quotation marks). Returns the size of the magic number (1, 2 or 4).

Return Value Type: Hex value.