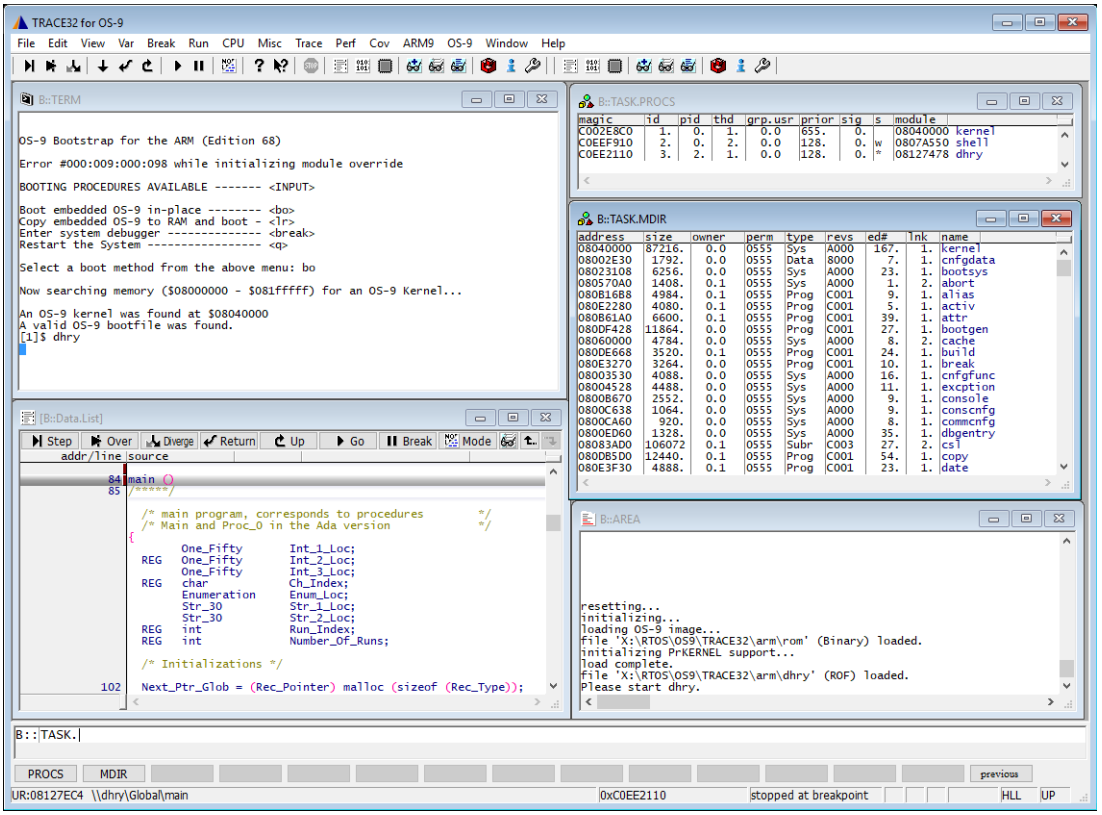
OS Awareness Manual OS-9

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>*”.

Overview



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.

Supported Versions

Currently OS-9 is supported for the following versions:

- OS-9 for ARM version 4.1
- OS-9 for PowerPC version 1.4
- OS-9 for 68K version 1.2

The PRACTICE script '~~/demo/m68k/kernel/os9/pos9.cmm' patches the kernel and configures the OS Awareness. The macros defined at the beginning of the file define the address of the kernel, the address of the global system variables and the vectors which are used to enter the kernel (e.g. clock interrupt). These values have to been checked and modified if necessary. The emulation memory has to mapped into the address space used by OS-9 to access the information by dual-port memory.

Format: **TASK.CONFIG os9** *<magic_address> <sleep> <globals> <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”.
<globals>	This argument must be the address of the system-global variables, which is used to display the tables.
<system_call_gate>	This argument is the address of the system call entry point, which is used by the command when executing system calls.

If the task selective debugging features are not used, the patching of the kernel is not required. The first two arguments are then not required. The following PRACTICE script will configure the command for data table display:

```
&globals=0cxxxx

TASK.CONFIG os9 0x0 0x0 &globals 0x0
```

TIP: The command **SETUP.DIS** can be used to display the OS-9 traps correctly in the disassembler windows.

The PRACTICE script '~~/demo/m68k/kernel/os9/pos9.cmm' can make the required patches to OS/9 and configure the display command:

```
DO pos9 nopatch                ; configures only display functions
                               ; no patches are made (TASK.OFF)

DO pos9 notask                 ; enables display and analyzer
                               ; functions task selective debugging
                               ; is off

DO pos9                        ; patches VRTX32 for task selective
                               ; debugging
```

The PRACTICE file must be modified according the software running on the target. The address of the system globals the memory for patching must be defined.

Hooks in OS-9

When the task selective debugging is used the entry and exit of the kernel must lead to a multitask breakpoint. To determine the entry of a task, patching the OS-9 kernel 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 process table address to the magic word of the OS Awareness and runs to a breakpoint.

The entries to OS-9 are patched directly in the vector table. The patches write the value 1 to the magic word and run to a breakpoint.

The correct setting of the breakpoints can be checked as follows:

1. Boot OS-9
2. Break program with 'break'
3. Execute 'patchos9.cmm'
4. Continue with 'go'
5. Disable debugging for the kernel
6. Set 'write'-breakpoints to the os-9 global-variables
7. The program should continue without breaking or spotting

Display of Kernel Resources

The extension defines new commands to display various kernel resources. Information on the following OS9 components can be displayed:

TASK.SYSGLOB	Globals and time
TASK.PROCS	Processes
TASK.PROCSL	Processes (extended)
TASK.QUEUES	Event table
TASK.EVENTS	Porcess queues
TASK.ALARMS	Alarm table
TASK.MDIR	Module directory
TASK.MFREE	Free memory
TASK.DEVS	Devices
TASK.IRQS	IRQ table

Symbol Relocation

The processes of OS9 use position independent code and data. The symbols need to be relocated, when they get a new address from OS9. The OS9 awareness provides a method to relocate the symbols automatically when necessary. This method extracts a table from the target memory which defines the locatio nof the position independent code and data sections. It will be called automatically after any program break, or by manually execute the command **sYmbol.RELOCate.Auto**.

Programs that are not active can be relocated to an unused address, when the command **sYmbol.RELOCate.Passive** is active. Without this command symbols of currently not used programs will stay at the last known location. Correct relocation requires knowledge about the base address for position independent data used by the linker. This address must be defined by the **sYmbol.RELOCate.Base** command. The identification of programs can either be done by a unique number, defined by the **sYmbol.RELOCate.Magic** command or based on an address in the code area of a program.

See the sYmbol.RELOCate commands in the **OS-9 Command** section

Example for automatic relocation of symbols for OS/9 (partial program):

```
TASK.CONFIG os9 0x0 0x0 0x1000

sYmbol.RELOCate.Passive 0x0ffff0000
sYmbol.RELOCate.Base 0x0ffff0000
sYmbol.RELOCate.Auto ON

Data.LOAD.ROF modul1 0x18000 0x0ffff8000
```

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 byte of all tasks. The breakpoints to the tcb status bytes are set by the **TASK.TASKState** command. The example script 'taskstat.cmm' makes a task state analysis with the demo application. NOTE: The analysis will only show task which were existent when the file is executed and after the measurement has completed.

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

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 Selective Debugging

Task selective debugging allows to disable or enable the analyzer and the trigger system for specific tasks and to stop one task while others continue to operate. This function has an impact on the response time of the multitask kernel. The feature should not be used when making performance or time measurements or with extremely time critical applications. Task selective debugging is currently not available on CPU32 and CPU32+ processors.

System Calls

Manually executing system calls requires a small program on the target, which makes the system call and stops execution after the call. Such a program is part of the standard patch procedure (pos9.cmm). The memory at the system parameter buffer (a part of the patch area) must be mapped internal.

sYmbol.RELOCate.Auto — Control automatic relocation

Format: **sYmbol.RELOCate.Auto** [**ON** | **OFF**]

Enables or disables the automatic relocation process. Without argument the command forces an immediate relocation base on the current values of the target. This manual triggered relocation is useful when the target can not be stopped, but analyzer or breakpoint features will be used. It can also be useful when the read of the relocation information structure of the target is time consuming and should not be performed after each breakpoint or step.

```
sYmbol.RELOCate.Auto           ; perform one single relocation

sYmbol.RELOCate.Auto ON        ; turn automatic relocation on
```

See chapter **Symbol Relocation**.

sYmbol.RELOCate.Base — Define base address

Format: **sYmbol.RELOCate.Base** *<class>*:*<base>*] [*<symbol_path>*|*<range>*]

Defines the current base address for one or more programs. The symbol path limits the definition to special symbols of a module or a program. If an address range is given, only the symbols in this range will be set. The memory class **P:** or **D:** defines which base address (program or data) is set. This program doesn't relocate symbols. The command is used after loading the symbols, when the default base address in the table doesn't match. The default program base is the first location of the program, the database is zero.

```
sYmbol.RELOCate.Base d:0x400000       ; assume all position independent
                                      ; data was linked to address 400000
```

See chapter **Symbol Relocation**.

sYmbol.RELOCate.List — List relocation info

Format: **sYmbol.RELOCate.List**

Displays information about the automatic relocation of symbols.

The magic column displays is the identifier of a program, zero means that the program is identified by an address inside the code area. The 'prog' and 'data' columns show the current base address for code and data. The 'active' field is set when the program is currently alive.

See chapter **Symbol Relocation**.

sYmbol.RELOCate.Magic — Define program magic number

Format: **sYmbol.RELOCate.Magic** *<program_magic>* [*<symbol_path>*|*<range>*]

Defines the program magic number for one or more programs. The symbol path limits the definition to special symbols of a module or a program. If an address range is given, only the symbols in this range will be set. The magic number can be used to identify a program and get a relation between task numbers in the target and program names. A magic number of zero (default) will use the program address as an identifier.

```
sYmbol.RELOCate.Magic 0x665f0 \\MODUL1     ; assignes the magic number
                                           ; 665f0 to the program MODUL1
```

See chapter **Symbol Relocation**.

sYmbol.RELOCate.Passive — Define passive base address

Format: **sYmbol.RELOCate.Passive** *<class>*:*<base>*

When a program is currently not used in the target, the code or data symbols are relocate to the address defined by this command. The memory class **P:** or **D:** defines which base address (program or data) is set. A base address of zero (default) turns the relocation off. In this case the symbols of not used (passive) programs stay where they are.

```
sYmbol.RELOCate.Passive d:0x0ffff0000      ; unused data symbols will be
                                           ; relocated to address
                                           ; 0ffff0000
```

See chapter **Symbol Relocation**.

TASK.SYSGLOB Display time

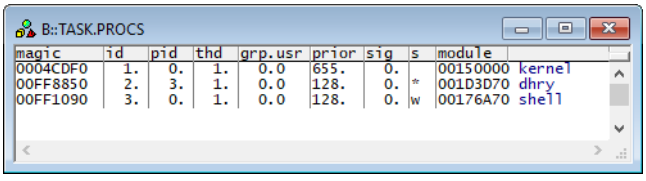
Format: **TASK.SYSGLOB**

Displays the current time and tick.

TASK.PROCS Process table

Format: **TASK.PROCS**

Displays the process table.



TASK.PROCSL Extended process table

Format: **TASK.PROCSL**

Displays the process table in an extended format.

TASK.QUEUES Process queues

Format: **TASK.QUEUES**

TASK.EVENTS Event table

Format: **TASK.EVENTS**

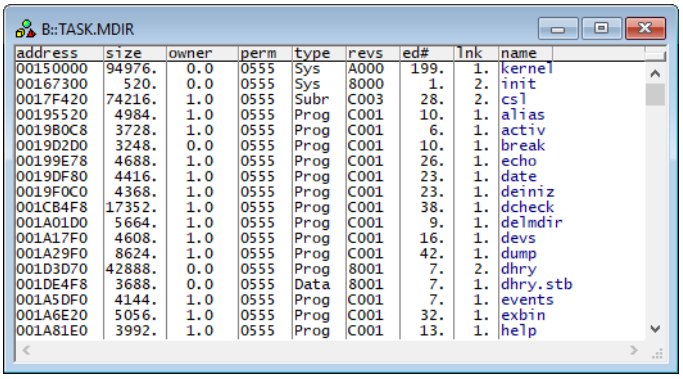
TASK.ALARMS Alarm table

Format: **TASK.ALARMS**

TASK.MDIR

Module table

Format: **TASK.MDIR**



B::TASK.MDIR

address	size	owner	perm	type	revs	ed#	lnk	name
00150000	94976.	0.0	0555	Sys	A000	199.	1.	kernel
00167300	520.	0.0	0555	Sys	8000	1.	2.	init
0017F420	74216.	1.0	0555	Subr	C003	28.	2.	csl
00195520	4984.	1.0	0555	Prog	C001	10.	1.	alias
0019B0C8	3728.	1.0	0555	Prog	C001	6.	1.	activ
0019D2D0	3248.	0.0	0555	Prog	C001	10.	1.	break
00199E78	4688.	1.0	0555	Prog	C001	26.	1.	echo
0019DF80	4416.	1.0	0555	Prog	C001	23.	1.	date
0019F0C0	4368.	1.0	0555	Prog	C001	23.	1.	deiniz
001CB4F8	17352.	1.0	0555	Prog	C001	38.	1.	dcheck
001A01D0	5664.	1.0	0555	Prog	C001	9.	1.	delmdir
001A17F0	4608.	1.0	0555	Prog	C001	16.	1.	devs
001A29F0	8624.	1.0	0555	Prog	C001	42.	1.	dump
001D3D70	42888.	0.0	0555	Prog	8001	7.	2.	dhry
001DE4F8	3688.	0.0	0555	Data	8001	7.	1.	dhry.stb
001A5DF0	4144.	1.0	0555	Prog	C001	7.	1.	events
001A6E20	5056.	1.0	0555	Prog	C001	32.	1.	exbin
001A81E0	3992.	1.0	0555	Prog	C001	13.	1.	help

TASK.MFREE

Free memory

Format: **TASK.MFREE**

TASK.DEVS

Device table

Format: **TASK.DEVS**

TASK.IRQS

Interrupt polling table

Format: **TASK.IRQS**

Format: **TASK.CCTL** <*option_word*>

Calls the F$Cttl function. The meaning of the bits induced the *option-word* is described in the OS-9 manual.

TASK.EXIT Exit system call

Format: **TASK.EXIT**

The current process is terminated by an OS-9 F$Exit call.

TASK.SEND Send signal

Format: **TASK.SEND** [<*id*>] [<*signal*>]

Sends a signal to one specific or all processes (F$Send).

```
task.send 5. 0.                ; kill task 5.
```

TASK.SysCall Generic system call

Format: **TASK.SysCall** <*code*> [<*d0*> … <*d4*> <*a0*> <*a1*> <*a2*>]

Executes any OS-9 system call. If the system call hangs the kernel, you can try to break manually and use the command **Register.SWAP** to restore the CPU registers in front of the system call. After the system call has been executed the system register set contains the values returned by the system call. The values of

D0..D4 and A0..A2 are displayed in the message line. This command if the most dangerous of the OS Awareness, as wrong arguments may cause the kernel to crash down. Use this command if it's really necessary only.

TASK.MDIR.ADDRESS() Program base address from module directory

Syntax: **TASK.MDIR.ADDRESS(**<*module_name*>**)**

Extracts the base address of a process from the module directory.

Parameter Type: String (*with* quotation marks).

Return Value Type: Hex value.

Frequently-Asked Questions

No information available