

OS Awareness Manual pSOS+

Release 02.2025



RACE32 Online Help	
RACE32 Directory	
RACE32 Index	
RACE32 Documents	Þ
OS Awareness Manuals	
OS Awareness Manual pSOS+	1
History	4
Overview	4
Brief Overview of Documents for New Users	4
Supported Versions	5
Configuration	6
Manual Configuration	6
Automatic Configuration	7
Quick Configuration Guide	8
Hooks & Internals in pSOS+	8
Features	9
Display of Kernel Resources	9
TRACE32 Board Support Package with pROBE+ Terminal Emulation	10
Task Runtime Statistics	10
Task State Analysis	10
Function Runtime Statistics	11
System Calls	11
Task Selective Debugging	11
pSOS specific Menu	12
pSOS Commands for i386, M68k and PPC	13
TASK.QC Configuration	13
TASK.QD Date and time	14
TASK.QO Objects	15
TASK.QP Partitions	15
TASK.QQ Queues	16
TASK.QR Regions	17
TASK.QS Semaphores	18
TASK.QT Tasks	19
TASK.QV Version	19
TASK.SC System calls	20
pSOSx86 Commands	21

TASK.QC	Configuration	21
TASK.QP	Process table	22
TASK.QT	Time	22
TASK.TASKState	Mark task state words	22
pSOS PRACTICE Functions		24
TASK.CONFIG()	OS Awareness configuration information	24

Version 13-Feb-2025

History

04-Feb-21 Removing legacy command TASK.TASKState.

Overview

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

Brief Overview of Documents for New Users

Architecture-independent information:

- **"Debugger Tutorial**" (debugger_tutorial.pdf): Get familiar with the basic features of a TRACE32 debugger.
- "General Commands" (general_ref_<x>.pdf): Alphabetic list of debug commands.
- "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.

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.

Currently pSOS is supported for the following versions:

- pSOS 1.1A with pROBE 1.0A on Intel x86, real mode
- pSOS 1.1.x, 1.2.x, 2.0.E, 2.1.x on M68k
- pSOS 2.0.7 with pROBE 3.0.8 on PowerPC
- pSOS 2.1.x on ARM7
- pSOS 2.2.3 on PPC
- pSOS 2.2.6 on Intel 386, protected mode.
- pSOS 2.3.0 on M68K,
- pSOS 2.5.x on PPC

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

Automatic configuration tries to locate the pSOS 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. This can be done by manual configuration which can require some additional arguments, too.

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.

Manual Configuration

Manual configuration for the OS Awarenesss for pSOS+ is only necessary for M68k controllers when using the kernel patch. The PRACTICE file 'ppsos.cmm' patches pSOS+ and pROBE and configures the OS Awareness. The macros defined at the beginning of the file specify the address of pROBE+/68k, the address of the current-tcb pointer, and the vectors which are used to enter the kernel.

Format: TASK.CONFIG psos <magic_address> <sleep> <args></args></sleep></magic_address>					
<magic_address></magic_address>	Specifies a memory location that contains the current running task.				
<sleep></sleep>	The argument for <i><sleep></sleep></i> is currently not used. Specify '0'.				
<args></args>	The additional arguments of the TASK.CONFIG command must be the address of the system call routine, the node configuration table, the pSOS data structures and a flag indicating the main pSOS version.				

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 PRACTICE script ~~/demo/m68k/kernel/psos/<*version*>/ppsos.cmm can make the required patches to pSOS+ and configures the display commands:

DO ppsos nopatch	; configures only display functions ; no patches are made (TASK.OFF)
DO ppsos noprobe	; patches pSOS when pROBE is not loaded ; task selective debugging in on
DO ppsos notask noprobe	; patches pSOS when pROBE is not loaded ; task selective debugging in off
DO ppsos	; patched pSOS and pROBE for task selective ; debugging

The PRACTICE script file must be modified, when the pSOS node anchor is not at the default location. 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 'psos.cmm' in the ~~/demo/m68k/kernel/psos directory can be started with the same parameters. The application may require modifying the 'psos.cmm' file to load the currently used version of pSOS+.

Automatic Configuration

For system resource display and analyzer functionality, you can do an automatic configuration of the OS Awareness. For this purpose it is necessary that all system internal symbols are loaded and accessible. Each of the **TASK.CONFIG** arguments can be substituted by '0', which means that this argument will be searched and configured automatically. For a fully automatic configuration specify '0' to the magic and sleep arguments and omit all other arguments:

Format:

TASK.CONFIG psos 0 0

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.

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

- Run the demo script (~~/demo/m68k/kernel/psos/<version>/psos.cmm) with your kernel without any patching. This requires that you copy the files from this demo together with your version of the pSOS+ kernel in your directory. Start the demo with 'do psos nopatch'. Add the argument 'noprobe' if you don't want to load pROBE+. The result should be a list of tasks, which continuously change their state.
- 2. Run the demo script with patching. For CPU32(+) devices you need the argument 'notask'. (skip for x86 and PPC)
- Try single stepping, starting and stopping a task. Display kernel resources. On CPU32(+) devices the interrupts during single stepping can be suppressed by SETUP.IMASKASM, by a monitor extension (SYStem.MonFile) which stops the interrupt source in the chip or by freezing the timer.
- 4. Try the analyzer demo scripts (tasksc, taskstat and taskfunc).
- 5. Make a copy of the 'ppsos.cmm' PRACTICE file. Modify the file according to your application. This can be changing the pSOS+ node anchor or choosing a different memory area for the patches.
- 6. 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).
- 7. Run your application with patching, but without task selective debugging ('notask' argument). (skip for x86 and PPC)
- 8. Run your application with task selective debugging, when required (not CPU32(+) devices). (skip for x86 and PPC)

Hooks & Internals in pSOS+

Kernel patching for M68k:

To determine the entry of a task, the patching of pSOS+, and pROBE+ when used, 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 and runs to a

breakpoint. The entries to pSOS are patched directly in the vector table. The patches write the value 1 to the magic-word and run to a breakpoint. The 'breakpoint'-trap of pROBE+ should be patched too. This will ease the combination of pROBE+ breakpoints together with the state analyzer.

The task-delete hook of pSOS is used to detect if a task has been deleted. If this hook (KC_DELETECO) is already in use, an additional patch in required.

To stop a task and continue the kernel, the debugger uses the 'manual round robin' feature of PSOS. In this case the debugger will executed the function-call 'tm_wkafter(1)'.

No hooks are used for x86 and PPC.

The OS Awareness for pSOS+ supports the following features.

Display of Kernel Resources

The extension defines new commands to display various kernel resources. The commands can either give an overview about one resource type or display a single resource in detail. The resource can be defined by its ID, magic or name. The following information can be displayed:

The following information can be displayed for i386, M68k and PPC:

TASK.QC		Configuration	
TASK.QO		Objects	
TASK.QT		Tasks	
TASK.QQ		Queues	
TASK.QS		Semaphores	
TASK.QR		Regions	
TASK.QP		Partitions	
TASK.QD		Date and time	
TASK.QV	(only available on PPC)	Versions	
TASK.SC	(only available on M68k)	System calls	

For a detailed description of each command, refer to chapter "pSOS Commands for i386, M68k and PPC".

The following information can be displayed for x86:

TASK.QC	Configuration
TASK.QP	Process table
TASK.QT	Time

For a description of the commands, refer to chapter "pSOSx86 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 pSOS holds its tables.

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

TRACE32 Board Support Package with pROBE+ Terminal Emulation

(only available for PPC) pSOS+ users can call for a TRACE32 board support package. This package is based on the SBC821 BSP. It allows to create applications that run on the in circuit emulator without any target.

The BSP contains a special console driver which connects the pROBE+ with a terminal emulation window of the emulator. The 'break' command pROBE, to stop a running application (default: 'CTRL-C') is changed to 'TAB'. The communication between pROBE+ and the terminal is done via two memory cells, requiring no external interface.

Our demo application was built with this TRACE32 BSP.

Task Runtime Statistics

The time spent in a task can be analyzed by marking the accesses to a word holding the current task descriptor. All kernel activities are added to the calling task. The example PRACTICE script 'taskfunc.cmm' can be used to make the measurement for this analysis.

Analyzer.List List.TASK DEFault Analyzer.STATistic.TASK

Analyzer.Chart.TASK

Display trace buffer and task switches Display task runtime statistic evaluation 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 example script 'taskstat.cmm' makes a task state analysis with the demo application.

Analyzer.STATistic.TASKState Analyzer.Chart.TASKState Display task state statistic evaluation Display task state time chart 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.List List.TASK FUNC
Analyzer.STATistic.TASKFunc
Analyzer.STATistic.TASKTREE
Analyzer.Chart.TASKFunc

Display function nesting Display function runtime statistic Display functions as call tree Display function time chart

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 (ppsos.cmm). The memory at the system parameter buffer (a part of the patch area) must be mapped internal (only available for M68k).

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 not available on x86, PPC, CPU32 and CPU32+ processors.

The menu file "psos.men" contains a menu with pSOS specific menu items. Load this menu with the **MENU.ReProgram** command.

You will find a new menu called **pSOS+**.

- The **pROBE Terminal** menu item brings up a terminal emulation window, which communicates with the preconfigured pROBE+ debugger.
- The **Break to pROBE** menu item performs a special break command inside the terminal emulation to gain control to pROBE (see TRACE32 BSP).
- The Query menu items launch the kernel resource display windows.

In addition, the menu file (*.men) modifies these menus on the TRACE32 main menu bar:

- The **Trace** -> **List** submenu is changed. You can additionally choose if you want a trace list window to show only task switches (if any) or task switches and defaults.
- The **Perf** menu contains the additional submenus for task runtime statistics, task-related function runtime statistics and statistics on task states. For the function runtime statistics, a PRACTICE script file called "men_ptfp.cmm" is used. This script file must be adapted to your application.

TASK.QC

Format:

TASK.QC [NODE | PSOS | PROBE]

Displays the configuration tables of pSOS+. Some of the fields are mouse sensitive. Double clicking on them will show the appropriate information.

🖧 B::TASK.QC		- • •
<pre>node configuration table node anchor = 00003044</pre>		
table addr = 00167000		
NC_CPUTYPE = 00000205 NC_PROBECT = 001670CC NC_RFU = 00000000 NC_PMONT = 00000000	NC_MPCT = 00000000 NC_PHILECT = 00000000 NC_PNACT = 0016713C	NC_PSOSCT = 001671F4 NC_PREPCT = 001671B4 NC_PSECT = 00000000
<		ير <
R::TASK.OC PSOS		
B::TASK.QC PSOS		
B::TASK.QC PSOS pSOS+ configuration table table addr = 001671F4		
■ B:TASK.QC PSOS pSO5+ configuration table table addr = 001671F4 KC_PSOSCODE = 00064F10 KC_RNOUSIZE = 00001000 KC_NSEMA4 = 00000032 KC_NLOCOBJ = 00000020 KC_ROTSADR = 00000004 KC_ROTSADR = 00002000 KC_SWITCHCO = 00000000 KC_RTCINIT = 00000000	KC_RNDSADR = 001A9480 KC_NTASK = 00000032 KC_NMGGBUF = 00000064 KC_TICKS2SEC = 00000064 KC_RO0TSSTK = 00001000 KC_STATCO = 0000000 KC_FATAL = 0000000 KC_RESERVED = 0000400	KC_RNDLEN = 00656840 KC_NQUEUE = 00000028 KC_NTIMER = 00000010 KC_TSLICE = 00000000 KC_R00TUSTK = 0000000 KC_DELETECO = 0000000 KC_DLECO = 0000000 KC_R00TPRI = 0000000

TASK.QD

Format:

TASK.QD

Displays the current time and tick.

The 'ilevel' field displays the level of nested interrupts. This value should always be positive and near to zero.

🖧 B::TASK.QD		- • ×
date	time	ticks
1. MAY 1995	8:30: 0	00000001
		~
<		>
P.		ii

TASK.QO

Objects

Format:

TASK.QO [<id> | <name>]

Displays the object-table of pSOS+.

With arguments it displays one object in detail.

🖧 B::TASK.	QO			×
magic	name	id	type	
001AACD0	RN#0	00000000	REGION	~
001AACF0	IDLE	00010000	TASK	
001AAD10	ROOT	00020000	TASK	
001AAD30	PTSD	00030000	TASK-DATA	
001AAD50	pNA0	00040000	TASK-DATA	
001AAD70	TIPT	00050000	TASK	
001AAD90	TOPT	00060000	TASK	
001AADB0	DRS1	00070000	SEMAPHORE	
001AADD0	DRS2	00080000	SEMAPHORE	
001AADF0	DBS1	00090000	SEMAPHORE	
001AAE10	DBS2	000A0000	SEMAPHORE	
001AAE30	PCON	000B0000	TASK-DATA	
001AAE50	PNAD	000C0000	TASK	
001AAE70	RA00	000D0000	SEMAPHORE	
001AAE90	WA00	000E0000	SEMAPHORE	
001AAEB0	RQ00	000F0000	QUEUE	\sim
<				>
1				

TASK.QP

Partitions

Format:

TASK.QP [<partition_id> | <partition_name>]

Displays the partition table of pSOS+.

With arguments it displays one partition in detail.

🔒 B::TASK.QP 📃 🗖 💌						x		
magic	name	ptid	buff siz	#buffers	#free	do?	address	
00151004	PTN1	00170000	00000200	00000004	00000000	NO	00151004	~
								\mathbf{v}
<							>	

Format:

TASK.QQ [<queue_id> | <queue_name>]

Displays the queue table of pSOS+.

With arguments it displays one queue in detail.

	🖧 B::TASK.QQ 001ADEC8 📃 🗉 🕰								x
	magic	name	qid	tq len	mq len	mq limit	mgb	qtype var	
	UUTADECO	RQUU	000F0000	0000000	00000000	00000010	Sys-poor	FIFU NO	^
	task queu	e: nam	e qid						-
		EMP	TY						
	message q	ueue:	0		4	8	С		
ľ									- ×
ļ	<								>

Format:

TASK.QR [<region_id> | <region_name>]

Displays the region table of pSOS+.

With arguments it displays one region in detail.

🖧 B::TASK.	QR								-	-	×
magic	name	rnid	address	length	unitsize	free	largest	tq len	do?	qtype	
001E01F4 007E5000	RN#O RMEM	00000000 00190000	001A9480 007E5000	00656B40 00000800	00001000 00000080	005ED000 00000700	005ED000 00000700	00000000	NO NO	FIFO FIFO	^
											~
<										>	

🖧 E	B::TASK.	QR 001E	01F4							-	•	×
magi 001	c 01F4	name RN#0	rnid 00000000	address 001A9480	1ength 00656840	unitsize 00001000	free 005ED000	largest 005ED000	tq len 00000000	do? NO	qtype FIF0	,
tasl	queu	e: nam	e qid									
		EMP.	TY	1								
Unit	t Usag	e: sta 001	E01F4 (00002E0C	Header							-
<											>	.d

TASK.QS

Format:

TASK.QS [<semaphore_id> | <semaphore_name>]

Displays the semaphore table of pSOS+.

With arguments it displays one semaphore in detail.

🖧 B::TASK.	QS				,	2	3
magic	name	smid	count	tq len	qtype		
001AF4F8	DRS1	00070000	00000001	00000000	FIFO	_	~
001AF528	DRS2	00080000	00000000	00000000	FIFO		
001AF558	DBS1	00090000	00000001	00000000	FIF0		
001AF588	DBS2	000A0000	00000000	00000000	FIFO		
001AF5B8	RA00	000D0000	00000001	00000000	FIFO		
001AF5E8	WA00	000E0000	00000001	00000000	FIFO		
001AF618	TC00	00110000	00000000	00000000	FIFO		
001AF648	CC00	00120000	00000000	00000000	FIF0		
001AF678	RELW	00140000	00000001	00000000	FIFO		
001AF6A8	RAMD	00150000	00000001	00000000	FIFO		
							¥
<						>	

🖧 B::TASK.QS "RELW"	
magic name smid count tqlen qty	oe i
001AF678 RELW 00140000 00000001 00000000 FIF0	~ ~
task queue: name qid	
EMPTY	
	~
<	>

Format:

TASK.QT [<task_id> | <task_name>]

Displays the task-table of pSOS+.

With arguments it displays one task in detail

🖧 B::TASK.	QT								×
magic	name	id	prio	mode	status	susp	parameters	ticks	
001DD5E8	IDLE	00010000	00	2000	Ready				^
001DD870	ROOT	00020000	E6	2000	Ready	YES			
001DD6C0	TIPT	00050000	FA	2001	Cpwait			forever	
001DD798	TOPT	00060000	FA	2001	Ready	YES			
001DD948	PNAD	000C0000	FF	2000	Evwait		EVENTS = C0000000	forever	
001DDA20	MEM1	001A0000	30	0000	Running				
001DDAF8	MEM2	001B0000	2F	0000	Ready				
001DDBD0	I01_	001C0000	1E	0002	Ready				
001DDCA8	I02	001D0000	1E	0002	Ready				
001DDD80	SRCE	001E0000	80	2000	Ready	YES			
001DDE58	SINK	001F0000	50	2000	Wkafter			00000001	
001DDF30	SUDO	00200000	81	0002	Wkafter			00000167	
									Υ.
<								>	

B::TASK.QT "ROOT" □	ĸ
magic name id prio mode status susp parameters ticks	
001DD870 ROOT 00020000 E6 2000 Ready YES	\wedge
Initial PC = 000D0004 Initial Pri = E6	
Initial SSP = 007ECF18 SStack Size = 00002F18	
Initial USP = 007ECF18 UStack Size = 00000000	
Timers:	
NONE	
<u>, , , , , , , , , , , , , , , , , , , </u>	

TASK.QV

Version

Format: TASK.QV

Displays the pSOS+ and pROBE+ versions.

🖧 B::TASK.QV	×
component versions pSOS+/PPC_V3.0.0.7	,
pROBE+/PPCuPSuV4.0.0.7	
pREPC+/PPCuV3.0.0.6	
pNA+/PPCuV4.2.0.9	
< >	

Format:	TASK.SysCall < <i>function</i> > < <i>d</i> 1> < <i>d</i> 2> < <i>d</i> 3> < <i>d</i> 4> < <i>d</i> 5> < <i>a</i> 0> < <i>a</i> 1>
<function>:</function>	PT_SGETBU Q_BROADCA EV_RECEIV TM_WKAFTE TM_EVEVER Q_VRECEIV Q_VBROADC Q_AVURGEN <function></function>

Executes a pSOS system call.

The function can only be executed, when the currently selected task is already stopped or can be stopped by the OS Awareness. When the task selective debugging is not active, the emulation must be stopped (in a regular task) before executing the command. Some functions are abbreviated to nine characters (see above list).

task.sc q_create 41424344 4 0 0 7; create new queuetask.sc q_send 0c0000 12 34 56 78; send message to queue

TASK.QC

Configuration

Format:

TASK.QC [SYSTEM | PSOS | PROBE]

Displays the configuration tables of pSOS+.





TASK.QP

Format: TASK.QP

Displays the process table of pSOS-86.

The state 'Running' is not displayed.

magic name ptid buff siz #buffers #free do? address 00151004 PTN1 00170000 00000200 00000004 00000000 N0 00151004	
00151004 PTN1 00170000 00000200 00000004 00000000 NO 00151004	
	~
	¥
< >>	

TASK.QT

Format:

Time

TASK.QT

Displays the current time and the nesting levels of system calls and interrupts.

🖧 B::TASK.	QT								×
magic	name	id	prio	mode	status	susp	parameters	ticks	
001DD5E8	IDLE	00010000	00	2000	Ready				~
001DD870	ROOT	00020000	E6	2000	Ready	YES			
001DD6C0	TIPT	00050000	FA	2001	Cpwait			forever	
001DD798	TOPT	00060000	FA	2001	Ready	YES			
001DD948	PNAD	000C0000	FF	2000	Evwait		EVENTS = C0000000	forever	
001DDA20	MEM1	001A0000	30	0000	Running				
001DDAF8	MEM2	001B0000	2F	0000	Ready				
001DDBD0	I01_	001C0000	1E	0002	Ready				
001DDCA8	I02_	001D0000	1E	0002	Ready				
001DDD80	SRCE	001E0000	80	2000	Ready	YES			
001DDE58	SINK	001F0000	50	2000	Wkafter			00000001	
001DDF30	SUDO	00200000	81	0002	Wkafter			00000167	
									\sim
<								>	
,									

TASK.TASKState

Mark task state words

Format:

TASK.TASKState

This command sets Alpha breakpoints on all task status words.

The statistic evaluation of task states (see **Task State Analysis**) requires recording of the accesses to the task state words. By setting Alpha breakpoints to these words and selectively recording Alpha's, you can do a selective recording of task state transitions.

Because setting the Alpha breakpoints by hand is very hard to do, this utility command automatically sets the Alpha's to the status words of all tasks currently created. It does NOT set breakpoints to tasks that terminated or haven't yet been created.

There are special definitions for pSOS specific PRACTICE functions.

TASK.CONFIG() OS Awareness configuration information

Syntax:	TASK.CONFIG(magic magicsize)	
---------	--------------------------------	--

Parameter and Description:

magic	Parameter Type : String (<i>without</i> 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 (<i>without</i> quotation marks). Returns the size of the task magic number (1, 2 or 4).

Return Value Type: Hex value.