LAUTERBACH
DEVELOPMENT TOOLS

Linux Debugging Reference Card

This reference card gives you an overview of frequently-used **TRACE32®** commands for debugging targets running Linux (stop mode).

Configure Linux for Debugging

Compile kernel with debug info	`CONFIG_DEBUG_INFO`
For module debugging set	`CONFIG_KALLSYMS`
Compile applications with option	`-g`

Attach to a Running Linux Target

```
;Initialize debugger
SYStem.CPU <cpu>
SYStem.Option <option>

;Enable space IDs
SYStem.Option MMUSPACES ON

;Attach to target
SYStem.Mode Attach
Break

;Load kernel symbols
Data.LOAD.Elf vmlinux /NoCODE
```

Continue with configuring Linux Awareness. Actual sequence heavily depends on target system.

Download and Run Linux with TRACE32

```
;Initialize debugger
SYStem.CPU <cpu>
SYStem.Option <option>

;Enable space IDs
SYStem.Option MMUSPACES ON

;Start at reset
SYStem.Up

;Let bootloader initialize the system
GO
WAIT 5.s
BREAK

;Load image to physical address
Data.LOAD.Binary uImage <phys.addr>

;Load kernel symbols
Data.LOAD.Elf vmlinux /NoCODE

;Continue bootloader
GO

;Start image with bootloader (in terminal window)
bootm <phys.addr>
```

Continue with configuring Linux Awareness. Actual sequence heavily depends on target system and loading mechanisms.

Configure Linux Awareness

MMU Declaration

```
MMU.FORMAT LINUX swapper_pg_dir \
    <kernel logical address range> \
    <physical start address>

TRANSlation.TableWalk ON
TRANSlation.ON
```

ARM based

```
TRANSlation.COMMON \
 (<kernel log. start addr>-0x01000000)--0xffffffff
```

Others

```
TRANSlation.COMMON \
  <kernel logical start address>--0xffffffff
```

Loading Linux-2.x Awareness

```
TASK.CONFIG \
~~/demo/<arch>/kernel/linux/linux-2.x/linux.t32

MENU.ReProgram \
~~/demo/<arch>/kernel/linux/linux-2.x/linux.men
```

Loading Linux-3.x Awareness

```
TASK.CONFIG \
~~/demo/<arch>/kernel/linux/linux-3.x/linux3.t32

MENU.ReProgram \
~~/demo/<arch>/kernel/linux/linux-3.x/linux.men
```

Defining Groups

```
GROUP.Create "kernel" \
    <kernel logical address range> /RED
```

Linux Resource Displays

Processes

```
;Linux menu ⟶ Display Processes
TASK.Process
```

Linux task list

```
;Linux menu ⟶ Display Tasks
TASK.DTask
```

Detailed task info

```
;Double-click on task “magic”
TASK.DTask "<name>"
```

“ps”

```
;Display ps-like
TASK.PS
```

File system internals

```
;Linux menu ⟶ Display File System
TASK.FS.*
```

Kernel log buffer

```
;Linux menu ⟶ Display Kernel Log
TASK.DMESG
```

Debugger task list

```
;Linux menu ⟶ Double-click on task field in status line
TASK.List
```

Debugging Processes

Display processes

```
;Linux menu ⟶ Display Processes
TASK.Process
```

Configure symbol loader

```
;Linux menu ⟶ Symbol Autoloader
TASK.sYmbol.Option AutoLoad Process
```

Start debugging at main()

```
;Linux menu ⟶ Process Debugging ⟶ Debug Process on main
DO app_debug <process>
```

The Watch process starts

```
;Linux menu ⟶ Process Debugging ⟶ Watch Processes
TASK.Watch
```

Attach to process

```
;Linux menu ⟶ Right click on "magic", Load Process Symbols
TASK.sYmbol.LOAD "<name>"
```

Show thread context

```
;Linux menu ⟶ Right click on "magic", Display Stack Frame
Frame /Task "<name>"
```

Thread specific breakpoints

```
;Linux menu ⟶  Select breakpoint, advanced, TASK
Break.Set <addr> /Task "<name>"
```

Trapping segmentation violation

```
DO segv ;(twice) – no menu
```

Debugging Libraries

Display libraries

```
;Linux menu ⟶ Display Processes, right-click on "magic",
;Display detailed, expand "code files"
TASK.DTask "<name>"
```

Configure symbol loader

```
;Linux menu ⟶ Symbol Autoloader
TASK.sYmbol.Option AutoLoad Library
```

Load symbols

```
;Linux menu ⟶ Right-click on library, load library symbols
TASK.sYmbol.LOADLib <proc> <lib>
```

Debugging Kernel Modules

Display modules

```
;Linux menu ⟶ Display Modules
TASK.MODule
```

Configure symbol loader

```
;Linux menu ⟶ Symbol Autoloader
TASK.sYmbol.Option AutoLoad Module
```

Start debugging at init routine

```
;Linux menu ⟶ Module Debugging ⟶ Debug Module on init
DO mod_debug <module>
```

Attach to module (load symbols)

```
;Linux menu ⟶ Right click on "magic", Load Module Symbols
TASK.sYmbol.LOADMod "<name>"
```

SMP Support

Setup	Selecting appropriate CPU automatically sets up debugger in SMP mode.
View	Debugger shows context of one core. Status line shows current core number. Data/Register windows change color.
Change core	Right-click to change CPU view `CORE` command
Thread core	**cpu** column in `TASK.DTASK` shows on which core the thread is running.
Breakpoints	Are set on all CPUs, current view switches automatically to core that hit the BP.

Dynamic Performance Measurement

Set PC/memory access method

```
;Perf menu ⟶ Perf Configuration: METHOD
PERF.METHOD
```

Show process/thread performance

```
;Perf menu ⟶ Perf Configuration: Mode ⟶ TASK and
;commands ⟶ ListTASK
PERF.MODE TASK
PERF.ListTASK
```

Show function performance

```
;Perf menu ⟶ Perf Configuration: Mode ⟶ PC, options ⟶
;MMUSPACES, command ⟶ ListFunc
PERF.MODE PC
PERF.MMUSPACES ON
PERF.ListFunc
```

Enable performance measurement

```
;Perf menu ⟶ Perf Configuration: state ⟶ Arm
PERF.Arm
```

Show details

```
;Perf menu ⟶ Perf Configuration: Double-click on function in
;PERF.ListFunc
List <function> /PERF
```

Trace-based Task Profiling

Setup with data trace

```
Break.Set TASK.CONFIG(magic) /Write /TraceEnable
```

Setup with context ID

```
ETM.ContextID 32
```

Thread switch patch

```
;See Appendix B in rtos_linux_stop.pdf
HELP.PDF ~~/pdf/rtos_linux_stop.pdf
```

Display task switches

```
Trace.List List.TASK /DEFault
```

Statistics

```
Trace.STATistic.TASK
```

Charts

```
Trace.Chart.TASK
Trace.ProfileChart.TASK
```

Trace-based Function Profiling

Setup with data trace

```
Break.Set TASK.CONFIG(magic) /Write /TraceData
```

Setup with context ID

```
ETM.ContextID 32
```

Thread switch patch

```
;See Appendix B in rtos_linux_stop.pdf
HELP.PDF ~~/pdf/rtos_linux_stop.pdf
```

Display trace with task switches

```
Trace.List List.TASK /DEFault
```

Statistics

```
Trace.STATistic.sYmbol
Trace.STATistic.Function
```

Charts

```
Trace.Chart.sYmbol
Trace.Chart.Function
Trace.ProfileChart.sYmbol
```

Code coverage

```
Trace.COVerage.ListModule
```

Documentation / Help / Support

User manuals	pdf/rtos_linux_stop.pdf pdf/rtos_linux_run.pdf
Training manual	pdf/training_rtos_linux.pdf pdf/training_rtos_linux_x86.pdf
Example scripts	demo/<cpu>/kernel/linux
Support	local sales offices, rtoslinux@lauterbach.com