

**Debugging
Solaris
using
Open Boot Prom**

Introduction

- **Basic Debugging**
 - basic OBP debugging commands
 - obpsym
 - introduction to forthdebug
- **Advanced Debugging**
 - introduction to forth
 - using forth with forthdebug

Basics

- **breakpoints**

add breakpoint *address +bp*

delete breakpoint *address -bp*

list breakpoints *.bp*

ok main +bp

ok trap +bp

ok idle 4 + +bp

ok .bp

10090528 1002d464 10042fa8

ok trap -bp

ok .bp

10090528 10042fa8

Basics

- **breakpoints (cont.)**

delete last breakpoint --bp

delete all breakpoints bpoff

```
ok main +bp
```

```
ok idle +bp
```

```
ok .bp
```

```
10090528 10042fa8
```

```
ok --bp
```

```
ok .bp
```

```
10090528
```

```
ok bpoff
```

```
ok .bp
```

Basics

- **breakpoints (cont.)**

continue from breakpoint go

continue *n* times *n* gos

ok trap +bp

ok go

1002d464 trap save %o6, ffffffff28, %o6

ok 4 gos

1002d464 trap save %o6, ffffffff28, %o6

1002d464 trap save %o6, ffffffff28, %o6

1002d464 trap save %o6, ffffffff28, %o6

1002d464 trap save %o6, ffffffff28, %o6

ok

Basics

- **breakpoints (cont.)**

single step

step

single step *n* times

n steps

```
1002d464 trap      save    %o6, ffffffff28, %o6
```

ok step

```
1002d468 trap+4   st      %i3, [%i6 + 50]
```

ok 3 steps

```
1002d46c trap+8   or      %g0, %i2, %l5
```

```
1002d470 trap+c   ld      [%g7 + a8], %i2
```

```
1002d474 trap+10  or      %g0, %i1, %l4
```

ok

Basics

- **breakpoints (cont.)**

continue until subroutine ends return

```
1002d468 trap+4      st      %i3, [%i6 + 50]
```

ok return

```
10007484 priv_rtt      rdpstate      %i3
```

same for a leaf subroutine returnl

```
1002d464 trap      save      %o6, ffffffff28, %o6
```

ok returnl

```
10007484 priv_rtt      rdpstate      %i3
```

Basics

- **display single register**

```
display register          reg .
```

```
ok %g7 .
```

```
6087a1c0
```

```
ok %i0 .
```

```
f005b2f8
```

Basics

- **set single register**

set register

n is reg

```
ok %o0 .
```

```
3028b5d8
```

```
ok 0 is %o0
```

```
ok %o0 .
```

```
0
```

Basics

- **displaying registers**

display C backtrace

ctrace

ok ctrace

PC: f0052878

Last leaf: jmpl f005b2f8 from 10007978 client_handler+38

0 w %o0-%o5: (10000000 16 f0000000 1042a9f8 10421a10 0)

call 10040878 p1275_sparc_cif_handler from 1003da5c
prom_enter_mon+34

1 w %o0-%o5: (f005b2f8 30027d88 1042a9f8 10420640 10421a10 0)

call 1003da28 prom_enter_mon from 10024d38 debug_enter+80

2 w %o0-%o5: (0 c 1042a9f8 1042a9f8 10421a10 0)

call 10024c90 abort_sequence_enter from 60546ae0 zs:zsa_xsint+26c

3 w %o0-%o5: (0 740000 d 63 4a4 c)

Basics

- displaying registers (cont.)

display global registers .registers

ok .registers

	Normal	Alternate	MMU	Vector
0:	0	0	0	0
1:	1003da5c	ffdf6c0	104800f0	1000
2:	f005b2f8	f0000000	104da480	c
...				
5:	0	f	ffde000	1fb0
6:	0	10006f84	8000000001f660b6	10404200
7:	3002bea0	17f	2	f0055cec

%PC f0052878 %nPC f005287c
%TBA 10000000 %CCR 88 XCC:Nzvc ICC:Nzvc

Basics

- displaying registers (cont.)

display local registers .locals

ok .locals

	INs	LOCALs	OUTs
0:	f005b2f8	1042b400	10000000
1:	30027d88	10404200	16
2:	1042a9f8	10409400	f0000000
3:	10420640	1	1042a9f8
4:	10421a10	1e	10421a10
5:	0	d	0
6:	30027d28	0	30027499
7:	1003da5c	0	10007978

Basics

- **displaying registers (cont.)**

display specified window *window# .window*

change current window *window# w*

ok ctrace

...

call 10024c90 abort_sequence_enter from 60546ae0 zs:zsa_xsint+26c

3 w %o0-%o5: (0 740000 10408800 10460968 0 c)

jmp 400 from 6054b2e4 zs:zs_high_intr+1e8

4 w %o0-%o5: (6054e000 80 6054e068 44 6054e068 ff0113)

jmp 0 from 6041c000 sbus:run_vec_poll_list+24

5 w %o0-%o5: (601b5880 6054b0fc 0 60536120 60 0)

jmp 104219a8 cpu0 from 6041c03c sbus:sbus_intr_wrapper+18

6 w %o0-%o5: (0 6024a680 1041bbec 0 0 60533e30)

ok 4 .window

	INs	LOCALs	OUTs
0:	601b5880	10000000	6054e000
1:	6054b0fc	400	80
2:	0	a	6054e068
3:	60536120	10421c40	44
4:	60	104219a8	6054e068

...

ok 4 w .locals

	INs	LOCALs	OUTs
0:	601b5880	10000000	6054e000
1:	6054b0fc	400	80
2:	0	a	6054e068
3:	60536120	10421c40	44
4:	60	104219a8	6054e068

...

Basics

- displaying v9 registers

```
display formatted %pstate      .pstate
```

```
ok .pstate
```

```
AG:0 IE:1 PRIV:1 AM:0 PEF:1 RED:0 MM:0 TLE:0 CLE:0 MG:0 IG:0
```

```
display trap registers        .trap-registers
```

```
ok .trap-registers
```

```
%TL:1 %TT:17f %TPC:f0052878 %TnPC:f005287c
```

```
%TSTATE:8800001404 %CWP:4
```

```
%PSTATE:14 AG:0 IE:0 PRIV:1 AM:0 PEF:1 RED:0 MM:0 TLE:0 CLE:0  
MG:0 IG:0
```

```
%ASI:0 %CCR:88 XCC:Nzvc ICC:Nzvc
```

Basics

- **displaying memory**

display memory

address len dump

```
ok %g7 20 dump
```

```
0 1 2 3 4 5 6 7 V 9 a b c d e f 01234567v9abcdef
```

```
606cb460 60 6c 76 20 c1 7c be cd 60 7d 5e 90 60 01 dd b0 `lv A|>M` }^.`.]0
```

```
606cb470 60 39 a0 78 00 00 00 00 00 00 00 00 00 00 00 00 `9 x.....
```

```
606cb480 60 1b 3b 98 00 00 00 00 02 00 00 00 00 00 00 00 `;.....
```

Basics

- **displaying variables**

display 8b word

address c?

display 16b word

address w?

display 32b word

address l?

display 64b word

address x?

ok freemem l?

f9

Basics

- **setting variables**

set 8b word *n address c!*

set 16b word *n address w!*

set 32b word *n address l!*

set 64b word *n address x!*

ok moddebug l?

0

ok 1 moddebug l!

ok moddebug l?

1

Basics

- **disassembling memory**

disassemble *address* dis

ok sfmmu_mp_startup dis

10022944 sfmmu_mp_startup save %o6, ffffffffafa0, %o6

10022948 sfmmu_mp_startup+4 call 10022850 sfmmu_set_tlb

1002294c sfmmu_mp_startup+8 sethi 0, %g0

continue disassembly +dis

ok +dis

10022950 sfmmu_mp_startup+c call 10008298 sfmmu_load_tsbstate

10022954 sfmmu_mp_startup+10 or %g0, 0, %o0

Basics

- **command line editing**

control-p	reprint last command
control-n	reprint next command
control-b	back one space
control-f	forward one space
control-a	beginning of line
control-e	end of line
control-h	erase last character
control-w	erase last word

Basics

- **miscellaneous**

translate virtual address

address map?

ok 10000000 map?

VA:10000000

G:0 W:0 P:1 E:0 CV:1 CP:1 L:1 Soft1:b PA[40:13]:c00 PA:1800000

Diag:0 Soft2:0 IE:0 NFO:0 Size:3 V:1

PA:1800000

switch cpu

cpu# switch-cpu

{0} ok 1 switch-cpu

{1} ok

Basics

- **miscellaneous (cont.)**

make os dump core

sync

display fp registers

.fregisters

Obpsym

- **allows obp to see kernel symbols**
- ***name* or *:name* searches all modules**

```
ok :main dis
```

```
10090528 genunix:main      save    %o6, ffffffff70, %o6
```

- ***module:name* searches specified module**

```
ok sd:_init dis
```

```
603a2000 sd:_init      save    %o6, fffffffffa0, %o6
```

- **enabled by default on debug kernel**
- **manually enabled by “set obpsym=1” in /etc/system**
- **avoid name conflicts with *:name* format**

Obpsym

- **symbol lookup control**

disable symbol lookup symbol-lookup-off

enable symbol looks symbol-lookup-on

- **commands that do lookups**

dis, +dis

ctrace

.adr

 ok 10000000 .adr

 10000000 trap_table

Forthdebug

- **forth-based debugger downloaded at boot**
 - similar to kadb's \$<xxx commands
 - much smaller footprint than kadb
 - not as buggy as kadb
 - sun4u only (has been backported to sun4m)
- **enabled by default on debug kernel**
- **manually enabled by “set forthdebug=1” in /etc/system**
- **must be activated each time obp is entered**

```
ok kdbg-words
```

```
ok words
```

```
.glm_unit   glm_unit-words       .glm_dsa
```

```
glm_dsa-words .glm_scsi_cmd glm_scsi_cmd-words       .glm
```

```
glm-words   clr_pagecol_stats     .io_mem_list_list
```

```
...
```

Forthdebug

- **kadb equivalents**

```
ok .threadlist
```

```
thread 10404080 pc: 1009bf30 sp: 10403a98
```

```
call 10090528 genunix:main from 10006cb8 _start+15c
```

```
( 1040b414 10463b7c 2 1042a98c 0 6e )
```

```
call 10050f40 krtld:exitto from 10051c9c krtld:kobj_init+1b0
```

```
( 10006b5c 10422194 3c 1045f888 170eb0 0 )
```

```
thread 30003ea0 free
```

```
...
```

Forthdebug

- **kadb equivalents (cont.)**

```
ok %l7 .regs
```

```
r_y = 0
```

```
r_npc = 127a4
```

```
r_pc = 127a0
```

```
r_o7 = 12
```

```
r_o6 = efff1d8
```

```
...
```

```
r_tstate = 82001a03
```

Forthdebug

- **kadb equivalents (cont.)**

ok %o5 .vnode

v_filocks = NULL

v_data = 6042dd90

v_rdev = 0 0

v_type = VDIR

v_pages = 104f5b00

v_stream = NULL

v_vfsp = 1045f2ac

v_op = 60360a18 ufs:ufs_vnodeops

v_vfsmountedhere = NULL

v_count = 59

v_flag = 1

Forthdebug

- **kadb extensions**

```
ok 2 .page-n
```

```
page 104dd400
```

```
p_share = 1
```

```
...
```

```
p_pagenum = 2
```

```
p_mapping = 104d5b70
```

```
p_paget =
```

```
    p_state = 0
```

```
...
```

```
    p_prev = 104dd400
```

```
    p_next = 104dd400
```

```
    p_vpnext = 104dd440
```

```
    p_vnode = 104213c4
```

Forthdebug

- **kadb extensions (cont.)**

ok 0 cpu-ttr

cpuid = 0

tick	type	level	trap PC
80002b7a80cace7a	dc	1	f0052850
80002b7a80cacd54	9c	1	f0052824
80002b7a80cacc14	9c	1	f0052824

ok 0 cpu-xttr

cpuid = 0

tick = 80002b7a80cace7a

trap type = dc

trap level = 1

trap pc = f0052850

Forthdebug

- **kadb extensions (cont.)**

More [`<space>`,`<cr>`,`q`,`n`,`p`,`c`] ?

<code><space></code>	next page
<code><cr></code>	next line
<code>q</code>	quit
<code>n</code>	quit
<code>p</code>	next page
<code>c</code>	continue without paging

Forthdebug

- **adding new words**

- fdbg files live in *uts/sun4u/forthdebug/fdbg*
- follows uts src layout (e.g. common structs in *common.fdbg*)
- add `#include` at top if not already there
- add struct name in proper section
- add optional field descriptions
- beware of *forth_start* and *forth_end* if not forth literate
- default is to print fields in hex

- **example #1 (struct page)**

- added to *common.fdbg*

...

```
#include <vm/page.h>
```

...

```
page
```

Forthdebug

- **format specifiers**

hex	x
decimal	d
symbol	.adr
string	.str
previous struct or enum	.foo
character	emit

Forth

- **language obp is written in**
- **interpreted (compilation not needed)**
- **postfix notation**
 - EE grads - think hp calculators
 - CS grads - think lisp backwards without parenthesis

```
ok 2 3 + .
```

```
5
```

```
ok 2 3 + 5 * .
```

```
25
```

```
ok main 8 + +bp
```

Forth

- **numeric output**

ok decimal 3 7 * .

21

ok hex 3 7 * .

15

ok octal 3 7 * .

25

ok d# 10 h# 10 * .h

a0

ok 10 .d

16

Forth

- **arithmetic**

ok 4 2 + .

6

ok 4 2 - .

2

ok 4 2 * .

8

ok 4 2 / .

2

ok 4 2 max .

4

Forth

- **arithmetic (cont.)**

ok 4 2 min .

2

ok fc 17 and .

14

ok fc 17 or .

ff

ok fc 17 xor .

eb

Forth

- **stack duplication**

```
ok showstack
```

```
1 2 ok dup
```

```
1 2 2 ok
```

```
1 2 3 ok 2dup
```

```
1 2 3 2 3 ok
```

```
1 2 3 4 ok 3dup
```

```
1 2 3 4 2 3 4 ok
```

Forth

- **stack duplication (cont.)**

1 2 3 ok over

1 2 3 2 ok

1 2 3 4 5 ok 2over

1 2 3 4 5 2 3 ok

1 2 3 ok tuck

1 3 2 3 ok

Forth

- **stack removal**

1 2 3 4 5 ok clear

ok

1 2 ok drop

1 ok

1 2 3 ok 2drop

1 ok

1 2 3 ok nip

1 3 ok

Forth

- **stack rearrangement**

1 2 3 ok swap

1 3 2 ok

1 2 3 4 5 ok 2swap

1 4 5 2 3 ok

1 2 3 4 ok rot

1 3 4 2 ok -rot

1 2 3 4 ok

Forth

- **memory access**

```
ok 10000000 c@
```

```
10 ok
```

```
ok 10000000 w@
```

```
1080 ok
```

```
ok 10000000 l@
```

```
10801978 ok
```

```
ok 10000000 x@
```

```
108019788f414000 ok
```

Forth

- **defining new words**

```
ok : hello-world ." hello world" ;
```

```
ok hello-world
```

```
hello world
```

- **using new words**

```
ok ' hello-world is .breakpoint
```

```
ok trap +bp
```

```
ok go
```

```
hello world
```

```
ok
```

Forth

- **using new words (cont.)**

```
ok : o0 ." %o0 = " %o0 .h ;
```

```
ok o0
```

```
%o0 = 10000000
```

```
ok trap +bpx o0
```

```
ok go
```

```
%o0 = 301b3ae8
```

Forth

- **conditional branches**

```
ok : is-true? if ." is true" then ;
```

```
ok true is-true?
```

```
is true
```

```
ok false is-true?
```

```
ok : what-is if ." is true" else ." is false" then ;
```

```
ok 0 what-is
```

```
is false
```

```
ok 1 what-is
```

```
is true
```

Forth

- **comparison operators**

: what-is if ." is true" else ." is false" then ;

ok 0 0 = what-is

is true

ok 0 0 <> what-is

is false

ok 1 3 > what-is

is false

ok 1 3 < what-is

is true

ok 1 1 <= what-is

is true

Forth

- **comparison operators (cont.)**

ok 1 3 >= what-is

is false

ok 0 0= what-is

is true

ok -1 0 u> what-is

is true

ok 5 4 6 between what-is

is true

ok 5 6 7 between what-is

is false

Forth

- **conditional loops**

```
ok : up-to-10 begin dup 10 < while dup .h 1+ repeat ;
```

```
ok 1 up-to-10
```

```
1 2 3 4 5 6 7 8 9 a b c d e f
```

```
ok 5 up-to-10
```

```
5 6 7 8 9 a b c d e f
```

```
ok : down-to-10 begin dup .h 1- dup 10 = until drop ;
```

```
ok 11 down-to-10
```

```
11
```

```
ok 20 down-to-10
```

```
20 1f 1e 1d 1c 1b 1a 19 18 17 16 15 14 13 12 11
```

Forth

- **conditional breakpoints**

```
ok : stop-on-null %o0 0<> if go else ." thar' she blows" then ;
```

```
ok trap +bpx stop-on-null
```

```
ok go
```

```
(sometime later...)
```

```
thar' she blows
```

Forth

- **counted loops**

```
ok 2 0 do ." window " i . cr i .window loop
```

```
window 0
```

	INs	LOCALs	OUTs
0:	f005b2f8	1042b400	10000000
...			
7:	1003da5c	0	10007978

```
window 1
```

	INs	LOCALs	OUTs
0:	0	1002b924	f005b2f8
...			
7:	10024d38	0	1003da5c

Forthdebug

- **each forthdebug entry creates two objects**

foo-words - a vocabulary containing one word per member

.foo - a word that prints each word in *foo-words*

- **example #1**

adaptive_mutex

creates:

adaptive_mutex-words - vocabulary with words *m_type*, *m_wlock*, *m_waiters*, *m_owner*, and *m_lock*; each of which retrieve the corresponding struct member

.adaptive_mutex - a word that prints the members of an *adaptive_mutex*

Forthdebug

- **example #1 (cont.)**

```
ok kdbg-words
```

```
ok also adaptive_mutex-words
```

```
ok words
```

```
m_type      m_wlock      m_waiters    m_owner      m_lock
```

```
ok : is-adap %o0 m_type 0<> if go else %o0 .h then ;
```

```
ok mutex_adaptive_enter +bpx is-adap
```

```
ok go
```

```
6024aed8
```

Forth

- **example #1 (cont.)**

```
ok kdbg-words
```

```
ok %o0 .adaptive_mutex
```

```
m_type = 0
```

```
m_wlock = 0
```

```
m_waiters = 0
```

```
m_owner = 2fea0
```

```
m_lock = 30
```

Forth

- **adding debug scripts**

- added between “forth_start” and “forth_end” in *.fdbg* files
- use “[also *foo*-words] “ to add vocabularies
- use “symbol” to lookup kernel symbols
 - necessary for debug symbols
 - good idea elsewhere
- comments antecede “\” or are within “(“ and “)”
- use stack comments liberally

Forth

- **example #2**

```
: .module-list ( -- )
```

```
  [ also modctl-words ]
```

```
  symbol modules dup          ( mod0 mod0 )
```

```
  begin                       ( mod0 mod )
```

```
    dup mod_next swap        ( mod0 mod' mod )
```

```
    mod_mp ?dup if          ( mod0 mod' mp )
```

```
      .module cr            ( mod0 mod' )
```

```
  then                        ( mod0 mod' )
```

```
  2dup = until              ( mod0 mod' )
```

```
  2drop                     ( )
```

```
  [ previous ]
```

Forth

- **example #2 (cont.)**

```
ok .module-list
```

```
filename = /platform/sun4u/kernel/unix
```

```
bss = 0
```

```
bss_size = 0
```

```
...
```

```
filename = misc/krtld
```

```
bss = 104351ec
```

```
bss_size = 11542
```

```
...
```

Futures

- **Better help facilities**

- macro writers can specify docstrings ala elisp
- forthdebug commands to print available docstrings

- **Kadb compatibility mode**

- alpha version available now
- accepts kadb-style commands
 - *address* , *count command*
 - most *address* expressions recognized
 - \$q \$r \$c \$b \$d commands
 - \$< macros with forthdebug equivalents
 - :d :z :b :s :c commands
 - most *command* formats recognized
- converts kadb commands to forthdebug

Bibliography

- **Mastering Forth**
 - Anita Anderson and Martin Tracy
 - Brady Communications Company, Inc
- **Forth: A Text and Reference**
 - Mahlon G. Kelly and Nicholas Spies
 - Prentice Hall, Inc
- **Starting Forth**
 - Leo Brodie
 - Prentice Hall, Inc
 - ISBN 0-13-843079-9