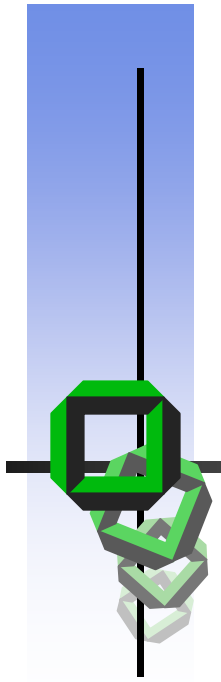


Systems Design and Implementation

II.4 – Debugging

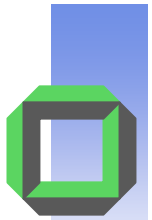
System Architecture Group, SS 2009

University of Karlsruhe

May 14, 2008

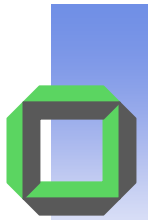
Jan Stoess

University of Karlsruhe



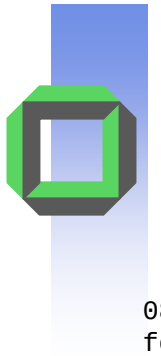
Potential Sources of Bugs

Your code	**Debug it! Fix it!**
Compiler	**No chance**
VMware	**Use real hardware!**
L4 Kernel / IDL^4	**Tell us!**
Your code	**Debug it! Fix it!**
Your code	**Debug it! Fix it!**
Your code	**Debug it! Fix it!**
Your code	**Debug it! Fix it!**



Inspecting Compiler Output

- You build an OS ...
  - Be concerned about efficiency
  - Check what the compiler did to your code
- Symbol table dump **`nm -n objfile`**
  - Shows address of symbol in object file
- Disassembler **`objdump -d objfile`**
  - **`-d`** disassemble
  - **`-l`** show source file line numbers
  - **`-S`** show source code
  - **`-l`** and **`-S`** require debug info in object
    - SDI builds with debug symbols by default (**`-ggdb`**)



Understanding **objdump** Output

```
080483b4 <foo>:
foo():
/tmp/test.c:7

int foo(int x)
{
 80483b4:       55                      push   %ebp
 80483b5:       89 e5                   mov    %esp,%ebp
 80483b7:       83 ec 08                sub    $0x8,%esp
 80483ba:       8b 45 08                mov    0x8(%ebp),%eax
/tmp/test.c:8
  int y = bar(x);
 80483bd:       83 c4 f4                add    $0xfffffff4,%esp
 80483c0:       50                      push   %eax
 80483c1:       e8 de ff ff ff          call   80483a4 <bar>
/tmp/test.c:9
  return 7 + y;
 80483c6:       83 c0 07                add    $0x7,%eax
 80483c9:       89 ec                   mov    %ebp,%esp
 80483cb:       5d                      pop    %ebp
 80483cc:       c3                      ret
/tmp/test.c:10
}
 80483cd:       8d 76 00                lea    0x0(%esi),%esi
```

objdump –d test
objdump –dl test
objdump –dlS test

Prologue

Local variables
Load parameter

Prepare function call

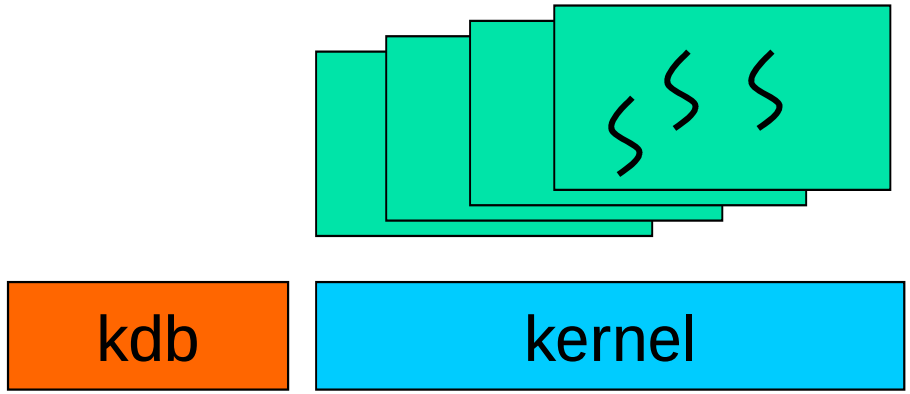
Call function bar

Add 7 to result
Epilogue

Return to caller

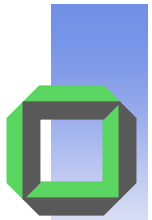
Byte stuffing for function alignment

The L4 Kernel Debugger



Conceptually:

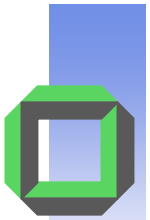
- Kernel does not know about KDB
- KDB knows about kernel internals
  - Display state of kernel objects (threads, ASs, ...)
  - Monitor kernel operations (system calls, exceptions, ...)
- Meant for debugging the kernel, but ...



L4 Kernel Debugger API

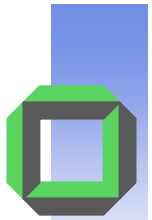
- L4_KDB_...
  - IA32: **`int3`** instruction followed by special code sequence
  - **`L4_KDB_PrintChar(char c)`**
    `L4_KDB_PrintString(char* s)`
    `L4_KDB_ClearPage()`
  - **`char L4_KDB_GetChar()`**
    `char L4_KDB_GetCharBlocked()`
  - **`L4_KDB_Enter("Oops!");`**
    - Invokes interactive KDB and prints message
    - String must be mapped

- Extended by SDIOS
  - **`#define bailout(S)`** **`L4_KDB_Enter(S)`**
  - **`#define assert(X)`** **`if ( !(X) ) { panic( #X ) }`**
  - **`void panic(char* s)`** **`{ printf(s);bailout("panic"); }`**



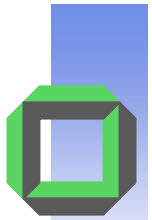
The L4Ka::Pistachio Kernel Debugger

- You can use it to ...
  - Stop the running system – ESC
  - Inspect thread state – t/T
  - Trace kernel events – r
  - Dump memory – d/D
  - View page tables – p
  - View mapping database – m
  - Set breakpoints – b
  - Dump the Kernel Interface Page – K
  - Single-step your program – s
  - Disassemble code – U
- May help you find cause for problems
  - Post mortem analysis – What happened?
  - Tracing – What's going on?



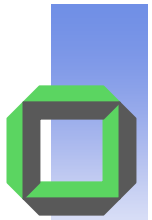
L4Ka::Pistachio KDB Navigation

- Menu root
  - BS - back up to previous menu
  - ? - this help message
  - ESC - back to previous menu
  - b - set breakpoints
  - U - disassemble
  - B - generic bootinfo
  - SPC - show current exception frame
  - F - show exception frame
  - K - dump kernel interface page
  - m - dump mapping database
  - p - dump page table
  - g - continue execution
  - d - dump memory
  - D - dump memory in other space
  - 6 - Reset system
  - q - show scheduling queue
  - t - show thread control block
  - T - shows thread control block (extended)
  - s - Single step
  - a – architecure specifics
    - C – dump CPU features
    - I – dump interrupt controller
    - V – dump VGA screen contents
    - a – ACPI access
    - c – show IA32 control registers
    - g – dump the GDT
    - i – dump the IDT
    - m – dump model specific registers
    - n – control nmi handling
    - p – IO port access
    - s – Dump small spaces
  - # – statistics
    - k – kernel memory allocator statistics
  - 0 – sigma0 interaction
    - m – dump sigma0 memory pools
    - v – change sigma0 verboseness
  - c – KDB configuration
    - m – change kernel debugger operation mode
    - c – Toggle console
    - w – change memdump word size
  - r - enable/disable/list tracepoints
    - d - disable tracepoint
    - D - disable all tracepoints
    - e - enable tracepoint
    - E - enable all tracepoints
    - l - list tracepoints
    - R - reset counters



g – Leave KDB to Continue

```
--- "KD# System started (press 'g' to continue)" ---
---------------------- (eip=f01050df, esp=f0105fa0) ---
> go
idle thread started on CPU 0
Early system infos:
Threads: Myself:c8001 Sigma0:c0001
```



[space bar] – Dump Exception Frame

- Processor state when KDB was invoked
- Invoker can be user (debug API) or in kernel

IP
(f01xxxxx = kernel)

SP
Only valid when user IP

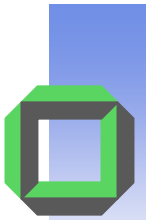
Exception frame address
in kernel TCB

```
> frame
fault addr: 00200196    stack: 0020a2f0 error code: 4 frame: e00197bc
eax: 00000000   ebx: 00209008
ecx: 0020a2dc   edx: 0020a3cc
esi: e0019000   edi: bf000100
ebp: 0020a3f8   efl: 00003246 [cPaZsodItr3]
cs:      001b   ss:      0023
ds:      0023   es:      0023
>
```

HW error code
optional

General purpose register set
+ segment registers

EFLAGS
Caps indicates bit set



t – Inspect Thread Control Block

- Thread state – TCRs, queues

User IP/SP
Where is it?

State
What is it doing?

Global ID

Local ID

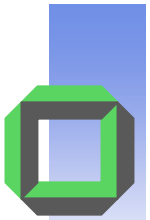
Address space

```
> showtcb
tcb/tid [current]: 000c8001
=== TCB: e0019000 === ID: 000c8001 = bf000100/f0146000 === PRIO: 0xff ========
UIP: 00203da1   queues: rswl      wait : 00000000:00000000   space: f0142000
USP: 0020a2e4   tstate: RUNNING   ready: e0018000:e0018000   pdir : 00142000
KSP: e00196d4   sndhd : 00000000  send : 00000000:00000000   pager: 000c0001
total quant:              0us, ts length:        10000us, curr ts:      9480us
abs timeout:              0us, rel timeout:          0us
sens prio: 255, delay: max=0us, curr=0us
resources: 00000000 []
partner: 00000000, s.partner: 00000000, s.state: RUNNING, scheduler: 000c8001
>
```

IPC partner
Who does/did it communicate with?

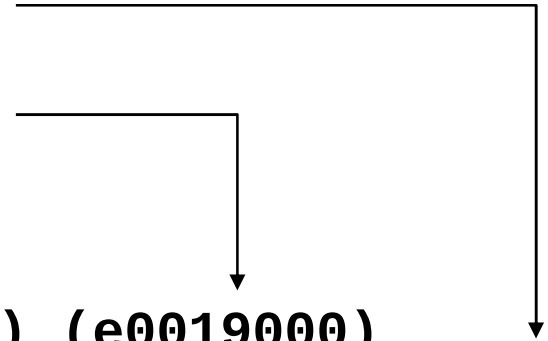
T – Extended Thread Control Block Dump

```
> showtcbext
...
user handle:       00000000  cop flags:      00        preempt flags:     00 [~~~]
exception handler: 00000000  virtual sender: 00000000  intended receiver: 00000000
xfer timeouts:     snd (never)
                   rcv (never)

mr( 0): ffe50002 00001000 00203da1 00000000 00000000 00000000 00000000 00000000
mr( 8): 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
mr(16): 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
mr(24): 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
mr(32): 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
mr(40): 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
mr(48): 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
mr(56): 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Message Tag: 2 untyped, 0 typed, label = ffe5, flags = ----

br( 0): 00000010 00000000 00000000 00000000 00000000 00000000 00000000 00000000
br( 8): 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
br(16): 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
br(24): 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
br(32): 00000000
Acceptor: 00000010 (s)
  fpage : (COMPLETE-FPAGE)
>
```

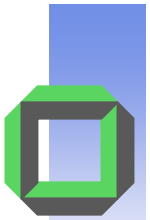


q – Scheduler Queue

- Show threads in the system
  - [prio]
  - runnable
  - (blocked)



```
> showqueue
[255]: (e0018000) (e0019000)
[100]: (e0019800) (e001a000) e001a800
[  0]: (e0008000)
idle : f0124000
>
```



p – Dump Page Table

```
> ptab
Space [current]: f0142000
Memory area (Complete/User/Kernel/Tcb/Small spaces) [complete]: user
00000000 [0014a027]: tree=f014a000
00200000 [00200027]:   phys=00200000  map=f014701c   4KB rwx (R~X) user WB
00201000 [00201027]:   phys=00201000  map=f0147040   4KB rwx (R~X) user WB
00202000 [00202027]:   phys=00202000  map=f0147064   4KB rwx (R~X) user WB
00203000 [00203027]:   phys=00203000  map=f0147088   4KB rwx (R~X) user WB
00204000 [00204027]:   phys=00204000  map=f014704c   4KB rwx (R~X) user WB
00209000 [00209027]:   phys=00209000  map=f0147028   4KB rwx (R~X) user WB
0020a000 [0020a067]:   phys=0020a000  map=f0147034   4KB rwx (RWX) user WB
0020b000 [0020b067]:   phys=0020b000  map=f0147058   4KB rwx (RWX) user WB
0020c000 [0020c067]:   phys=0020c000  map=f0147070   4KB rwx (RWX) user WB
0020d000 [0020d067]:   phys=0020d000  map=f014707c   4KB rwx (RWX) user WB
bf000000 [00148027]: tree=f0148000
bf000000 [00146027]:   phys=00146000  map=bf000000   4KB rwx (R~X) user WB
bfc00000 [00144027]: tree=f0144000
bff00000 [00123025]:   phys=00123000  map=bff00000   4KB r~x (R~X) user WB
>
```

Permissions

Page size

User/Kernel

UTCB area

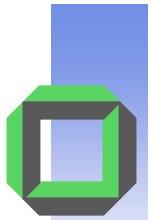
KIP

Reference bits

Virtual address

Physical address

Cacheability



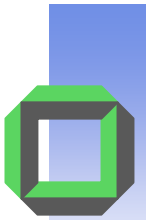
d – Dump Memory Contents

Words (not bytes)

Printable characters

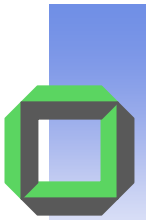
```
> memdump
Dump address [0x0]: 00204f00
00204f00  c35dc0b6 83e58955 458b08ec 24048908   ¶.].U.å. ì..E...$
00204f10  fffea7e8 00c3c9ff 90e58955 00107d83   è§.ÿÿ... U.å..}..
00204f20  458b2674 00388008 458b1e75 0c558b08   t&.E..8. u..E..U.
00204f30  3a00b60f eb027402 0845ff0f ff0c458d   .¶.:.t.ë .ÿE..E.ÿ
00204f40  10458d00 d4eb08ff 0f08458b 558b00be   ..E.ÿ.ë. .E.....U
00204f50  12be0f0c c35dd029 00000000 00000000   ....).]. ........
00204f60  532a2a2a 65747379 7473206d 6570706f   ***Syste m stoppe
00204f70  2a2a2a64 00000000 00000000 00000000   d***.... ........
00204f80  746f6f52 6b736174 746e6520 64657265   Roottask  entered
00204f90  69616d20 0a29286e 6f6f5200 73617474    main(). .Roottas
00204fa0  6168206b 68542073 64616572 25204449   k has Th readID %
00204fb0  00000a78 00000000 00000000 00000000   x....... ........
00204fc0  746f6f52 6b736174 73616820 65784520   Roottask  has Exe
00204fd0  6f697470 6148206e 656c646e 78252072   ption Ha ndler %x
00204fe0  6f52000a 6174746f 68206b73 50207361   ..Rootta sk has P
00204ff0  72656761 0a782520 6f6f5200 73617474   ager %x. .Root###
Continue? (Continue/Quit) [continue]: continue
00205000  ######## ######## ######## ########   ######## ########
```

Page not mapped



r – Tracepoints

- Trace kernel events as they happen
  - Count occurrences (always on)
  - Optionally display event information (enable = y)
  - Optionally enter kernel debugger (enter kdb = y)
- Important tracepoints
  - SYSCALL_* – system calls with parameters
  - PAGEFAULT_* – page faults caused by user/kernel
  - IPC_* – IPC message transfer in detail
  - FPAGE_MAP – mapping established
  - INTERRUPT – hardware interrupt fired
- rE – enable all



b - Breakpoints

- IA-32: four hardware breakpoints

```
> breakpoint
breakpoint [-/?/0..3]:  (-/?/0/1/2/3) [?]: 0
Type (Instr/Access/pOrt/Write/-/+) [instr]: instr
Address [0x0]: 200196
> go
--- Debug Exception ---
> breakpoint
breakpoint [-/?/0..3]:  (-/?/0/1/2/3) [?]: ?
DR7: 00000402
DR6: ffff0ff1
DR3: 00000000
DR2: 00000000
DR1: 00000000
DR0: 00200196 *
> frame
fault addr: 00200196    stack: 0020a2f0 error code: 4 frame: e00197bc
eax: 00000000   ebx: 00209008                  ...
```

Disable all

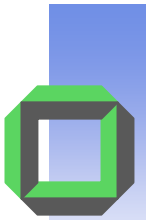
Pick breakpoint register 0-3

Select type
execution
read/write
in/out
write
disable
enable

Address

IA-32 debug registers

Breakpoint 0 hit



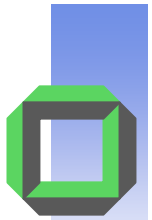
K – Dump the Kernel Interface Page

```
...
  UTCB area info    min size: 4KB, alignment: 512, UTCB size: 512
  KIP area info     min size: 4KB
  Boot info ptr     0x00001000
  Thread info       user base: 0x030, system base: 0x010, thread bits: 17
  Page info         sizes: 4K 4M, rights: rw
Root servers:
  sigma0            ip: 0x00021b90, sp: 0x00000000, 0x00020000:0x000283a0
  sigma1            ip: 0x00000000, sp: 0x00000000, 0x00000000:0x00000000
  root server       ip: 0x00200000, sp: 0x00000000, 0x00200000:0x0020d320
Memory regions (19):
  Physical:         0x00000000 - 0xffffffff   shared
                    0x00000000 - 0x0009f7ff   conventional
                    0x0009f800 - 0x0009ffff   architecture specific (2)
                    0x000dc000 - 0x000dffff   architecture specific (2)
                    0x000e4000 - 0x000fffff   architecture specific (2)
                    0x00100000 - 0x01eeffff   conventional
                    0x01ef0000 - 0x01efefff   architecture specific (3)
                    0x01eff000 - 0x01efffff   architecture specific (4)
                    0x01f00000 - 0x01ffffff   conventional
                    0xfec00000 - 0xfec0ffff   architecture specific (2)
                    0xfee00000 - 0xfee00fff   architecture specific (2)
                    0xfffe0000 - 0xffffffff   architecture specific (2)
                    0x000a0000 - 0x000bffff   shared
                    0x000c0000 - 0x000effff   shared
                    0x01000000 - 0x01eeffff   reserved
                    0x00889000 - 0x00897bff   bootloader specific (1)
                    0x00001000 - 0x00001fff   bootloader specific (0)
                    0x00100000 - 0x0014ffff   reserved
  Virtual:          0x00000000 - 0xbfffffff   conventional
```

GRUB-loaded modules

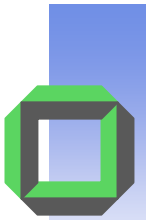
MultiBootInfo

Kernel



If nothing else helps anymore ...

- Use a wide terminal window for minicom
- Press ? for list of available commands
- Or press 6 to reboot
  - Warning: PgDown over serial line causes reboot
    - ^[[6~ = ESC [6 ~



Lecture Schedule

21.4.	Introduction	23.4.	L4 API Crash Course (I)
28.4.	Communication	30.4.	L4 API Crash Course (II)
5.5.	OS Interfaces	7.5.	IDL4, Debugging on L4
12.5.	Naming	14.5.	Debugging on L4 (Lab)
19.5.	J. Stoess – Project Kittyhawk	21.5.	- Christi Himmelfahrt -
26.5.	File Systems	28.5.	Name Service Design (3)
2.6.	Threads, Scheduling	4.6.	File Service Design (2)
9.6.	Memory Management	11.6.	- Fronleichnam-
16.6.	Drivers	18.6.	Task Service Design (2)
23.6.	Device Service Design (2)	25.6.	MM Service Design (2)
30.6.	Lab	2.7.	Lab
7.7.	Lab	9.7.	Lab
14.7.	Lab	16.7.	Lab
21.7.	Lab	23.7.	Lab Demos + Conclusion