Fiasco Kernel Debugger Manual

Frank Mehnert
Jan Glauber
Jochen Liedtke

Technische Universität Dresden
Department of Computer Science

November 2006
Version 1.1.6

Revision History

20.04.2001 – Version 1.00 *(Jan Glauber)*
- Inital release based on Jochen's LN kdb manual.

18.07.2001 – Version 1.0.1 *(Frank Mehnert)*
- added documentation for u command (disassemble memory)
- added documentation for ls and ld command (enable/disable IPC result logging)
- added documentation for Ss and Sd command (enable/disable show thread tcb after singlestop)
- all commands now expect thread numbers by t*xxx.yy* notation

23.08.2001 – Version 1.0.2 *(Frank Mehnert)*
- added documentation for lp and lr commands
- completed documentation for T command
- when expecting a thread/task, *txxxxxxxx* notation is possible
- described j command
- ls/ld → l+/l–, Ss/Sd → t+/t–

11.10.2001 – Version 1.0.3 *(Frank Mehnert)*
- added description for B command
- extended description for trace-buffer dump
- some cosmetical changes

14.11.2001 – Version 1.0.4 *(Frank Mehnert)*
- added some remarks on symbols section
- some cosmetical changes

10.5.2002 – Version 1.0.5 *(Hendrik Tews)*
- added r command

15.12.2003 – Version 1.1 *(Frank Mehnert)*
- major update (added commands A, C, M, O, R)
- added command line switch reference
- various changes

27.12.2003 – Version 1.1.1 *(Adam Lackorzynski)*
- added command J, C is gone

03.05.2004 – Version 1.1.2 *(Adam Lackorzynski)*
- added lt, Jo, t? commands
- removed s command
- added `-kmemsize` parameter switch

13.10.2004 – Version 1.1.3 *(Frank Mehnert)*
- added N, ka, kb, kP, C commands
- cosmetical fixes
- `-loadcnt` switch documentation
- commands for setting marks and jumping to marks

07.11.2005 – Version 1.1.4 *(Frank Mehnert)*
- added Jd command
- added Z command

05.12.2005 – Version 1.1.5 *(Frank Mehnert)*
- added PR+/− command
- enter_kdebug("*#") description improved
- documentation of -comport=*n* for $n > 4$
- typos

15.09.2006 *(Adam Lackorzynski)*
- Exchange -nowait with new command -wait

11.11.2006 – Version 1.1.6 *(Adam Lackorzynski)*
- Describe keys in memory dump.

Contents

1 General Remarks

About JDB

The Fiasco Kernel Debugger (JDB) is primarily a tool to test Fiasco and to fix bugs in Fiasco. However, it can sometimes also be useful for testing higher layers. Consequently,

- JDB always freezes the entire machine state when invoked. All interrupts are disabled, even the clock is halted. No kernel or user process proceeds while JDB is active.
- JDB is a stand-alone debugger. It does neither use parts of Fiasco nor other device drivers. JDB includes simple device drivers for keyboard, display (CGA and hercules), and the serial interfaces COM1 and COM2. These device drivers are *not* interrupt driven.

Although developed and traditionally packaged together with Fiasco, the Fiasco Kernel Debugger is *not* part of the Fiasco µ-kernel. Fiasco can run without JDB or with another kernel debugger (see compile options). There is no other connection between Fiasco and JDB that JDB intimately knows Fiasco's data structures.

About Exception Handling

JDB is invoked by exceptions, in particular by the debug exception (INT 1) and the breakpoint exception (INT 3). Some exceptions are handled normally by Fiasco, for example a page fault. All other exceptions are distributed by Fiasco either to JDB or to the current user-level thread. The according algorithm is:

exception dispatch:
if excption occurred in kernel
then invoke JDB with kernel error
elif current thread defined an IDT AND corresponding IDT entry $\neq 0$
then invoke user-level exception handler by upcall
else invoke JDB with user error
fi .

Summarizing: JDB is used for kernel errors. It is used for user errors only as long as no debugger is installed at user level.

2 Fiasco-Specific Functions

2.1 Thread Control Block: t

Displays the *commented thread control block*, the thread's *kernel stack*, and the thread's *general purpose registers*.

t	Displays the thread control block of the current thread.
t*xxx.yy*	Displays the thread control block of thread *yy* of task *xxx*. *xxx* and *yy* are both hexadecimal; leading zeros may be omitted. For example, thread 1 of task 4 can be denoted by 4.01 or 4.1
t*xxxxxxxx*	Same as t*xxx.yy* but specify low dword of thread id instead.
t+/−	Enable/disable automatic displaying of current thread control block when entering kernel debugger.

thread control block	The fields of the thread control block (TCB) are labeled.
kernel stack	The kernel stack of the displayed thread is dumped from the current stack top (first line) down to the stack bottom. With the cursor keys its possible to scroll inside the TCB borders of the displayed thread. When pressing the Enter key, the address at the cursor is dumped in memory mode. When pressing the Space key, the address at the cursor is disassembled. When entering kernel mode, the processor first pushes SS, ESP, EFlags, CS and EIP onto the kernel stack. So the last line always looks like ... userEIP | userCS | userEFLAGS | userESP | userSS
registers	are displayed if the *currently active* thread is displayed. For this thread, JDB knows the registers through the trap's state, which invoked JDB. Depending on their state, other threads might not have saved all their registers on the stack.
thread lists	With 'r'eady or 'p'resent followed by 'p'revious or 'n'ext, cycling through the thread queues is possible.

2.2 Thread Lists: l

Display the *present list* (lp command) or the *ready list* (lr command). The cursor keys may be used to scroll in the list. The Enter key shows the TCB of the selected thread. The Space key allow to change the sort mode: Unsorted, sorted by thread priority and sorted by thread identifier. The Tab key switches to the thread in list the current thread is waiting for.

2.3 Timeout List: lt

Display all current active timeouts. The cursor keys may be used to scroll in the list. The Enter key shows the TCB of the owner of the selected timeout.

2.4 Mapping: m

Permits to parse the Fiasco mapping database.

m*xxxxx*	Starts parsing the mapping subtree corresponding to physical page frame *xxxxx*000. The CursorUp and CursorDown keys can be used to proceed to the next or previous page. The Escape key allows to leave this mode.

2.5 Dump Backtrace: bt

Displays the backtrace of a thread.

bt	Display the user-backtrace of the last thread running (currently active thread) before entering JDB. Instruction pointers are shown from newest at the top down to oldest. If the symbols for the current task are loaded, the proper function names are also displayed.
bt*xxx.yy*	Display the user-backtrace of thread *yy* of task *xxx*. Also shows the kernel backtrace of the selected thread.
bt*xxxxxxxx*	Display the user-backtrace specifying the low dword of thread id
bt*xxxxxxxx*	Display the backtrace of the current thread interpreting address *xxxxxxxx* as framepointer.

2.6 Kernel Data: k

Displays some global Fiasco kernel data that is not thread specific.

ka	Display some information about the local APIC.
kb	Determine how many cycles does it take to execute some special instructions.
kc	Display some information about the processor.
kf	Interprets the content of the kernel info page.
kg	Display entries of the GDT (global descriptor table).
kh	Short help about available subcommands.
ki	Displays the kernel clock, current pdbr, cr0, cr4, idt, gdt, ldtr, tr, I/O bitmap pointer.
kI	Display entries of the IDT (interrupt descriptor table).
km	Kernel memory footprint.
kp	Display ports of the PIC.
KP	List all PCI devices.
kr	Information about the memory manager.
ks	Display information about small spaces.

3 General Test Functions

3.1 Breakpoint: b

Sets/resets four global breakpoints for kernel-mode and user mode.

bl	Displays the breakpoints and breakpoint restrictions
bi ...	Sets an instruction breakpoint.
b{w,a,p}{1,2,4} ...	Sets a data-write breakpoint (w) or a data-access breakpoint (a) for a 1-, 2- or 4-byte variable or sets a breakpoint for in/out to a 1-, 2- or 4-byte i/o port (p)
... *xxxxxxxx*	The breakpoint address can be absolute or relative. *xxxxxxxx* is the breakpoint address. *xxxxxxxx* is a hexadecimal number where leading zeros can be omitted.
b–[*bpn*]	Resets a system-global breakpoint. The number of the breakpoint *bpn* (1..4) has to be specified
b+[*bpn*]	Enter kernel debugger when breakpoint matched (default action)
b*[*bpn*]	Don't enter kernel debugger when breakpoint matched, instead create a tracebuffer entry

br[*bpn*] ...	Restricts a breakpoint. The breakpoint exception invokes JDB only when all set restrictions are met. Otherwise, the breakpoint exception is ignored. The number of the breakpoint *bpn*, which should be restricted must be specified.
... t*xxx.yy*	Restricts breakpoints to thread *xxx.yy*.
... T*xxx.yy*	Restricts breakpoints to threads ≠*xxx.yy*.
... a*xxx*	Restricts breakpoints to task *xxx*.
... A*xxx*	Restricts breakpoints to tasks ≠*xxx*.
... e[*reg*] [*yyyyyyyy*, *zzzzzzzz*]	The specified register *reg* must have a value in the specified interval (if *yyyyyyyy*≤*zzzzzzzz*) or outside the interval [*zzzzzzzz*, *yyyyyyyy*] (if *yyyyyyyy*>*zzzzzzzz*).
... {1,2,4}*xxxxxxxx* [*yyyyyyyy*, *zzzzzzzz*]	The specified 1-, 2- or 4-byte variable must have a value in the specified interval (if *yyyyyyyy*≤*zzzzzzzz*) or outside the interval [*zzzzzzzz*, *yyyyyyyy*] (if *yyyyyyyy*>*zzzzzzzz*).
... –	All breakpoint restrictions of the breakpoint with number *bpn* are reset.

Note:
Breakpoint restrictions are *not* reset when the breakpoint address or type are changed or when the breakpoint is reset (b-). Breakpoint restrictions can be explicitly reset (br-). However, changes of the restrictions do not require prior reset.
br{t,T}, br{e..} and br{1,2,4} restrictions are logically anded. However, setting a restriction overwrites a prior restriction of that type.

3.2 Dump Memory: d

Displays physical and virtual memory.

d*xxxxxxxx*	displays memory beginning from address *xxxxxxxx*. The dump is 32-bit-word oriented so that addresses are always truncated to 4-byte aligned addresses. The cursor keys, PgUp, PgDn and Home can be used to move the dump cursor and display further memory. The Enter key can be used to jump to the address at the cursor, the Home key returns if possible. This command always displays virtual memory of the current address space. Pages that are not mapped to physical memory are shown as .....
dt*yyy xxxxxxxx*	displays virtual memory of task *yyy* beginning from address *xxxxxxxx*.
dt*yyyyyyyy xxxxxxxx*	displays virtual memory of a task specifying the low dword of the thread id beginning from address *xxxxxxxx*.

Memory can be displayed in various modes. The Space key switches between these modes:

d-mode	32-bit words are shown as dwords, uppermost nibble leftmost, e.g. 00000002 for the value 2. Values 0 and FFFFFFFF are displayed specially: 0 and -1.
b-mode	32-bit words are shown bytewise, uppermost byte righmost, e.g. 02000000 for the value 2. Values 0 and FFFFFFFF are not treated differently.
c-mode	Bytes are shown as ASCII characters. Unprintable characters are shown as .

The view offers several keys to change the view:

Space	Changes display modes between d, b and c mode, see above.
Return	Goes to the address under the cursor.
Tab	Whether to show contents of adapter memory or not. Reading adapter memory may cause unwanted behaviour. Default is not to show adapter memory.
e	Memory values can be edited. When pressing the key, a new value for the address where the cursor points to can be entered.
c	Pressing the key highlights all values which are up to 0x100000 below or above the current value. Only works in d-mode.
u	Changes into disassembler view with the value under the cursor preselected as the address as well as the current task.

3.3 Physical memory: A

Display or modify physical memory.

A*xxxxxxxx*	Read 32 bits from physical address *xxxxxxxx* and display the result
Aw*xxxxxxxx yyyyyyyy*	write 32 bits contained in *yyyyyyyy* to address *xxxxxxxx*

3.4 Disassemble Memory: u

Disassembles virtual memory.

u*xxxxxxxx*	disassembles memory beginning from address *xxxxxxxx*. The cursor keys, PgUp and PgDn can be used to move one line backwards or forwards or to jump one page backwards or forwards, respectively. This command always disassembles virtual memory of the current address space. Pages that are not mapped to physical memory are shown as .....
ut*yyy xxxxxxxx*	disassembles virtual memory of task *yyy* beginning from address *xxxxxxxx*.
ut*yyyyyyyy xxxxxxxx*	disassembles virtual memory of a task specifying the low dword of the thread id beginning from address *xxxxxxxx*.

3.5 Dump Page Tables: p

Displays page tables and virtual memory.

p	displays the higher-level page table (page directory) of the current address space. The cursor keys, PgUp, PgDn can be used like in the dump case. Pressing the Enter key when the cursor points to a valid lower-level page table switches to this table. In the same way, the Enter key there switches to the data page. The Home key always "returns" to the lower- or higher-level page table, respectively.
p*xxx*	displays the higher-level page table of task *xxx*.
	The Space key changes between page table mode and raw mode.
	Page-table entry formats:
-	Nil entry.
xxxx--r	User-level, read-only, pointing to physical address 0*xxxx*000.
xxxx--w	User-level, read/write, pointing to physical address 0*xxxx*000.
xxxx--R	Kernel, read-only, pointing to physical address 0*xxxx*000.
xxxx--W	Kernel, read/write, pointing to physical address 0*xxxx*000.
xx/4--{r,w,R,W}	4M-page entry, pointing to physical address 0*xx*×4MB.

3.6 Page Fault Monitoring: P

Monitors page faults. Page faults are monitored *before* they are handled by Fiasco.

P+	Switches page-fault monitoring on. Whenever a page fault occurs, page-fault address, instruction pointer and thread number are displayed and the system stops until a key is hit on the debug console. Pressing the 'i' key invokes the full JDB menu; any other key resumes normal operation, i.e. starts normal page-fault handling.
P–	Switches page-fault monitoring off.
P*	Switches *traced* page-fault monitoring on. Instead of presenting any single page fault, all page faults are monitored in a trace buffer. This buffer stores up to 1024 entries. When JDB is invoked the next time, the entries can be displayed by the *dump trace* command T.
PR+	Enables monitoring of the return code of the page fault handler. To log these events, PF logging must be enabled either by P+ or by P*.
PR–	Disables monitoring of the page fault handler return code.
Pr ...	Restricts the monitored page fault. Page faults that do not meet all specified restrictions are ignored by the monitoring system.
... t*xxx.yy*	Restricts page faults to thread *xxx.yy*
... T*xxx.yy*	Restricts page faults to threads $\neq$*xxx.yy*.
... x [*yyyyyyyy*, *zzzzzzzz*]	Only page faults with fault addresses inside the interval [*yyyyyyyy*, *zzzzzzzz*] (if *yyyyyyyy*$\leq$*zzzzzzzz*) or outside the interval [*zzzzzzzz*, *yyyyyyyy*] (if *yyyyyyyy*>*zzzzzzzz*) are monitored.
... –	All page-fault restrictions are reset.

3.7 Miscellaneous Monitoring: O

Monitors miscellaneous operations. Unlike the P command and the I command, the implementation of this logging events induces additional (small) costs at runtime. Therefore these events can be completely disabled at compile time and may not be available.
For a complete list of available events type O without any argument.

3.8 Monitor "Next Period" IPC: N

Monitors the "next period" IPC.

N*	Enables monitoring of "next period" IPC.
N–	Disables monitoring of "next period" IPC.

3.9 IPC Monitoring: I

Monitors IPC operations. IPCs are monitored *before* they are handled by Fiasco.

I+	switches IPC monitoring on. Whenever an IPC operation is invoked, sender thread number, operation type (call, send, wait, wait for, reply and wait), destination thread number (if specified), message type (normal or map), message words 0 and 1 and the instruction pointer are displayed. The system stops until a key is hit on the debug console. Pressing the 'i' key invokes the full JDB menu; any other key resumes normal operation, i.e. starts normal IPC handling.
I–	switches IPC monitoring off, single as well as traced monitoring.
I*	switches *traced* IPC monitoring on. Instead of presenting any single IPC, all IPCs are monitored in a trace buffer. This buffer stores 1024 entries per default (its size can be modified using the `-tbuf_entries=` command line switch — see section 5 for details). When JDB is invoked the next time the entries can be displayed by the *dump trace* command T.
IR+	switches monitoring of IPC system call results on. To log these events, IPC logging must be enabled either by I+ or by I*.
IR–	switches monitoring of IPC system call results off
IT+	Use special log format containing both IPC data and IPC result. The format is used together with the L4 userland tracing library.
IT–	Switch back to normal IPC log format.
Ir ...	restricts the monitored IPCs. IPCs that do not meet all specified restrictions are ignored by the monitoring system.
... t*xxx.yy*	restricts IPCs to thread *xxx.yy*.
... T*xxx.yy*	restricts IPCs to threads $\neq$*xxx.yy*.
... a*xxx*	restricts IPCs to task *xxx*.
... A*xxx*	restricts IPCs to tasks $\neq$*xxx*.
... s	Only IPCs containing a send part are monitored, i.e., call, send, reply and wait.
... –	All IPC restrictions are reset.

3.10 Trace Buffer: T

Shows the trace buffer. The according messages of a trace entry are stored in a trace buffer, together with timing information and up to two values of performance counters.

T	Enters trace-buffer dump. The newest information is displayed at the top. On all processors except 486, trace-buffer entries get timestamps when they are entered into the buffer. Accordingly, they can be displayed in various timing modes.
index	Entries are shown with their number in the buffer.
tsc diff	Entries are shown with their time difference (in 40 bit hexadecimal notation or µs, ms, or s – switchable by Space key) to their displayed predecessor entry.
tsc rel	Entries are shown with their time difference relative to the reference element.
tsc start	Entries are shown with their time difference relative to the system start.
kclock rel	The kernel clock with their difference relative to the reference element is shown.
kclock	The absolute kernel clock is shown.
pmc1 diff	Entries are shown with the differences of their first performance counter.
pmc2 diff	Entries are shown with the differences of their second performance counter.
pmc1 rel	Entries are shown with the difference of their first performance counter relative to the reference element.
pmc2 rel	Entries are shown with the difference of their second performance counter relative to the reference element.

On a 486, timing information and performance counters are not available. On Pentium and all further processors, each trace-buffer entry is accomplished with a 64-bit *timestamp* which is read from the processor's internal time-stamp-counter register using the `rdtsc` instruction.
On a Pentium Pro and mostly all further processors (Pentium MMX, Pentium II, AMD K7), each trace-buffer entry can be accomplished with two 40-bit *performance counter* values which are read from the processor's internal performance counter 0 and 1.

3.10.1 Key Description

←, →	Switch between timing modes.
Space	Changes between displaying raw time stamp counter values and timed representation.
↑, ↓, PgUp, PgDn, Home, End	Move the trace-buffer cursor and display further trace-buffer entries.
c	Clear the trace buffer.
Enter	The code the event occured at is disassembled. Home then returns to the trace-buffer dump.
r	Set the *reference entry* used when displaying relative time/counter values.
jr	Jump to the reference element.
s0-9	Set mark0 .. mark9. Marks are highlighted.
j0-9	Jump to mark0 .. mark9.
/, ?, n	The trace buffer can be searched for a regular expression (similar to `less`).
F	Filter the tracebuffer view by specifying a regular expression.

3.10.2 Possible Tracing Events

Kernel events:

- IPCs and IPC results (see I* command at page 13),
- "next period" IPC (see N* command at page 12),
- page faults and page fault results (see P* command at page 12),
- other logging events (see O command at page 12),
- break points (see b* command at page 8)

User level trace events:

- ```
  #include <l4/sys/kdebug.h>
  enter_kdebug("*text");
  ```
  Creates a trace buffer entry containing the string `text`.
- ```
  #include <l4/sys/kdebug.h>
  enter_kdebug("*+text");
  ```
  Creates a trace buffer entry containing the string `text`. The values of the registers `EAX`, `ECX`, and `EDX` are stored together with the message.
- ```
  #include <l4/sys/ktrace.h>
  fiasco_tbuf_log("text");
  ```
  Creates a trace buffer entry containing the string `text`. In contrast to using the sequence `enter_kdebug("*text")`, the string `text` may be created at runtime.
  ```

- `#include <l4/sys/ktrace.h>`
  `fiasco_tbuf_log_3val("text", val1, val2, val3);`

  The string `text` and three hexadecimal values are stored into a single trace buffer entry.

3.10.3 Performance Monitoring

The following options are only accessible while the trace-buffer dump mode is active and the CPU supports performance counters.

P	Permits to activate, deactivate and change performance monitoring *while you are in the trace-buffer dump*. (This command does not work outside the trace-buffer dump. Performance monitoring is not available on 486 processors. P5, P6, and K7 processors have two counters. Not all events are available for each counter.
P[*num*]+	Turns performance monitoring for counter *num* on (kernel and user-mode activities)
P[*num*]–	Turns performance monitoring for counter *num* off
P[*num*]u	Turns performance monitoring on (only user-mode activities)
P[*num*]k	Turns performance monitoring on (only kernel-mode activities)
P[*num*]e	Count on edge. This allows to measure not only the fraction of time spent in particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be served). This option makes only sense with selected events (see hardware documentation).
P[*num*]d	Count on duration
P[*num*]?	Select a performance monitoring event from list for counter *num*. Also kernel event counter (see N* command at page 17) may be specified.

3.11 Kernel Event Counters: C

Kernel event counters must be enabled using the configuration option `JDB_ACCOUNTING`. There exist counters for address space switches, context switches, page faults, and others.

Cl	Show kernel event counters.
Cr	Reset kernel event counters.

3.12 Port I/O: i/o

I/O from/to ports of the x86 ports address space.

i{1,2,4}*xxxx*	Reads (*in*) from the specified 1-, 2-, or 4-byte port.
ip*xxxxxxxx*	Reads (*in*) from the specified PCI configuration register. It is not required to set bit 32 of *xxxxxxxx*. PCI configuaration registers are always 4-byte registers. Their address *xxxxxxxx* must always be 4-byte aligned.
o{1,2,4}*xxxx*{*yy*...}	Writes (*out*) *yy*... to the specified 1-, 2-, or 4-byte port.
op*xxxxxxxx*	Writes (*out*) *yyyyyyyy* to the specified PCI configuration register. It is not required to set bit 32 of *xxxxxxxx*. PCI configuaration registers are always 4-byte registers. Their address *xxxxxxxx* must always be 4-byte aligned.

3.13 I/O Bitmap: r

Displays mapped ports, gives information about pages mapped for the I/O bitmap, and shows the I/O port counter (used for determining privileged tasks).

r	display the I/O bitmap for the current task.
r*xxx*	display the I/O bitmap for task *xxx*.

3.14 Machine Specific Registers: M

This command allows to display/modify machine status registers.

Mr*xxxxxxxx*	read 32 bits from the machine specific register *xxxxxxxx* and display the result
Mw*xxxxxxxx yyyyyyyy*	write 32 bits *yyyyyyyy* to the machine specific register *xxxxxxxx*

4 Miscellaneous

4.1 Help: h

Shows a screen with a short description of JDB commands.

4.2 JDB options: J

Jc	Set the color of the debug prompt.
Jd-/Jd+	Disable/enable direct console.
Jh	Set the height of the screen available for JDB.
JH	Adapt the height of the screen to the height of the terminal window by detecting its current size using escape sequences.
Jo	list attached Jdb consoles.

4.3 Go: g

Resumes execution after/at the instruction which invoked JDB.

4.4 Special Go: j

Resumes execution after/at the instruction which invoked JDB.

jb	Continue until next branch (*call*, *ret*, *jmp*, *int*, *iret* instruction). This mode works using the single step mode of the processor.
jr	Continue until current function returns (until *ret* or *iret* instruction at the **current code level**). This mode uses the single step mode of the processor.
js	Single Step (in difference to S+, we do not enter the single step mode)

4.5 Single Stepping: S

If the single step mode is activated, the processor enters the kernel debugger after each instruction. **Note:** On `int 0xxx`, the whole system call is executed.

S+	enables single stepping mode
S–	disables single stepping mode

4.6 List IRQ threads: R

Rl	list all IRQ threads
Ra	attach specific IRQ to Jdb

4.7 Screen Output Buffer: B

The output buffer stores all output written to the Fiasco kernel debugger console using `outstring()`, `outchar()`, `outdec()`, `kd_display()` and so on. The default size of the output buffer is 8192 bytes. It can be changed using the `-out_buf=` option (see section 5).

B	Show complete output buffer.
B*n*	Show last n lines of output buffer.

4.8 Video Mode and Remote Console: V

Va	switches JDB output to VGA monitor
Vh	switches JDB output to hercules monitor
Vs	tries to activate source level debugging

4.9 Escape: E

With escape enabled pressing the ESC key instantly stops execution and invokes JDB. Note: a better method to enter the kernel debugger is sending any character to the serial console.

E+	activates escape
E–	deactivates escape

4.10 Debug Message: Return

Shows the debug message with trap type and the instruction pointer, where JDB was invoked.

4.11 Reboot

The ^ key (european keyboard: Shift-6) resets the machine.

4.12 Halt Thread: H

Halt a specific thread. This can be useful for instance if a thread stays in an endless loop raising page faults.

4.13 Display Thread ID of TCB address: t?

It is useful to find the appropriate thread ID belonging to a TCB address. The t? command does this job.

4.14 Symbols

Symbols are alternative notations for addresses and can replace them at any place where JDB awaits the input of an address. The s key signals JDB that a symbol name instead of an address follows. When entering a symbol, the TAB key allows symbol completion.
Symbols are useful, for instance, to specify breakpoints and memory dump addresses.

Note:
The use of symbols requires that the symbol table of the specified task table is loaded into JDB. This has be done by user level programs by using the debug trap mechanism. The symbols for the Fiasco kernel can be loaded using the RMGR (see appropriate manpage).

4.15 Remote Control from Userland

Some non-interactive JDB commands may be executed from a user level program by using

```
#include <l4/sys/kdebug.h>
enter_kdebug("*#");
```

For example, to log the IPC traffic for a specific code sequence, use

```
  /* start IPC logging inclusive results */
  enter_kdebug("*#I*IR+");

  ...

  /* IPC here */

  ...

  /* stop IPC logging */
  enter_kdebug("*#I-");
```

5 Jdb-related Command Line Switches

`-wait`	Emit a debug trap after the kernel has been initialised and before user programs are started. This will usually enter the kernel debugger and show the debugger prompt.
`-nojdb`	Disables the builtin kernel debugger JDB.
`-nokdb`	Disables the GDB stub. Should Fiasco raise an exception, it will just call the builtin jdb.
`-noscreen`	Disables output to VGA/Hercules console.
`-noserial`	Disables output to serial console. If this switch is not given, kernel messages will output additionally to the serial interface. If `-nokdb` is enabled, you can use a terminal program on the host to control Fiasco. If you are connected to a remote GDB, messages are copied to GDB's console.
`-comspeed=n`	Will set the rate of the serial interface to n bytes/second. 115200 baud is the default.
`-comport=n`	Will use COMn for serial communication. COM1 is the default. Possible values for n are 1, 2, 3, and 4. n is interpreted as I/O port if $n > 4$.
`-hercules`	Redirect kernel messages to the Hercules (or other MGA-compatible) console.
`-esc`	Enable esc hack. On every timer interrupt, ask the keyboard if the Escape key was pressed. If so, do enter into kernel debugger. Applications (e.g. L^4Linux) may be confused by dropped key events so better use `-serial_esc`.
`-kmemsize=n`	Overwrite Fiasco's heuristic for required kernel memory. Set the memory reserved for mapping trees, TCBs, and other to `n` MB.
`-watchdog`	Enable watchdog. On every timer interrupt, tell the watchdog that we are sill alive. If a task disables the interrupts and loops, the timer interrupt is'nt called anymore and after 2 seconds the watchdog releases an non maskable interrupt (NMI) which forces Fiasco to step into the kernel debugger. Requires at least an Intel PPro or AMD K7 Model 2.
`-loadcnt`	Initialize a performance counter for counting all cycles the CPU is not halted. The counter is accessible from userland via rdpmc(x) where x is 0 on P6/K7, and P4.
`-apic`	Initialize the builtin Local APIC. If the Local APIC is disabled by the BIOS but available, it is re-enabled.
`-serial_esc`	Enter jdb on serial receive interrupts. This is only necessary if kdb was disabled by `-nokdb`.
`-tbuf_entries=n`	Set number of lines to store in the debugging trace buffer. Default is 1024.
`-out_buf=n`	Set output buffer for kd_display functions to n bytes. Default is 8192.
`-jdb_cmd=cmds`	Execute Jdb commands non-interactive at startup. Example: use `-jdb_cmd=I*IR+P*PR+` to log all IPCs and page faults from startup.