Kernel function instrumentation - tool analysis

Nicholas Mc Guire

Distributed & Embedded Systems Lab
SISE,Lanzhou University, Lanzhou,P.R.China
mcguire@lzu.edu.cn, http://dslab.lzu.edu.cn
March 3, 2006

Contents

Version	Author	Date	Comment
1.0	Nicholas Mc Guire	Jan 2005	First shot
1.1	Georg Schiesser	18 Jan 2005	converted to TEX document
1.2	Nicholas Mc Guire	19 Feb 2005	cleanup
1.3	Nicholas Mc Guire	Jan 2006	2.6 revision

1. Kernel function instrumentation - tool analysis

In the framework of Work Package 5 - Boot-Time Optimization, of "A Comparative Study on Real-time enhanced Linux Variants", research on existing tools to analyze boot-times was performed. As one of the most promising tools kernel function instrumentation KFI, was investigated in detail. In its current state it has some serious problems and shortcomings - never the less it is actually usable and with some preliminary extensions usable for analysis on a system scope and not only at kernel level [**?**]. In this article we describe the tools analysis, KFI usage, and data acquisition for the 2.4.20 and 2.6.11 kernel. It should be noted that we found no serious problem when applying KFI to slightly different kernels (i.e. 2.4.26).

A brief introduction to the core technology concept and its application in user-space process and libraries is given - though incomplete. For additional information on function profiling in user-space see [**?**].

1.1. Source

The sources were obtained from the celinuxforum [4]. Sources carry no copyright, not quite clear why - most likely because the concepts was derived from a number of posts on the LKML, and then packaged at some point by Montavista. Discussion/clarification of License issues is under way with `Todd Poynor, tpoynor@mvista.com` - but it looks like its GPL as its statically linked into the kernel.

- `kfi-0.8.tar.gz` - analysis tools and a minimum documentation (`README.kfi`)
- patch for 2.4.20 (only ?) not to invasive - coded as stand-alone modules.

dependencies: bigphysarea (if you want to get usable dynamic traces out of this)

1.2. Patch file

- `driver/char/kfi` - isolated kernel function instrumentation "driver"
- `kernel/sys.c` - a few `kfi_dump_log` inserted i
- `init/main.c` - this patch seems useless - all it does is add a call to a empty function - not clear what this is supposed to achieve (TODO: clarify).

patch applies clean against stock 2.4.20, fails uncritically against 2.4.26 (Makefile, and config.in - .rej files sufficient to patch) kfi dri ver is sufficiently isolated to patch into most likely any 2.4.X kernel.

Name test4 a bit irritating - there is no doc what test1-3 were about ...

1.3. Patch analysis

Generally not really invasive functionally - well isolated - should be trivial to port.

Patch does not seem to be really arch dependent - currently it seems configurable for arm, mips and X86 - but from the structure of the patch it should be fairly simply to move on to any other platform. Hard-coded CPU frequency (again) - bas idea - you need to compile for a specific platform (that at least could be passed as a config option...)

Times are read with low level (hardware dependent but fast) calls to arch specific clocks (i.e. rdtsc on x86) - rough math to stay in 32bit provided (no `do_gettimeofday` or the like) so this clock is available at a very early stage in the boot process. The records are based on deltas not absolute values (simplifies things as there are no potential overflow issues).

Initial entry is set to 0 so the first value is garbage. This could (should) be trivially fixed by having a hard-coded read of the tsc in `start_kernel` (would help a bit to get boot-times).

Patch does not record any pre- `start_kernel` events (bad) no info on decompression times.

Patch does not personalize the kfi patched kernel - potential collision of modules with unpatched 2.4.20 - `EXTRAVERSION=-kfi` set.

We strongly advice using this at runtime only if core functions are marked with `no_instrument_functions` - brute force instrumentation of each and every function is obviously a performance problem - and actually not that useful anyway. For boot-time analysis it seems reasonable the way static setup is provided (from `start_kernel` to just before `execve /sbin/init`).

The `profile_func_enter/exit` seems quite heavy weight - might be worth stripping this to the bare minimum possible (something like func,caller,timestamp ??- I admittedly don't know exactly what the current implementation does :).

The log entries are quite heavy weighted - the distinction between kernel, interrupt and PID context is quite useful - its a bit irritating that kernel threads are not marked any different than processes (TODO: look into adding this in `kfi.c` and in the `kfiresolve.py`).

1.4. Basic technology

Function instrumentation is a feature of `gcc` - by compiling applications with the `-finstrument-function` flag each call is preceded and followed by a call to a profiling function.

A simple example of a modified hello world should make this clear easily.

```
#include <stdio.h>

void  __attribute__((__no_instrument_function__))
__cyg_profile_func_enter(void *this_fn, void *call_site)
{
        printf("func_enter: function = %p, called by = %p\n",
                this_fn,
                call_site);
}

void __attribute__((__no_instrument_function__))
__cyg_profile_func_exit(void *this_fn, void *call_site)
{
        printf("func_exit: function = %p, called by = %p\n",
                this_fn,
                call_site);
}

main(){
   printf("hello world\n");
   return 0;
}
```

Compiled with `gcc -finstrument-functions hello.c -o hello` and run as `./hello` we get the output:

```
func_enter: function = 0x8048420, called by = 0x40041936
hello world
func_exit: function = 0x8048420, called by = 0x40041936
```

The instrumentation is done at compile time the normal code:

```
    subl    $12, %esp
    pushl   $.LC0
    call    printf
    addl    $16, %esp
```

is surrounded by calls to the profiling enter and exit functions passing the pointer of the calles (`$main`) and the function being called.

```
        subl    $8, %esp
        pushl   4(%ebp)
        pushl   $main
        call    __cyg_profile_func_enter
        addl    $16, %esp
    subl    $12, %esp
    pushl   $.LC2
    call    printf
    addl    $16, %esp
        movl    $0, %ebx
        subl    $8, %esp
        pushl   4(%ebp)
        pushl   $main
        call    __cyg_profile_func_exit
        addl    $16, %esp
```

Function instrumentation not only is available by coding it directly in the source files, which would be kind of inconvenient, but also can be wrapped up in a library. An example of a library that will produce the same output format for traces that the kernel space implemtation is giving (and thus allows to correlate events in kernel and user-space by sorting time-stamps) is given here as example:

```
/*
 * Compile as shared library with:
 * gcc -fPIC -Wall -g -O2 -shared  -o libfunc_profile.so.0 libfunc_profile.c
 *
 * Log all function calls to to the logfile in /tmp/func.log (default)
 * the format is chosen to allow usage of kfiresolver.py to reverse the log
 * entriew
 */
```

```
#include <stdio.h>     /* fprintf */
#include <unistd.h>    /* exit , getpid*/
#include <sys/types.h> /* getpid */
#include <stdlib.h>    /* getenv */

#define _FCNTL_H
#include <bits/fcntl.h>

/* initialize and cleanup logfile(s) on load/unload of lib */
void __func_profile_init(void) __attribute((constructor));
void __func_profile_exit(void) __attribute((destructor));

long long int start,last,now;
FILE *logfile;
char default_fname[]="/tmp/func.log";
char *logfile_name;

__inline__ unsigned long long int hwtime(void)
{
    unsigned long long int x;
    __asm__ __volatile__("rdtsc\n\t"
        :"=A" (x));
    return x;
}

void  __attribute__((__no_instrument_function__))
__func_profile_init(void)
{
   if ((logfile_name = getenv("PROFILE_LOG")) != 0) {
      printf("using %s\n",logfile_name);
    } else {
        logfile_name=default_fname;
      printf("using %s (no PROFILE_LOG set in environment)\n",logfile_name);
   }

    if((logfile=fopen(logfile_name,"a+")) == NULL )
    {
        perror("Cannot open logfile\n");
        exit(-1);
    }
```

```
    /* logfile header */
    fprintf(logfile,
    " Entry      Delta       PID      Function    Caller\n");
    fprintf(logfile,
     "--------   --------   --------   --------   --------\n");

    /* initialize time stamp */
    start=hwtime();
    last=start;
}

void  __attribute__((__no_instrument_function__))
__func_profile_exit(void)
{
    fclose(logfile);
}

void  __attribute__((__no_instrument_function__))
__cyg_profile_func_enter(void *this_fn, void *call_site)
{
    unsigned long long delta;
    pid_t pid=getpid();
    delta=0LL;

    now=hwtime();
    delta=now-last;
    last=now;

    fprintf(logfile, "%8lu   %8lu   %7d    %08x   %08x\n",
        (unsigned long)(now-start),
        (unsigned long)delta,
        pid,
        (unsigned int)this_fn,
        (unsigned int)call_site);
}

void __attribute__((__no_instrument_function__))
__cyg_profile_func_exit(void *this_fn, void *call_site)
{
    unsigned long long delta;
    pid_t pid=getpid();
```

```
    delta=0LL;

    now=hwtime();
    delta=now-last;
    last=now;

    fprintf(logfile, "%8lu   %8lu   %7d    %08x   %08x\n",
        (unsigned long)(now-start),
        (unsigned long)delta,
        pid,
        (unsigned int)this_fn,
        (unsigned int)call_site);
}
```

With such a libary it suffices to recompile user space applications with -finstrumetn-function -lfunc_profile.so.

Basically the same scheme is implemented in `driver/char/kfi.c` writing it into a log buffer. A kernel specific problem that needs to be resolved is that of inline functions. The passing of the address of the caller and the called function is done even for inline functions, resulting in taking the address of the inline functions which is not accessed via call. For functions declared `extern inline` this results in `undefined symbol` errors. For `static inline` it causes a static version to be compiled in each object file that uses the inline function. So to allow function instrumentation all inline functions are treated as static inline functions - this slows down things even more than would be done just by the overhead of logging data, but this kernel modification is not intended for systems that require high performance.

1.5. Installation

Installation of the official patches is more or less broken - patches apply cleanly but compilation fails without quite heavy modifications in the low level code (pre vmlinux stuff).

```
tar -xjf linux-2.4.20.tar.bz2 (kernel.org)
cd linux-2.4.20
patch -p1 < ../kfi-24-test4.patch
make menuconfig

Kernel hacking ---->
  [ } Kernel debugging
  [*] Kernel Function Instrumentation (NEW)
  [*] Static Instrumentation Config
```

Note: there is no help available for both of the new options - so here is a minimum summary of what this does.

Kernel Function Instrumentation

This basically enables kfi (builds the `driver/char/kfi` and inserts `kfi_dump_log` points.

Static Instrumentation Config

This sets up a linked list as automatic variable instead of dynamically allocating it further more if this option is enable a empty `to_userspace()` function is called before calling `init` - this is used by the static instrumentation config to locate the point where instrumentation should be turned off (so called trigger).

For static instrumentation there is a config file in `drivers/char/kfistatic.conf` that allows setting of instrumentation parameters at compile time (would be nice to have this in the kernel config menu !)

Make procedure of the unmodified patch `2.4.20-test4`:

```
make dep
make   (fails due to scripts/mkkfirun.pl being mode 644 not 755)
chmod 755 scripts/mkkfirun.pl
make   (fails in drivers/ide/ide-cd.h line 440 type : __u8 short -> __u8)
make   (compilation completed - further warnings ignored ;)
make modules
make modules_install
make bzImage
```

(fails with undefined references to `__cyg_*...` looks like instrumentation is not implemented correctly - the scope of the entry/exit functions is limited to `vmlinux` (kernel proper) but the low level stuff (`/arch/i386/boot/misc.c lib/inflate.c` referenced in `misc.c`) was also compiled with `-finstrument-functions`

Reconfigured without static instrumentation (which would makes it quite unusable for boot-time analysis though) same problem.

Unresolved symbols in `misc.c` - guess `misc.c` should NOT be compiled with `-finstrumentation`... (TODO: need to clarify if this EVER worked with a vanilla 2.4.20 kernel) - workaround turn it off on a per function basis using `__noinstrument` (pain in the but - as you have to recompile the kernel from scratch - which then yields a new set of unresolved symbols ;) .

Files cleaned: `lib/inflate.c arch/i386/boot/compress/misc.c` - basically all function calls got instrumentation turned off - which is not tragic as `misc.c` is pre `start_kernel` any way and `inflate.c` functions are not called during the system initialization.

1.6. Data acquisition

Once it compiles and installs the README in the `kfi-0.8 tools` (`README.kfi`) should be sufficient. The only requirement is phyon -but as data resolution from addresses to names can be done off-line (only need the `System.map` of the profiled system to do it) - python is no issue.

```
tar -xuf kfi-0.8.tar.gz
cd kfi-0.8
mknod /dev/kfi c 10 51
make
./kfi read 0 > kfiboot.log
vi kfiboot.log
./kfiresolve.py ./kfiboot.log \
   ../linux-2.4.20-kfi/System.map  > kfiboot.lst
vi kfiboot.lst
```

Note that in dynamic mode you can only use the read and reset command - all other command will give you `IOCTL` errors - for dynamic instrumentation logs see the next section. The strange limit of 8092 bytes for static logs seems to stem from the log being on the kernel stack (ugh!)- TODO: fix that .

1.7. Dynamic Data acquisition (post boot)

When compiled without static setup it does not work (at least no out of the box) the command `new` to `kfi` will aboard with an `EINVAL` in `IOCTL (NEW_RUN)`. Dynamic acquisition looks like its not quite completed yet - the problem with the distributed version is that there are two header files that both define `MAX_RUN_LOG_ENTRIES` - in kernel space its defined to be 8092 (not 8192) and in user-space `kfi.h` its set to 20000 - in the `IOCTL` command switch for `NEW_RUN` the passed entry value is checked against kernel side `MAX_RUN_LOG_ENTRIES` and not to surprising exits with `-EINVAL`. If set larger than about 6000 (that is clearly below the default 8092 !) it fails due to `kmalloc` failing (TODO: check and if necessary move to `vmalloc`)

Once that is fixed it actually kind of works (a bit;)

Hard-code what you want to see in `kfi.h` (user-space) - the `MAX_RUN_LOG_EVENTS` must be smaller than the value set in `include/linux/kfi.h` !

```
make
./kfi reset
./kfi new
new run created, id = 0
./kfi start
runid 0 started
./kfi stop (looks like this is broken !)
STOP ioctl error: Invalid argument
./kfi read > log
-rw-r--r--  1 root root 217441 2005-01-01 16:56 log
./kfi status
```

Kernel Instrumentation Run ID 0

Logging started at 963768895 usec by system call Logging stopped at 963769264 usec by log full

Filters:

Filter Counters: Total entries filtered = 0 Entries not found = 1

Number of entries after filters = 4096

```
./kfi reset
./kfiresolve.py log /usr/src/linux-2.4.20-kfi/System.map > lst
```

`lst` contains the "call graph" of the booting kernel with timestamps now. The header of the list file is the same as you would get with the `./kfi` status command.

Kernel Instrumentation Run ID 0

Logging started at 111102362 usec by system call Logging stopped at 114108365 usec by log full

Filters:

Filter Counters: Total entries filtered = 0 Entries not found = 16

Number of entries after filters = 32768

Entry	Delta	PID	Function	Called At
0	no exit	251	fput	sys_ioctl+0x87
1	no exit	251	do_page_fault	error_code+0x34

```
       1    no exit     251                  find_vma   do_page_fault+0x99
       1    no exit     251           handle_mm_fault   do_page_fault+0x198
       1    no exit     251                 pte_alloc   handle_mm_fault+0x54
       1    no exit     251               do_no_page    handle_mm_fault+0x7f
       1    no exit     251            filemap_nopage   do_no_page+0xa8
       1    no exit     251           __find_get_page   filemap_nopage+0xe4
```

The `no exit` in the Delta field is due to the delta recorded being 0 - this is the case when the resolution of the `kfi_readclock()` is higher than the runtime of the function being traced (TODO: check up on this - looks strange considering they are using the TSC).

```
driver/char/kfi.c:
static inline unsigned long __noinstrument
update_usecs_since_boot(void)
{
        unsigned long machine_cycles, delta;

        machine_cycles = kfi_readclock();
        delta = machine_cycles - last_machine_cycles;
        delta = kfi_clock_to_usecs(delta);

        usecs_since_boot += delta;

        last_machine_cycles = machine_cycles;
        return usecs_since_boot;
}
```

To help interpret data there is a tool in the `kfi-0.8.tar.gz` archive that allows filtering the `lst` data:

```
./kd -h
usage: kd [<options>] <filename>
```

This program parses the output from a set of kfi message lines

```
Options:
  -h            Show this usage help.
  -c <count>    Only show the <count> most time-consuming functions
  -t <time>     Only show functions with time greater than <time>
```

```
-f <format>   Show columns indicated by <format> string.  Column IDs
              are single characters, with the following meaning:
                F = Function name
                c = Count (number of times function was called)
                t = Time (total time spent in this function)
                a = Average (average time per function call)
                r = Range (minimum and maximum times for a single call)
                s = Sub-time (time spent in sub-routines)
                l = Local time (time not spent in sub-routines)
                m = Max sub-routine (name of sub-routine with max time)
                n = Max sub-routine count (# of times max sub-routine
                    was called)
                u = Sub-routine list (this feature is experimental)
              The default column format string is "Fctal"
-l            Show long listing (default format string is "Fctalsmn")
```

I.e. `./kd -c 10 lst` will show the 10 most heavy weight functions being called in the trace.

Function	Count	Time	Average	Local
schedule	278	6010296	21619	6010296
schedule_timeout	139	6008929	43229	45
sys_select	169	3005012	17781	26
do_select	169	3004985	17780	36
sys_read	140	3004086	21457	21
tty_read	129	3004050	23287	21
read_chan	129	3004029	23287	30
default_idle	421	3003589	7134	3003589
do_IRQ	429	2657	6	0
handle_IRQ_event	429	1654	3	0

kmalloc switch to bigphysarea:

To improve dynamic tracing kfi was modified to use bigphysarea instead of kmalloc (which is limited to the infamous 128k) - unfortunately this does not allow tracing from `start_kernel` on bug the startup must be done later:

```
      0    no exit       0               start_kernel   L6+0x0
      1       9420       0                 setup_arch   start_kernel+0x30
    208       9110       0                paging_init   setup_arch+0x1cc
```

```
  222       9096       0            zone_sizes_init   paging_init+0x42
  222       9096       0             free_area_init   zone_sizes_init+0x43
  222       9096       0         free_area_init_core   free_area_init+0x4f
  222       4474       0        __alloc_bootmem_node   free_area_init_core+0x3b7
  222       4474       0        __alloc_bootmem_core   __alloc_bootmem_node+0x49
 9421      14609       0               parse_options   start_kernel+0x51
 9424      14606       0                  checksetup   parse_options+0x18f
 9424      14606       0           bigphysarea_setup   checksetup+0xaa
 9424      14605       0             __alloc_bootmem   bigphysarea_setup+0x79
 9424      14605       0        __alloc_bootmem_core   __alloc_bootmem+0x4e
24067     377850       0                   time_init   start_kernel+0x6a
```

Until after `__alloc_bootmem_core` `bigphysarea_malloc` will cause a system lockup/reboot - if bigphysarea should be used to get post-init traces the trigger point would need to be set to `timer_init()` instead of `boot_kernel`.

For dynamic traces the limit of the log length is limited by physical ram that can be allocated to bigphysarea via kernel parameter `bigphysarea=NUMBER_OF_PAGES`, note that the trace structure entries are 256bytes so 1024 pages would result in a limit of 131072 (128k entries).

1.8. Data interpretation

see `kfiboot.log` and `kfiboot.lst` - basically simply run down the list and search for functions that take a long time - there also are structural issues one can find - like heavy invocation of delays. TODO: cleanup.

1.9. Performance Impact

The performance impact of kfi is substantial - it is a diagnostic tool and no runtime debug tool - insofar it is inferior to tools like `ltt`- but for boot-time issues it is superior to `instrumented` `printk` or `ltt` (the later launches much to late to be of much use for boot-time issues).

Performance impact is run by comparing `lmbench` on an unpatched 2.4.20 kernel with kfi being used at runtime (not boot-time - basically because we can't run `lmbench` at boot time).

2. Conclusion 2.4.X

KFI is a tool usable for test-runs - its absolute values are not that usable but the relative values are very usable. The granularity of the results is sufficient to pinpoint potential optimizations very quickly.

A drawback is that it does not trace the low level initialization code before `start_kernel` (this does not seem to be a config issue - but due to the low level stuff requiring to be compiled without instrumentation (at least we did not manage to do it without `__noinstrument` in the low level stuff)).

The state of the project is not yet stable - documentation is still incomplete and the patches don't seem to be clean (yet) - the project is of interest but only can be recommended for use if a local technician sufficiently understands the technology to actually maintain it.

All though only arm,mips and x86 are supported porting to further arch should be quite simply as the core of kfi is well encapsulated in `drivers/char/kfi.c` .

A clear disadvantage at this point is that the trace configuration for boot-time traces is statically configured and requires recompiling when changed.

KFIs advantage over i.e. ltt is that it is non-invasive - the driver is well encapsulated and the function tracing is done via gcc's build in instrumentation capabilities - thus adding and expanding to new kernel routines or custom drivers is trivial - furthermore the trace is flat with respect to covered functions (if one excludes the issues of inline ed funcs) as compared to ltt where the trace points are designed by the developers (although ltt does allow adding custom trace points - its just not automatic). The clear drawback is its impact on the system - kfi is an analysis tool but no runtime-debug tool like ltt.

kfi is a non-standard tool, although standardization is not a mandatory feature it would help with comparing results from other tools.

It also should be noted that the development is still in an early stage and that currently the risk of relying on kfi may be fairly high - if a project wishes to utilize kfi it is recommended that at least one team member actually work into it to a level where a self-sustained maintenance and continuation is possible.

Dynamic tracing is close to being called broken - but it should not be too hard to fix it (see above).

KFI is a very interesting technology and hopefully will be continued - its current state is usable with a bit of tuning but it is currently at best beta.

3. KFI for 2.6.X

KFI has undergone major changes and restructuring for the 2.6.X series of kernels, the concepts and the core technology is the same though, so we don't repeat that in this 2.6.X section, refer to the above 2.4.X sections for an introduction to the technological concepts of KFI.

dependencies: bigphysarea (if you want to get usable dynamic traces out of this)

3.1. Patch file

kfi moved from a pseudo-driver in driver/char/kfi.c to a pure kernel function set implemented kernel/kfi.c. The implementation is not arch dependent and does not modify any kernel code conceptually, though off course it is very invasive at runtime as every function is preceded/followed by the `__cyg_profile_func_enter/exit` calls.

- `kernel/kfi.c`;
  - Time management functions:
    All low level time management functions, for reading of time and converting to microseconds, the time stamp precision is unfortunately not platform/arch independant and `kfi.c` comes with an ugly platform sepcific define:

    ```
    #define CLOCK_FREQ 1602319000ULL
    ```

    - this really should be a config option or calculated from existing values, not hard coded this way, if set incorectly the reported times are obviously garbage.
  - run list management:
    Functions to manage run lists entries, conditional logging etc.
  - the actual instrumentation function:
    in kfi for 2.6 __cyg_profile_func_enter/exit both map to the same function `func_entry_exit` which is the actual instrumentation function.
  - proc related functions:
    All the functions to setup and remove the proc entries and to read from them are in here aswell. This include functions to parse setting, provided by echoing into `/proc/kfi` and formating/dumping functions to output the data aquired by `cat /proc/kfi_trace`

- `fs/proc/proc_misc.c`:
  Not quite clear why but the `kfi_trace` creation is not done in `kfi.c`but rather in `fs/proc/proc_misc.c` though all operations are from `kfi.c` - this might need moving ?

- `include/linux/kfi.h`:
  The main structures for kfi are in here:

  - struct kfi_run:
    This represents a runtime probe
  - struct kfi_filter:
    The filter conditions, basically time and context
  - struct kfi_trigger:
    the trigger conditions description

Slight inconsistencies in the naming, kfi-26-test1.patch is kfi-2.6.7.patch, kfi-2.patch is currently kfi-2.6.11.patch it is not quite clear to us if kfi-2.patch is the "current" patch file or not. You probably only can unpack the patch and open it with an editor to know for exactly what version this is.

Note that KFI in the 2.6.X patches is only available for X86, limiting is usability as a general tool seriously - though it is usable to support analysis of the arch independent kernel parts (which is about 95

3.2. Building 2.6.11 with KFI

```
root@rtl17:/usr/src# tar -xjf linux-2.6.11.tar.bz2
root@rtl17:/usr/src# mv linux-2.6.14 linux-2.6.11-kfi
root@rtl17:/usr/src# mv kfi-2.patch kfi-2.6.11.patch
root@rtl17:/usr/src# cd linux-2.6.11-kfi
root@rtl17:/usr/src/linux-2.6.11-kfi#
root@rtl17:/usr/src/linux-2.6.11-kfi# patch -p1 --dry-run < ../kfi-2.6.11.patch
```

The patch seems to not be against vanilla 2.6.11 - thus quite a lot of fuzz and offset with this patch - none of this seems to be functionally critical though.

```
Hunk #1 succeeded at 115 (offset 10 lines).
Hunk #1 succeeded at 75 (offset -4 lines).
Hunk #1 succeeded at 69 (offset 23 lines).
Hunk #1 succeeded at 25 with fuzz 2 (offset -1 lines).
Hunk #1 succeeded at 2626 (offset -1 lines).
```

3.2.1. kernel config

```
root@rtl17:/usr/src/linux-2.6.11-kfi# make menuconfig
```

```
  Kernel hacking  --->
    [ ] Kernel debugging
    [*] Debug preemptible kernel (NEW)
    [ ] Compile the kernel with frame pointers
    [*] Use 4Kb for kernel stacks instead of 8Kb
    [ ]   Static Instrumentation Configs (NEW)
    (0)   Scaling factor for early initialization of kfi clock (NEW)
    [*] Kernel Function Instrumentation
```

note that `Debug preemptible kernel` and `Use 4Kb for kernel stacks instead of 8Kb` are on by default and are not related to the usage of KFI.

After creating a new configuration we recommend copying your `.config` to a meaningful name i.e. `cp .config config_2.6.11_kfi`, and add it to your source management system.

3.2.2. Configuration bug

The configuration has a hard-coded clock value, which is used by `kfi_clock_to_usecs` in `kernel/kfi.c` - this needs to be adjusted to what ever you find in `/proc/cpuinfo` - this can be considered a bug in KFI, either this has to use available kernel parameters (i.e. cpu_khz) or must be a mandatory config option - no idea why this is hard-coded here...

```
#define CLOCK_FREQ 400000000ULL
```

On the test-system, a 1.6GHz AMD Duron this value was set to

```
#define CLOCK_FREQ 1602319000ULL
```

according to what was found in `/proc/cpuinfo`

```
cpu MHz         : 1602.319
```

Note that if this is NOT fixed then your results are complete garbage.

3.2.3. Compiling and Installing

Compiling and installing is the default procedure for Linux kernels.

```
root@rtl17:/usr/src/linux-2.6.11-kfi# make bzImage
...
kernel/kfi.c: In function ‘kfi_new_run’:
kernel/kfi.c:1015: warning: comparison of distinct pointer types lacks a cast
  LD      kernel/built-in.o
  CC      mm/bootmem.o
...
Root device is (3, 2)
Boot sector 512 bytes.
Setup is 2794 bytes.
System is 1889 kB
Kernel: arch/i386/boot/bzImage is ready
```

Note that there are still a few glitches in the current 2.6.X KFI sources - though none of these seem problematic as of 2.6.11.

```
root@rtl17:/usr/src/linux-2.6.11-kfi# make modules
...
root@rtl17:/usr/src/linux-2.6.11-kfi# make modules_install
...
  INSTALL net/ipv4/netfilter/ipt_NOTRACK.ko
  INSTALL net/ipv4/netfilter/iptable_raw.ko
if [ -r System.map ]; then /sbin/depmod -ae -F System.map  2.6.11; fi
```

Basically all except the last stage can be run as non-root user - only the last step will require root privileges - so if you did all this as non-root user you need to become root now.

```
root@rtl17:/usr/src/linux-2.6.11-kfi# cp arch/i386/boot/bzImage \
/boot/2611kfi
```

Lilo Config

After copying the image to `/boot` we add the new kernel to the `lilo.conf`. Note that the paths shown here are not mandatory and they may be different on your distribution. For details see `man 5 lilo.conf`.

```
root@rtl17:/usr/src/linux-2.6.11-kfi# cd /etc
root@rtl17:/etc# vi lilo.conf
```

Add the following lines to your lilo.conf - note that you off course must adjust your `root=` setting to fit your system.

```
image = /boot/2611kfi
  root = /dev/hda2
  label = 2611kfi
  read-only
```

Don't forget to run `lilo` - you should get something like:

```
Added Linux *
Added 2611kfi
```

To reboot into the new kernel you can select it at the Lilo prompt or use lilo's one-time selection like so:

```
root@rtl17:/etc# lilo -R 2611kfi
root@rtl17:/etc# reboot
```

GRUB Config

To boot the new kernel with grub add the following line to your `menu.lst`. On many distributions you can find it in `/boot/grub`, though this is not mandatory, thus don't be surprised if you don't find it there.

```
title  2611kfi
kernel (hd0,1)/boot/2611kfi root=/dev/hda2 read-only
```

Note that grub starts counting partitions at 0 thus `/dev/hda2` maps to `hd0,1`. As grub knows how to read filesystems you don't need to reinstall grub, but simply reboot after adding the above entry and select it at the boot-prompt - grub does not have a `lilo -R target` alike command, once you find your new kernel is ok you can set the default boot to the target to boot, for details we refer you to the man pages of grub.

3.3. Data acquisition

As the documentation is badly out of sync we have written up our procedure here - this is not in any way official though - so it might be incomplete or partially incorrect - We did not want to spend more time with source code review to get this working...

3.3.1. What not to do in 2.6.X

Following the instructions on the web-page and the README.kfi from `kfi-0.8.tar.gz` you would find the following procedure:

```
root@rtl17:~# mkdir kfi
root@rtl17:~# cd kfi/
root@rtl17:~/kfi# tar -xzf kfi-0.8.tar.gz
root@rtl17:~/kfi# cd kfi-0.8
root@rtl17:~/kfi/kfi-0.8# make
```

Runs smoothly and creates the kfi user space application which is responsible for controlling KFIs kernel internal actions by a set of commands issued via ioctl's on `/dev/kfi`.

```
root@rtl17:~/kfi/kfi-0.8# mknod /dev/kfi c 10 51
```

The README tells you to dump the data with

```
root@rtl17:~/kfi/kfi-0.8# ./kfi read 0 > log
```

Which only will give you:

```
Error opening KFI device: No such device
```

The sequence we found to work, after reading through kernel/kfi.c, as documentation is totally out of sync... was to setup and configure the traces via the proc interface only.

- `/proc/kfi`:
  This is the control interface - which replaces what was done via ioctls in the 2.4.X KFI implementation (and provided by the `kfi-0.8.tar.gz` package).
- `/proc/kfi_trace`:
  Access point to the trace buffer. After configuring a trace and stopping it you can dump the buffer via `/proc/kfi_trace`.

3.3.2. Configuring KFI at runtime

KFIs runtime configuration is done by runtime parsing of commands written to /proc/kfi - the control "port" of KFI. The commands are a series of token value pairs terminated by the end token that stands alone, and all of it cat'ed into /proc/kfi. As a simple example you could do:

```
root@rtl17:~/kfi# echo "new logsize 4000 end" > /proc/kfi
root@rtl17:~/kfi# echo start > /proc/kfi
root@rtl17:~/kfi# echo stop > /proc/kfi
root@rtl17:~/kfi# cat /proc/kfi_trace > trace.data
```

This is the raw trace data that will look something like:

```
Kernel Instrumentation Run ID 0

Logging started at 1216155378 usec by user action
Logging stopped at 1216234815 usec by log full

Filter Counters:
Total entries filtered = 0
Entries not found = 57

Number of entries after filters = 4000


 Entry    Delta     PID        Function                  Caller
-------- -------- -------- ----------------          ------------
1216155377        1    1362  0xc010ff40                  0xc014352a
       3        1    1362  0xc01a3900                0xc016ae85
       4        0    1362  0xc0116b10                0xc01a39c0
      10       31    1362  0xc0180f30                0xc0103927
      10        1    1362  0xc018d2e0                0xc0180fce
...
```

The mess-up of the columns settings is due to the TSC on this box running a bit too fast I guess - KFI might need some format cleanups...

The run is 4000 entries long - which is what we configured - note that there is a hard coded limit of MAX_RUN_LOG_ENTRIES which is at 10000 by default (configured in include/linux/kfi.h).

The following KFI commands are available (as of 2.6.11):

- prime: start as soon as the trigger start condition is hit
- start: start unconditionally
- stop: stop unconditionally
- new: setup new run

3.4. Data interpretation

The tools from the `kfi-0.8.tar.gz` that we compiled, are not needed for the 2.6.X version of KFI - what is needed is the decoder to reverse-map the addresses to the corresponding function names, which is though provided within the kernel patch and ends up as `scripts/addr2sym` in the linux source tree.

3.4.1. addr2sym

This phyton script basically builds a table of symbols from the `System.map` and uses the hex addresses found there to assign the recorded entry/exit addressses to the respective function.

`addr2sym` will first try to do an exact match of each symbol, basically the call entry and exit points should be well known addrsses in the System.map, for those that can't be found a neerest match is done - we assume this means function names listed may be a bit off in some cases, but have not verified this. If addresses show up in the output as unconverted values then they were out of range, obviously `addr2sym` can't produce any better results in such a case.

As a gereral note, the `addr2sym` tool is trying to prvoide functionality that basically exists in the 2.6.X series of kernel, it might be better to rely on thos tools (i.e. print_symbol() and frinds).

3.4.2. Data Acquisition 1st shot

```
root@rtl17:/usr/src/linux-2.6.11-kfi # chmod 755 scripts/addr2sym
root@rtl17:/usr/src/linux-2.6.11-kfi # scripts/addr2sym /tmp/kfi.data \
-m System.map > /tmp/kfi.lst
```

For kernels that use modules it can result in incomple results, in that case you want to use /proc/kallsysm instead of System.map

```
root@rtl17:/usr/src/linux-2.6.11-kfi # scripts/addr2sym /tmp/kfi.data \
-m System.map > /tmp/kfi.lst
```

This will produce the `kfi.lst` file with the same format as the old tool, just that the `no_exit` is now listed by putting a 0 in the delta field.

```
Kernel Instrumentation Run ID 0

Logging started at 1216155378 usec by user action
Logging stopped at 1216234815 usec by log full

Filter Counters:
Total entries filtered = 0
Entries not found = 57

Number of entries after filters = 4000


 Entry    Delta     PID        Function                    Caller
-------- -------- -------- ----------------             ------------
16155377        1    1362  sched_clock                  kfi_start+0x5a
       3        1    1362  dnotify_parent               vfs_write+0xe5
       4        0    1362  sub_preempt_count            dnotify_parent+0xc0
      10       31    1362  sys_dup2                     syscall_call+0x7
      10        1    1362  expand_files                 sys_dup2+0x9e
...
```

A note on the "Entries not found = " line, the problem with staring a trace in the middle of a running system is obviously that you can get an exit from a funciton without ever having seen an entry, KFI will flag this by marking the delta as 0 and incrementing `run>notfound` to indicate this. So one must be carfull with interpreting data, generally we would recommend not trying to interprete a single run at all, but to make atleast three runs starting at a specific trigger point and then try to interprete data once one has reproducible snapshots, this will generally imply very short "shutter times", that is trigger entry and trigger stop will need to be relatively close.

3.4.3. kfiresolve.pl problems

It should be noted that kfiresolve.py/pl are not the intendet tools for the 2.6.X patches, although it would be preferable to reuse these tools and not to add new - functionally

equivalent tools in our opinion. For those that have them in use a few notes on the kfi tools from `kfi-0.8.tar.gz`.

```
root@rtl17:~/kfi# kfi-0.8/kfiresolve.pl  trace.data \
/usr/src/linux-2.6.11-kfi/System.map > trace.log
```

this runs without any errors but, to our surprise, without any usable results either - the trace.log file still displays hex addresses not function names - looks like this is broken - but we did not bother to follow up as we preferred the python based tool (see below).

3.4.4. kfiresolve.py problems

```
root@rtl17:~/kfi# kfi-0.8/kfiresolve.py  trace.data /usr/src/linux-2.6.11-kfi/System.map
Traceback (most recent call last):
  File "kfi-0.8/kfiresolve.py", line 141, in ?
    main()
  File "kfi-0.8/kfiresolve.py", line 134, in main
    caller = callername(funclist, calleraddr)
  File "kfi-0.8/kfiresolve.py", line 58, in callername
    addr = eval("0x"+addr_str+"L")
  File "<string>", line 1
    0x0xc014352aL
                ^
SyntaxError: unexpected EOF while parsing
```

The fix was at line 55 and line 130 of the provided kfiresolve.py:

```
--- kfiresolve.py       2006-02-08 14:00:13.000000000 +0100
+++ kfiresolve.py.org   2006-02-08 13:59:39.000000000 +0100
@@ -55,7 +55,7 @@
 # return string with function and offset for a given address
 def callername(funclist, addr_str):
        # convert address from string to number
-       addr = eval(addr_str)
+       addr = eval("0x"+addr_str+"L")

        # if address is outside range of addresses in the
        # map file, just return the address without converting it
@@ -130,7 +130,7 @@
```

```
                        pid = pid[:-1]
                        in_int = 1

-               func = callername(funclist, funcaddr)
+               func = funcname(funcmap, funcaddr)
                caller = callername(funclist, calleraddr)

                if delta=="0": delta="no exit"
```

We are not sure this "fix" in generally applicable - but we would be interested in hearing of any problems with this fix (or if anybody has the original working and if so with what distro/python version).

3.4.5. Data Acquisition 2nd shot

```
root@rtl17:~/kfi# kfi-0.8/kfiresolve.py  trace.data /usr/src/linux-2.6.11-kfi/System.map
```

This seems to be ok now and the results are reasonable - note that we are not reprinting the header here, as the header is redisplayed unchanged:

```
 Entry      Delta      PID            Function                Called At
--------   --------   -----   -------------------------   --------------------
16155377          1    1362             sched_clock+0x0   kfi_start+0x5a
       3          1    1362          dnotify_parent+0x0   vfs_write+0xe5
       4    no exit    1362       sub_preempt_count+0x0   dnotify_parent+0xc0
      10         31    1362                 sys_dup2+0x0   syscall_call+0x7
      10          1    1362             expand_files+0x0   sys_dup2+0x9e
      11    no exit    1362       sub_preempt_count+0x0   sys_dup2+0xd8
      11         29    1362                filp_close+0x0   sys_dup2+0xf7
      12    no exit    1362           dnotify_flush+0x0   filp_close+0x5a
```

3.4.6. field description

The fife fields in the decoded file have the following meanings.

- Entry:
  The number that this function had in the entry list - the entry list is also limited by a hard coded maximum of 512 entries `include/linux/kfi.hMAX_FUNC_LIST_ENTRIES`

- Delta:
  Delta is the time difference in `machin_cycles` aka TSC resolved to micro seconds by `kfi_clock_to_usecs()` in `kfi.c`, unfortunately 0 is a legal return value here considering that a microsecond is quite a lot of time for modern CPUs. Note that the "no exit" string that is printed in the delta simply means the delta was 0.
- PDI:
  The process ID of the process that was being traced, the kernel shows up with PID 0 and interrupt context shows up as PID -1.
- Function:
  The function that was called
- Called At:
  The function + offset at which the call happened.

3.4.7. Runtime Configuration

The first trace we did above left everything at the defaults, which is start now, trace everything, stop at the maximum logsize. To configure it to track specific events you need to configure KFI.

An example:

```
root@rtl17:/kfi# grep do_IRQ /proc/kallsyms
c01059d0 T do_IRQ
c0143d50 T __do_IRQ
root@rtl17:~/kfi# ocho "new \
        trigger start entry c0143d50  \
        trigger stop exit c0116280 \
        filter mintime 0 \
        logsize 4000 \
        end" > /proc/kfi
root@rtl17:~/kfi# dmesg
status: run id 2, not primed, not triggered, not complete

config:
  mode 1
  trigger start entry 0xC0143D50
  trigger stop exit 0xC0116280
  filter mintime 5
```

```
  filter maxtime 0
  logsize 4000
KFI: new kfi run installed
root@rtl17:~/kfi# echo prime > /proc/kfi
...<give it some time>
root@rtl17:~/kfi# echo stop > /proc/kfi
root@rtl17:~/kfi# cat /proc/kfi_trace > do_irq.data
root@rtl17:~/kfi# /usr/src/linux-2.6.11-kfi/scripts/addr2sym -m /proc/kallsyms  test2.dat
```

This sequence will record the program flow starting at __do_IRQ and after decoding result in a quite readable file.

```
Kernel Instrumentation Run ID 2

Logging started at 3620056640 usec by entry to function __do_IRQ
Logging stopped at 3620152644 usec by log full

Filter Counters:
Total entries filtered = 0
Entries not found = 40

Number of entries after filters = 4000


 Entry    Delta     PID        Function                    Caller
-------- -------- -------- ----------------             ------------
       1       45      -1i __do_IRQ                     do_IRQ+0x5c
       3        6      -1i mask_and_ack_8259A           __do_IRQ+0xb2
       7        1      -1i sub_preempt_count            mask_and_ack_8259A+0x71
...
```

The current filter configuration of the specific run id (in this case 5) will be dumped to the console with a very high log level - so you most likely get it smeared all over your current console (...), now to the details of the new command:

- trigger:
  trigger is the condition at which tracing should start or stop - trigger needs to be further configured with:

  - start:
    the start condition
  - stop:
    the stop condition

  both the start and the stop condition take the hex address of a kernel function , the functions that can be passed are all that are listed in the respective System.map file (generated in the top level kernel directory on kernel build). You can specify if the entry or the exit of the passed function is to trigger by passing the entry or exit modifier before the address of the function.

- filter:
  The filter directive allows setting of time filters - to reduce the amount of traced data one can tell KFI to only trace events that fulfill the specified timing criteria of
  - mintime:
    the minimum time in microseconds that the function must have taken
  - maxtime:
    the maximum time in microseconds that the function may have taken to execute - not really sure what this maxtime would be good for - but its configurable.
- end the end keyword tells the parser to stop looking for tokens in the new command.

So the above command , with the following mappings from the System.map file:

```
c01167f0 T scheduler_tick
c0116280 T schedule_tail
```

give a tracer starting at `scheduler_tick` (0xc01167f0) and ending at `schedule_tail` c0116280, recording all functions that run at least for 5 microseconds, and logging 4000 entries.

Note on mode filed The following mode fields are supported, which are passed as flags value internally.

```
#define KFI_MODE_TIMED          0x01
#define KFI_MODE_AUTO_REPEAT    0x02
#define KFI_MODE_STOP_ON_FULL   0x04
```

A further mode flag noted in a comment `KFI_MODE_OVERWRITE` does not seem to exist any more (if anyone can clarify this - drop me a note pleas). We also did not find any configuration option to set the KFI mode - so this seems to be "work in progress".

3.4.8. Performance Impact

The performance impact of kfi - even without any active run - is so high that it hardly is suitable for production systems.

```
            L M B E N C H  3 . 0   S U M M A R Y
            ------------------------------------
            (Alpha software, do not distribute)

Basic system parameters
------------------------------------------------------------------------
           OS Description              Mhz  tlb  cache  mem   scal
                                           pages line   par   load
                                                 bytes
------------- ----------------------- ---- ----- ----- ------ ----
Linux 2.6.14S       i686-pc-linux-gnu 1599    32    64 2.7700    1
Linux 2.6.14S       i686-pc-linux-gnu 1599    32    64 2.7500    1
Linux 2.6.14S       i686-pc-linux-gnu 1599    32    64 2.7000    1
 Linux 2.6.11                   kfi 1599    32    64 2.8200    1
 Linux 2.6.11                   kfi 1599    32    64 2.7800    1
 Linux 2.6.11                   kfi 1599    32    64 2.7500    1

Processor, Processes - times in microseconds - smaller is better
------------------------------------------------------------------------
           OS  Mhz null null      open slct sig  sig  fork exec sh
                   call  I/O stat clos TCP  inst hndl proc proc proc
------------- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
Linux 2.6.14S 1599 0.19 0.55 3.24 4.84 8.80 0.49 1.98 183. 1200 9523
Linux 2.6.14S 1599 0.19 0.47 3.18 4.80 22.1 0.49 2.04 174. 1106 9070
Linux 2.6.14S 1599 0.19 0.38 3.12 4.50 18.4 0.49 1.97 163. 1064 8972
 Linux 2.6.11 1599 0.46 2.41 22.3 29.9 176. 2.48 17.2 593. 2554 14.K
 Linux 2.6.11 1599 0.46 2.53 22.8 31.4 164. 2.47 17.5 624. 2631 15.K
 Linux 2.6.11 1599 0.46 2.58 22.8 31.5 164. 2.47 17.5 614. 2591 14.K

Basic integer operations - times in nanoseconds - smaller is better
------------------------------------------------------------------------
           OS  intgr intgr  intgr  intgr  intgr
                bit   add    mul    div    mod
------------- ------ ------ ------ ------ ------
Linux 2.6.14S 0.6300 0.6300 2.5100   25.7   26.9
```

```
Linux 2.6.14S 0.6300 0.6300 2.5000   25.7   26.9
Linux 2.6.14S 0.6300 0.6300 2.5100   25.7   26.9
 Linux 2.6.11 0.6400 0.6400 2.5700   26.3   27.5
 Linux 2.6.11 0.6400 0.6400 2.5500   26.0   27.3
 Linux 2.6.11 0.6400 0.6300 2.5400   26.0   27.3

Basic float operations - times in nanoseconds - smaller is better
-----------------------------------------------------------------
           OS  float  float  float  float
               add    mul    div    bogo
------------ ------ ------ ------ ------
Linux 2.6.14S 2.5000 2.5000   10.9 6.2800
Linux 2.6.14S 2.5000 2.5100   10.9 6.2800
Linux 2.6.14S 2.5100 2.5000   10.9 6.2800
 Linux 2.6.11 2.5500 2.5500   11.2 6.4200
 Linux 2.6.11 2.5500 2.5500   11.1 6.3800
 Linux 2.6.11 2.5500 2.5500   11.1 6.3800

Basic double operations - times in nanoseconds - smaller is better
------------------------------------------------------------------
           OS  double double double double
               add    mul    div    bogo
------------ ------  ------ ------ ------
Linux 2.6.14S 2.5000 2.5000   11.0 5.6600
Linux 2.6.14S 2.5000 2.5100   11.0 5.5900
Linux 2.6.14S 2.5100 2.5000   10.9 5.5700
 Linux 2.6.11 2.5500 2.5500   11.2 5.6300
 Linux 2.6.11 2.5400 2.5400   11.1 5.5800
 Linux 2.6.11 2.5400 2.5400   11.1 5.5900

Context switching - times in microseconds - smaller is better
-------------------------------------------------------------------------
           OS  2p/0K 2p/16K 2p/64K 8p/16K 8p/64K 16p/16K 16p/64K
               ctxsw  ctxsw  ctxsw ctxsw  ctxsw   ctxsw   ctxsw
------------ ------ ------ ------ ------ ------ ------- -------
Linux 2.6.14S 0.5600 1.2100   88.0   32.8  123.5    33.7   111.0
Linux 2.6.14S 1.2400 0.6500   82.7   32.2  123.8    34.0   110.7
Linux 2.6.14S 0.6100 0.9300   89.4   33.6  122.8    33.9   111.2
 Linux 2.6.11 5.2500 5.5800   99.1   40.2  131.1    41.2   119.1
 Linux 2.6.11 5.4400 5.7700   95.1   40.5  132.6    41.5   120.3
 Linux 2.6.11 7.8800 5.6900   98.9   40.5  132.8    41.8   120.0
```

```
*Local* Communication latencies in microseconds - smaller is better
---------------------------------------------------------------------
           OS 2p/0K  Pipe AF     UDP  RPC/   TCP  RPC/ TCP
              ctxsw       UNIX         UDP         TCP conn
------------ ----- ----- ---- ----- ----- ----- ----- ----
Linux 2.6.14S 0.560 5.768 9.00  17.8  38.5  25.2  51.6 100.
Linux 2.6.14S 1.240 6.114 10.4  18.2  38.4  27.9  54.3 101.
Linux 2.6.14S 0.610 5.789 9.29  18.1  34.7  26.5  58.0 108.
 Linux 2.6.11 5.250  36.8 59.8 144.2 196.7 174.0 234.5 636.
 Linux 2.6.11 5.440  37.8 60.7 144.6 191.6 174.8 228.1 625.
 Linux 2.6.11 7.880  38.8 60.2 144.2 192.8 176.1 229.9 621.

File & VM system latencies in microseconds - smaller is better
---------------------------------------------------------------------
           OS   0K File      10K File     Mmap    Prot   Page   100fd
              Create Delete Create Delete Latency Fault  Fault  selct
------------ ------ ------ ------ ------ ------- ----- ------- -----
Linux 2.6.14S   35.0   16.5  115.2   32.7   931.0 0.531 2.58240  13.3
Linux 2.6.14S   34.2   16.1  115.4   31.5   943.0 0.605 2.62850 5.605
Linux 2.6.14S   34.9   17.0  106.3   34.0   715.0 0.313 2.20530 5.854
 Linux 2.6.11  280.3  103.1  572.1  196.4  6172.0 0.197 8.41610 111.6
 Linux 2.6.11  280.0  103.7  570.1  196.7  6076.0       8.26760 111.6
 Linux 2.6.11  334.1  113.5  641.0  224.4  6438.0       8.45520 112.4

*Local* Communication bandwidths in MB/s - bigger is better
---------------------------------------------------------------------
          OS  Pipe AF    TCP  File   Mmap  Bcopy  Bcopy  Mem   Mem
                   UNIX      reread reread (libc) (hand) read write
------------ ---- ---- ---- ------ ------ ------ ------ ---- -----
Linux 2.6.14S 160. 157. 102.  400.6  696.2  292.0  292.6 608. 425.4
Linux 2.6.14S 154. 161. 102.  398.6  696.3  290.4  289.2 608. 430.2
Linux 2.6.14S 158. 167. 101.  398.1  696.3  288.1  289.0 608. 432.7
 Linux 2.6.11 121. 345. 69.1  313.0  683.0  287.8  288.9 596. 415.7
 Linux 2.6.11 124. 351. 69.4  314.2  682.5  286.2  284.2 596. 419.9
 Linux 2.6.11 124. 334. 68.8  312.4  682.8  284.3  285.2 596. 424.3

Memory latencies in nanoseconds - smaller is better
    (WARNING - may not be correct, check graphs)
---------------------------------------------------------------------
           OS   Mhz   L1 $   L2 $    Main mem    Guesses
```

```
-------------   ---   ----   ----    --------    -------
Linux 2.6.14S  1599 1.8780   12.8       157.5
Linux 2.6.14S  1599 1.8770   12.8       157.5
Linux 2.6.14S  1599 1.8780   12.8       157.5
 Linux 2.6.11  1599 1.9050   13.2       160.2
 Linux 2.6.11  1599 1.9050   13.3       160.3
 Linux 2.6.11  1599 1.9050   13.3       160.2
```

Lines listing Linux 2.6.14S are standard (unpatched Linux), lines listing 2.6.11 are kfi patched kernel runs. The only really unexplainable results are the AF UNIX results - no idea why a KFI patched kernel would show a factor two higher bandwidth (?). It also is interesting to observe how strong the distortion is - whith impact reaching from negligable (¡8% on 16p/64K context switches) to a factor 8 slower on 0k file creation.

Results are from three consecutive runs of lmbench-3.0-3a [?].

3.5. Status of KFI for 2.6.X

Unfortunately the quality of the documentation and the tools is clearly lower than was the case with the 2.4.X kfi releases. The kernel configuration is to be qualified as buggy.

4. Conclusion 2.6.X

Although 2.6.X code looks a lot cleaner than the 2.4.X code - there seems to be legacy comments (thats really bad) and some unimplemented "features" - comments on those (i.e. `TRIGGER_PROC`, `TRIGGER_LOG_FULL`) would be help-full. Also the lack of some sort of status interface is iritating (i.e. list installed runs and status of these).

Never the less KFI stays an interesting tool for runtime debugging of embedded Linux systems, and even more so for learning the details of the kernel.

Given the current status one can only recommend using/relying on this tool if one is willing to invest the engineering effort of actually understanding it at the source code level (which is not that much effort) making it possible to maintain/modify it on your own.

Although the current patch is officially for X86 only - we did not discover anything architecture dependent with the exception of the config files of other arch not being patched - thus it should be trivial to run KFI on other architectures as was the case with the 2.4.X KFI versions.

5. List of Acronyms

CVS - Concurent Version Control
GNU - GNU Not UNIX (recursive acronym)
LKML - Linux Kernel Mailing List
KFI - Kernel Function Instrumentation
GCC - GNU C Compiler
LTT - Linux Trace Toolkit
TSC - Time Stamp Counter (x86)
X86 - Intel 80X86 Processor family

References

[1] Free Software Foundation, *Free Documentation License*, as published by the Free Software Foundation, `http://www.gnu.org/copyleft/fdl.html`,FSF,2004

[2] Free Software Foundation, *General Public License*, as published by the Free Software Foundation, `http://www.gnu.org/copyleft/gpl.html`,FSF,2004

[3] Slackware Linux, *Slackware 10.1*, `http://www.slackware.org/`,Slackware Linux Inc.,2005

[4] CE Linux Forum, *Kernel Function Instrumentation*, `http://tree.celinuxforum.org`, (C) CE Linux Forum Member Companies, 2005.

[5] - Embedded Linux Kickstart Session, `http://www.opentech.at/documents.html`, 2004.

[6] Distributed and Embedded Systems Lab, Lanzhou University `http://dslab.lzu.edu.cn/`, DSLabs, 2006

[7] OpenTech EDV Research GmbH - OpenTech documents, `http://www.opentech.at/documents.html`,OpenTech,2005