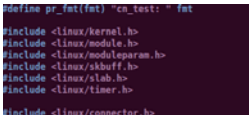
Linux Debugging and Kernel APIs

For Leadcoretech
Barry Song
April 2013

Kernel debug

Kernel – printk

- /proc/sys/kernel/printk
- dev_xxx
- pr_xxx
- #define pr_fmt(fmt) "hw-breakpoint: " fmt



Kernel – WARN_ON

- print_modules();
- dump_stack();
- print_oops_end_marker();

Kernel – BUG_ON/BUG

- printk("BUG: failure at %s:%d/%s()!\n", __FILE__, __LINE__, __func__); \
- panic("BUG!"); \

Kernel – oops - bug

```
/kernel$ git diff
diff --git a/tools/cma/cma_test.c b/tools/cma/cma_test.c
index 7eb96db..ba02d35 100644
--- a/tools/cma/cma_test.c
+++ b/tools/cma/cma_test.c
@@ -44,6 +44,9 @@ cma_test_read(struct file *file, char __user *buf,
size_t count, loff_t *ppos)

        if (!alloc)
                return -EIDRM;
+
+       volatile int *p=0;
+       *p = 0;

        dma_free_coherent(cma_dev, alloc->size, alloc->virt,
                alloc->dma);
```

Kernel – oops - PC

sh-4.2# cat /dev/cma_test
[33.491284] Unable to handle kernel NULL pointer dereference at virtual address 00000000
[33.507357] pgd = cc914000
[33.507379] [00000000] *pgd=0c902831, *pte=00000000, *ppte=00000000
[33.513769] Internal error: Oops: 817 [#1] PREEMPT
[33.518538] last sysfs file: /sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state
[33.526350] Modules linked in: vxdkernel pvrsrvkm sirfsoc_gps sirfsocfb cma_test
[33.533729] CPU: 0 Not tainted (2.6.38.8-sirf #28)
[33.538857] PC is at cma_test_read+0x8c/0xec [cma_test]
[33.544067] LR is at vfs_read+0xb8/0x144
[33.547966] pc : [<bf06f1cc>] lr : [<c039d64c>] psr: a0000013
[33.547972] sp : cc90ff08 ip : cc90ff30 fp : cc90ff2c
[33.559421] r10: 00000000 r9 : 00000003 r8 : beafd890
[33.564631] r7 : cc90ff68 r6 : bf06f4ec r5 : 00000000 r4 : d3e16460
[33.571139] r3 : 00000000 r2 : 00000000 r1 : beafd890 r0 : cc8e6e20
[33.577650] Flags: NzCv IRQs on FIQs on Mode SVC_32 ISA ARM Segment user
[33.584768] Control: 10c53c7d Table: 0c914059 DAC: 00000015
[33.590495]
[33.590498] LR: 0xc039d5cc:

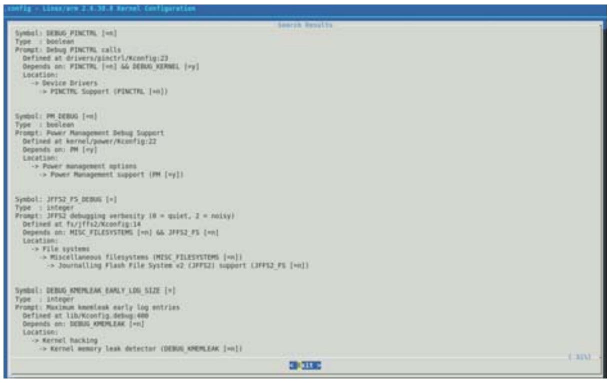
Kernel – oops -backtrace

[34.152437] Backtrace:
[34.154882] [<bf06f140>] (cma_test_read+0x0/0xec [cma_test]) from [<c039d64c>] (vfs_read+0xb8/0x144)
[34.163982] r6:beafd890 r5:cc8e6e20 r4:00001000
[34.168592] [<c039d594>] (vfs_read+0x0/0x144) from [<c039d724>] (sys_read+0x4c/0x108)
[34.176394] r8:beafd890 r7:00001000 r6:beafd890 r5:cc8e6e20 r4:000a8c34
[34.183093] [<c039d6d8>] (sys_read+0x0/0x108) from [<c02dc720>] (ret_fast_syscall+0x0/0x30)
[34.191416] Code: 03e0002a 0a000012 e59f6058 e3a05000 (e5855000)
[34.294365] ---[end trace f0d7620b9f61d90d]---

Kernel – oops - objdump

[33.538857] PC is at cma_test_read+0x8c/0xec [cma_test]

Kernel – DEBUG options



Kernel – PM_DEBUG and no_console_suspenbd

- Enable PM_DEBUG in Kconfig

[*] Power Management support | |
| | [*] Power Management Debug Support | |
| | [*] Extra PM attributes in sysfs for low-level debugging/testing | |
| | [*] Verbose Power Management debugging

- Set no_console_suspend in bootargs

Kernel – Android init "untracked PID exited"

```
diff --git a/kernel/exit.c b/kernel/exit.c
index 7cdb3a6..36ead86 100644
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -909,6 +909,16 @@ NORET_TYPE void do_exit(long code)
        struct task_struct *tsk = current;
        int group_dead;

+       /*
+        * we want to track the exit of "untracked PID exited" events in Android, eg,
+        * in the process of suspend/resume:
+        * [  222.660982] task exit: tgid(PID):1293 pid:1302 name:synergy_service
+        * [  222.731344] init: untracked pid 1293 exited
+        */
+       if (tsk->parent->tgid == 1)
+               printk(KERN_DEBUG "task exit: tgid(PID):%d pid:%d name:%s\n",
+                       tsk->tgid, tsk->pid, tsk->comm);
+
        profile_task_exit(tsk);

        WARN_ON(atomic_read(&tsk->fs_excl));
```

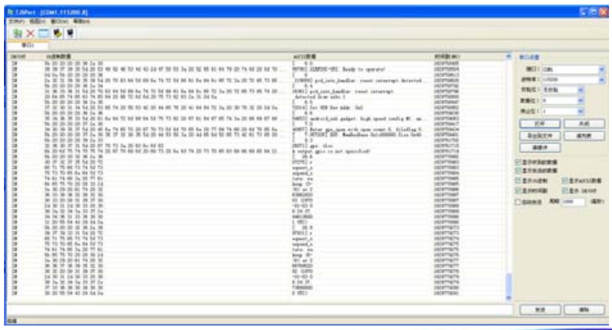
Kernel – /dev/ttyprintk

- ttyprintk is a pseudo TTY driver, which allows users to make printk messages

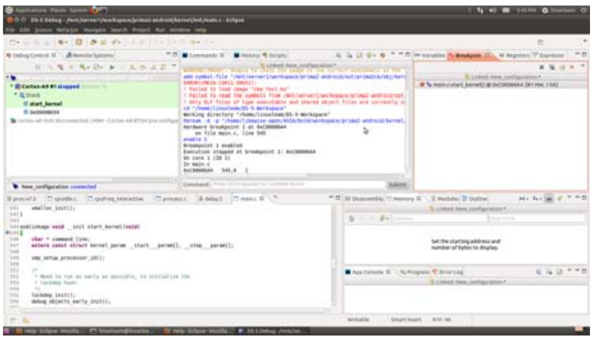
sh-4.2# echo "hello world" > /dev/ttyprintk
[12005.502521] [U] hello world

Print with timestamp

- Tjsport and similar tools



DS-5 – debug kernel



DS-5 – debug module

- Target

sh-4.2# pwd
/sys/module/cma_test/sections

sh-4.2# ls -a
. .note.gnu.build-id
.. .rodata
.bss .rodata.str1.1
.data .strtab
.exit.text .symtab
.gnu.linkonce.this_module .text
.init.text

sh-4.2# cat .text
0xbf06f000
sh-4.2# cat .data
0xbf06f36c

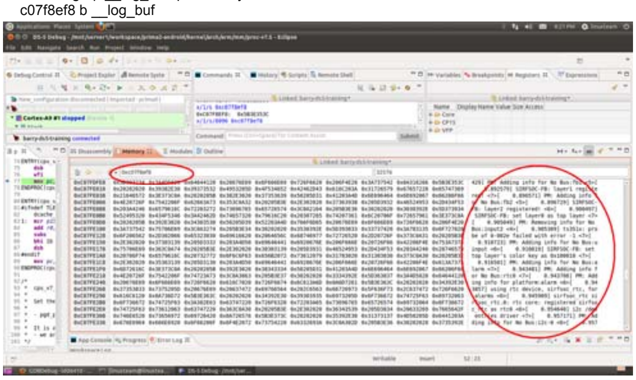
- Host

add-symbol-file kernel/tools/cma/cma_test.ko -s .text 0xbf06f000 -s .data 0xbf06f36c

b cma_test_read

DS-5 – get printk even after kernel hang

sh-4.2# grep __log_buf /proc/kallsyms
c07f8ef8 b __log_buf



Kernel – DEBUG_LL/EARLY_PRINTK

Enable EARLY_PRINTK, DEBUG_LL to support early print and include definitions of printascii, printch, printhex in the kernel. This is helpful if you are debugging code that executes before the console is initialized.

Kernel – initcall_debug

Passing the option "initcall_debug" on the kernel command line will cause timing information to be printed to the console for each initcall. You will need to enable CONFIG_PRINTK_TIME and CONFIG_KALLSYMS in your kernel configuration for this to work correctly.

calling ipc_init+0x0/0x28 @ 1
msgmni has been set to 42
initcall ipc_init+0x0/0x28 returned 0 after 1872 usecs

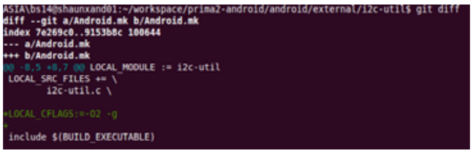
dmesg -s 128000 | grep "initcall" | sed "s/\(.*\)after\(.*\)/\2 \1/g" | sort -n

---

GDB

GDB – compile with debug info

- -g



```
ASIA\bs14@shaunxand01:~/workspace/prima2-android/android/external/i2c-util$ git diff
diff --git a/Android.mk b/Android.mk
index 7e269c0..9153b8c 100644
--- a/Android.mk
+++ b/Android.mk
@@ -8,5 +8,7 @@ LOCAL_MODULE := i2c-util
 LOCAL_SRC_FILES += \
        i2c-util.c \

+LOCAL_CFLAGS:=-O2 -g
+
 include $(BUILD_EXECUTABLE)
```

- …/**symbols**/system/bin$ file i2c-util

i2c-util: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), not stripped

---

GDB – ARM Native GDB

- Target

```
sh-4.2# gdb /system/bin/i2c-util
…
Reading symbols from /system/bin/i2c-util...done.
(gdb) r
Starting program: /system/bin/i2c-util
[ 1263.613036] init: untracked pid 1032 exited
BFD: /system/bin/linker: warning: sh_link not set for section `.ARM.exidx'
BFD: /system/bin/linker: warning: sh_link not set for section `.ARM.exidx'
warning: Unable to find dynamic linker breakpoint function.
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.

Program received signal SIGSEGV, Segmentation fault.
main (argc=1, argv=0xbea14a54) at external/i2c-util/i2c-util.c:131
131     external/i2c-util/i2c-util.c: No such file or directory.
        in external/i2c-util/i2c-util.c
(gdb) bt
#0  main (argc=1, argv=0xbea14a54) at external/i2c-util/i2c-util.c:131
```

- Known issues: can't recognize multi-threads.

GDB – gdbserver/gdb over ADB

- Target

```
sh-4.2# gdbserver :5039 /system/bin/i2c-util
Process /system/bin/i2c-util created; pid = 1059
Listening on port 5039
```

- Host

```
# adb forward tcp:5039 tcp:5039
prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin/arm-eabi-gdb /mnt/server1/workspace/lianlab-android/android/out/target/product/lianlab-
        armv7/symbols/system/bin/i2c-util
(gdb) target remote :5039
(gdb) set solib-absolute-prefix android/out/target/product/lianlab-armv7/symbols/
(gdb) set solib-search-path /mnt/server1/workspace/lianlab-android/android/out/target/product/lianlab-armv7/symbols/system/lib/
(gdb) c
Program received signal SIGSEGV, Segmentation fault.
main (argc=1, argv=0xbeeb3a74) at external/i2c-util/i2c-util.c:131
131         *p = 0;
(gdb) l
126         unsigned w_val;
127         unsigned char rw_val;
128
129         volatile unsigned int *p=0;
130
131         *p = 0;
132
133         if (argc < 5) {
134                 printf("Usage:\n%s /dev/i2c-x start_addr reg_addr rw[0|1] [write_val]\n", argv[0]);
135                 return 0;
```

GDB – attach one process

- Target

```
sh-4.2# ps
  PID USER       VSZ STAT COMMAND
  599 root     40256 S    /system/bin/mediaserver
sh-4.2# gdbserver :5039 --attach 599
Attached; pid = 599
Listening on port 5039
```

- Host

```
# adb forward tcp:5039 tcp:5039
 prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin/arm-eabi-gdb /mnt/server1/workspace/lianlab-android/android/out/target/product/lianlab-
        armv7/symbols/system/bin/mediaserver
(gdb) set solib-absolute-prefix /mnt/server1/workspace/lianlab-android/android/out/target/product/lianlab-armv7/symbols/
(gdb) set solib-search-path /mnt/server1/workspace/lianlab-android/android/out/target/product/lianlab-armv7/symbols/system/lib/
(gdb) target remote :5039
Remote debugging using :5039
…
Reading symbols from /mnt/server1/workspace/lianlab-android/android/out/target/product/lianlab-armv7/symbols/system/lib/libc.so...done.
Loaded symbols for /mnt/server1/workspace/lianlab-android/android/out/target/product/lianlab-armv7/symbols/system/lib/libc.so
…
__ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15
15        ldmfd   sp!, {r4, r7}
(gdb) info threads
[New Thread 617] …
[New Thread 1019]
  12 Thread 1019  __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200 …
* 1 Thread 599  __ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15
```

GDB – multi-threads

- Switch threads

```
(gdb) info threads
 …
 8 Thread 709  __ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15
 7 Thread 627  __ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15
 6 Thread 626  __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200
* 5 Thread 623  __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200
 4 Thread 622  __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200
…
(gdb) thread 8
[Switching to thread 8 (Thread 709)]#0  __ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15
15        ldmfd   sp!, {r4, r7}
(gdb) info threads
 …
 9 Thread 710  __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200
* 8 Thread 709  __ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15
 7 Thread 627  __ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15
 6 Thread 626  __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200
 5 Thread 623  __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200
…
```

- Global mode

```
(gdb) set scheduler-locking off
```

- Single thread mode

```
(gdb) set scheduler-locking on
```

Core Dump

- sh-4.2# ulimit -c unlimited
- sh-4.2# /data/i2c-util /dev/i2c-0 0x10 1 0

Segmentation fault (core dumped)

strace – trace the system call

```
sh-4.2# strace /system/bin/i2c-util
execve("/system/bin/i2c-util", ["/system/bin/i2c-util"], [/* 25 vars */]) = 0
...
stat64("/vendor/lib/libm.so", 0xbec7f6f8) = -1 ENOENT (No such file or directory)
stat64("/system/lib/libm.so", {st_mode=S_IFREG|0644, st_size=91328, ...}) = 0
open("/system/lib/libm.so", O_RDONLY|O_LARGEFILE) = 3
lseek(3, 0, SEEK_SET)                = 0
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0(\0\1\0\0\0\0\0\0\000"..., 4096) = 4096
…
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40127000
mprotect(0x40127000, 4096, PROT_READ)   = 0
--- SIGSEGV (Segmentation fault) @ 0 (0) ---
…
read(3, 0xbec7f724, 1)               = ? ERESTARTSYS (To be restarted)
--- SIGCONT (Continue) @ 0 (0) ---
read(3, "", 1)                       = 0
close(3)                             = 0
sigaction(SIGSEGV, {SIG_IGN}, {0xb0005a91, [], SA_RESTART}, 0) = 0
sigreturn()                          = ? (mask now [QUIT TRAP FPE SEGV PIPE CHLD CONT])
--- SIGSEGV (Segmentation fault) @ 0 (0) ---
+++ killed by SIGSEGV +++
```

ltrace – A library call tracer

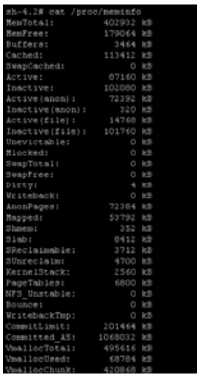
```
__libc_start_main(0x400844, 1, 0x7fff01389398, 0x400880, 0x400870 <unfinished ...>
    _ZNSt8ios_base4InitC1Ev(0x600e2c, 65535, 0x7fff013893a8, 3,
    0x31e1752350)            = 2
    __cxa_atexit(0x40082c, 0, 0x400960, 3, 0x31e1752350)                 = 0
    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc(0x600d10, 0x400968,
    0x7fff013893a8, 4, 0x31e1752370) = 0x600d10
    _ZNSolsEPFRSoS_E(0x600d10, 0x4006e0, 0, 0xfbad2a84, 0xffffffff <unfinished ...>
    _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(0x600d10, 0x4006e0, 0,
    0xfbad2a84, 0xffffffffHello World !
    ) = 0x600d10
    _ZNSt8ios_base4InitD1Ev(0x600e2c, 0, 0x31e1752370, -1,
    0x2b2e79280530)                = 3
```

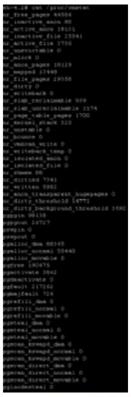
schedutils

- chrt
- taskset

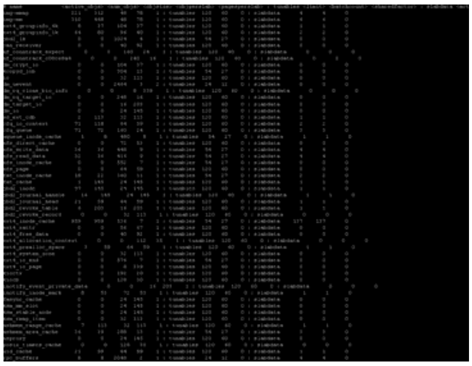
Memory usage and leak check

/proc/meminfo, vmstat

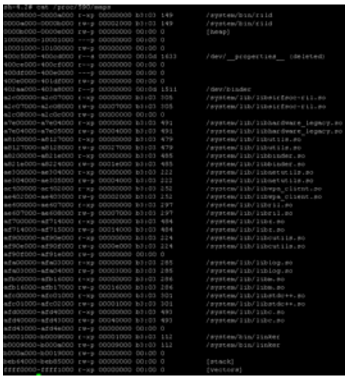




/proc/slabinfo

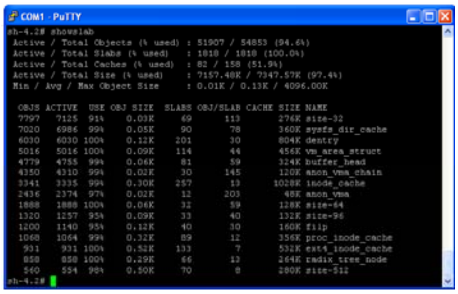


/proc/pid/maps

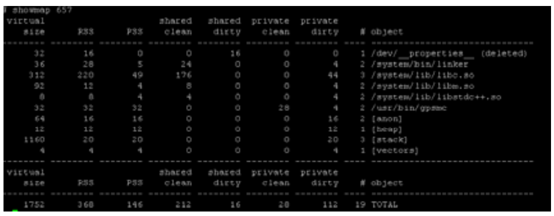


free

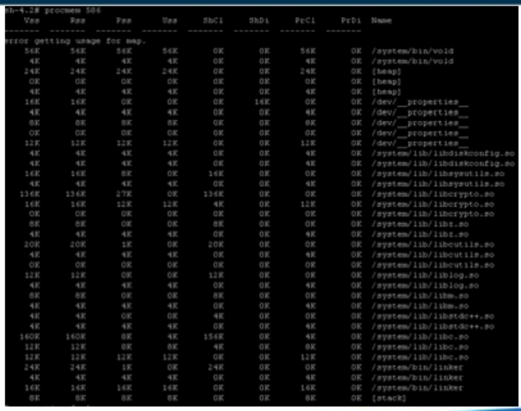
showslab



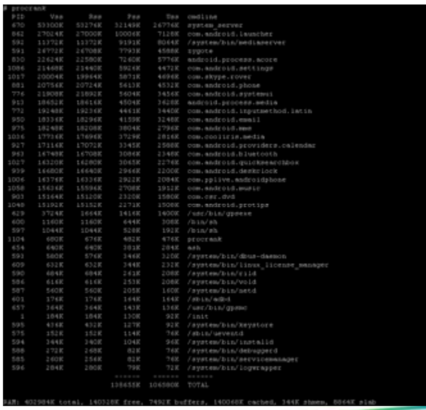
showmap



procmem

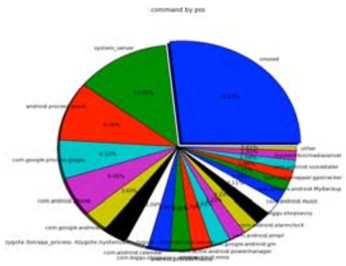


procrank



smem – process memory

$ ./smemcap >memdata.tar
$ adb pull /data/smem/memdata.tar .
smem -S memdata.tar
smem -S memdata.tar --pie=command



smem – the users of libraries

smem -S memdata.tar -m -M lib -s pids

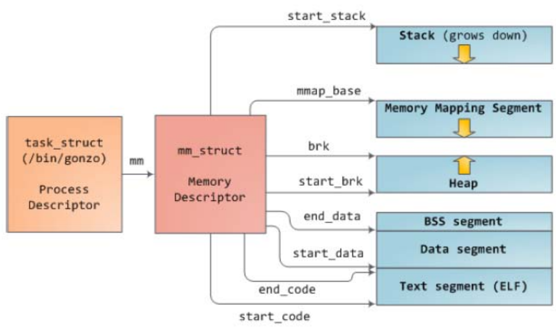
```
root@barry-VirtualBox:/media/sf_Developer# smem -S memdata.tar -m -M lib -s pids
Map                                          PIDs   AVGPSS      PSS
/lib/libcsrcert.so                              1       12       12
/lib/libpthread-2.13.so                         1       56       56
/system/lib/dejitter.so                         1        8        8
/system/lib/gst/libGSTPlayerImpFactory.s        1       52       52
/system/lib/gst/libavcodec.so.52.72.2           1      104      104
/system/lib/gst/libavformat.so.52.64.2          1       40       40
/system/lib/gst/libavutil.so.50.15.1            1       16       16
/system/lib/gst/libglib-2.0.so.0.2400.1         1      440      440
/system/lib/gst/libgmodule-2.0.so.0.2400        1       20       20
/system/lib/gst/libgobject-2.0.so.0.2400        1      272      272
/system/lib/gst/libgstaudio-0.10.so.0.22        1      104      104
/system/lib/gst/libgstbase-0.10.so.0.27.        1      232      232
/system/lib/gst/libgstinterfaces-0.10.so        1       32       32
/system/lib/gst/libgstpbutils-0.10.so.0.        1       36       36
/system/lib/gst/libgstreamer-0.10.so.0.2        1      892      892
/system/lib/gst/libgstriff-0.10.so.0.22.        1       12       12
/system/lib/gst/libgsttag-0.10.so.0.22.0        1       64       64
/system/lib/gst/libgstvideo-0.10.so.0.22        1       12       12
/system/lib/gst/libgthread-2.0.so.0.2400        1       20       20
/system/lib/gst/libogg.so.0.7.0                 1       28       28
/system/lib/gst/libomx_core.so                  1      148      148
/system/lib/gst/plugins/libgstaudioconve        1       56       56
```

smem –multi-processes using one library

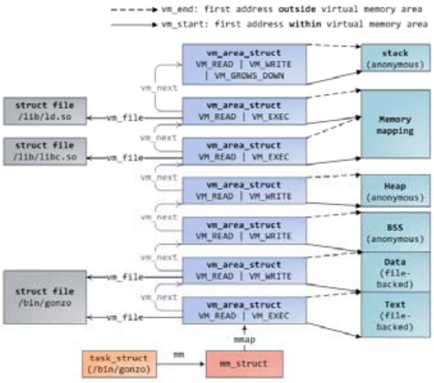
smem -S memdata.tar -M libc

```
root@barry-VirtualBox:/media/sf_Developer# smem -S memdata.tar -M libc
  PID User     Command                         Swap      USS      PSS      RSS
  567 1000     /system/bin/servicemanager         0       12       15      152
  570 0        /system/bin/debuggerd              0       12       15      156
  578 0        /system/bin/logwrapper /sys        0       12       16      172
  640 0        /usr/bin/gpsmc                     0       12       17      188
  613 0        /usr/bin/gpsexe -triglite -        0       12       18      196
  575 1002     /system/bin/dbus-daemon --s        0       16       19      160
 1430 10006    android.process.media              0        8       20      404
 1511 10004    com.csr.dvd                        0        8       20      404
 1522 10008    com.android.providers.calen        0        8       20      404
 1536 10012    com.android.bluetooth              0        8       20      404
 1544 10030    com.android.deskclock              0        8       20      404
 1559 10032    com.pplive.androidphone            0        8       20      400
 1578 10013    com.android.protips                0        8       20      404
 1587 10016    com.android.music                  0        8       20      404
 1595 10029    com.android.quicksearchbox         0        8       20      404
  572 0        /system/bin/rild                   0       16       21      208
  576 0        /system/bin/installd               0       16       21      196
 1391 10005    com.android.inputmethod.pin        0        8       21      412
 1396 10009    com.android.launcher               0        8       21      412
```

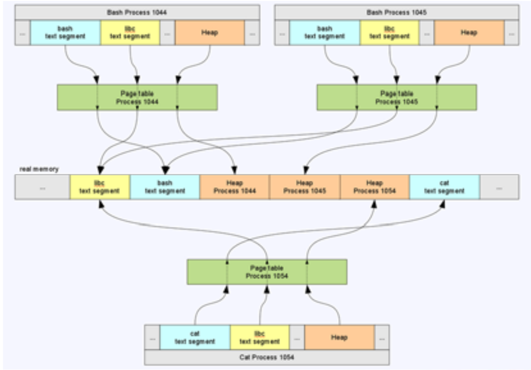
Linux Process Memory



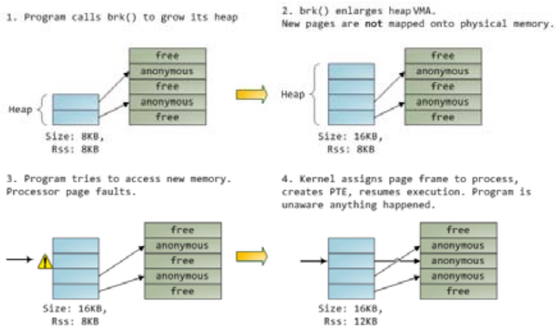
Binaries and Libraries



Memory Mapping of Multi-processes



VSS vs. RSS



valgrind - memory leak detection

Push to /data/local/valgrind

export VALGRIND_LIB=/data/local/valgrind/lib/valgrind

Run:
/data/local/valgrind/bin/valgrind /system/bin/i2c-util

```
sh-4.2# ./valgrind /system/bin/i2c-util
==1139== Memcheck, a memory error detector
==1139== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==1139== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==1139== Command: /system/bin/i2c-util
==1139==
==1139== Invalid write of size 4
==1139==    at 0x8840: main (i2c-util.c:131)
==1139==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==1139==
==1139==
==1139== HEAP SUMMARY:
==1139==     in use at exit: 0 bytes in 0 blocks
==1139==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==1139==
==1139== All heap blocks were freed -- no leaks are possible
==1139==
==1139== For counts of detected and suppressed errors, rerun with: -v
==1139== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

valgrind - memory check

- Illegal read / Illegal write errors
- Use of uninitialised values
- Use of uninitialised or unaddressable values in system calls
- Illegal frees
- When a heap block is freed with an inappropriate deallocation function
- Overlapping source and destination blocks
- Memory leak detection

valgrind - issues

```
main()
{
        {
                int x;
                printf ("x = %d\n", x);
        }
        {
                char* arr  = malloc(10);
                int*  arr2 = malloc(sizeof(int));
                write( 1 /* stdout */, arr, 10 );
        }
        {
                char a[100];
                memcpy(a, a + 20, 40);
        }
        {
                char *q;
                q = malloc(1024*1024);

                q[1] = 1024;
        }
        {
                char *p;
                p = malloc(1024*1024);

                p[0] = p[0];
                p[1] = 1024;

                free(p);
                free(p);
        }
}
```

valgrind - reports

/data/local/valgrind/bin/valgrind --leak-check=full --track-origins=yes /data/check

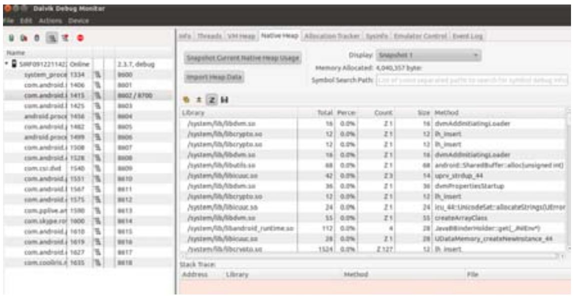
valgrind on Android

To start an application (eg Firefox), a message is sent via a socket to the Zygote. This creates a child with fork(), and the child then goes on to load the relevant bytecode and (presumably) native code and “becomes” Firefox. So there’s no exec() boundary for Valgrind to enter at.

the AOSP folks modified Zygote so that it can start selected processes under the control of a **user-specified wrapper**, which is precisely the hook we need.

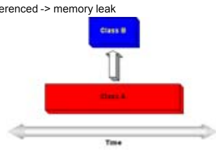
Android JNI native memory leak

add "native=true" in ~/.android/ ddms.cfg
*adb shell setprop **libc.debug.malloc** 1*
adb shell stop
adb shell start
export ANDROID_PRODUCT_OUT=/mnt/server1/workspace/lianlab-android/android/out/target/product/lianlab-armv7

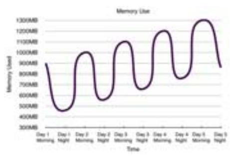


Java memory leak

- Java is that they no longer have to worry about allocating and freeing memory
- Unused but still referenced -> memory leak



- Memory leak diagram



- hprof : heap dump

Minor utilities

logcat

- Logcat usage

```
Usage: logcat [options] [filterspecs]
options include:
  -s              Set default filter to silent.
                  Like specifying filterspec '*:s'
  -f <filename>   Log to file. Default to stdout
  -r [<kbytes>]   Rotate log every kbytes. (16 if unspecified). Requires -f
  -n <count>      Sets max number of rotated logs to <count>, default 4
  -v <format>     Sets the log print format, where <format> is one of:

                  brief process tag thread raw time threadtime long

  -c              clear (flush) the entire log and exit
  -d              dump the log and then exit (don't block)
  -t <count>      print only the most recent <count> lines (implies -d)
  -g              get the size of the log's ring buffer and exit
  -b <buffer>     request alternate ring buffer
                  ('main' (default), 'radio', 'events')
  -B              output the log in binary
```

logwrapper

- Init will redirect service stdin, stdout, stderr to /dev/null
- Logwrapper will read all printf/perror and call LOG(LOG_INFO, tag, "%s", &buffer[a]);

dup2(child_ptty, 1);
dup2(child_ptty, 2);

- Examples:

service license-manager /system/bin/logwrapper /system/bin/linux_license_manager
  oneshot

Log in CPP

```
/*
 * Simplified macro to send a verbose log message using the current LOG_TAG.
 */
#ifndef LOGV
#if LOG_NDEBUG
#define LOGV(...)   ((void)0)
#else
#define LOGV(...) ((void)LOG(LOG_VERBOSE, LOG_TAG, __VA_ARGS__))
#endif
#endif

#ifndef LOGV_IF
#if LOG_NDEBUG
#define LOGV_IF(cond, ...)   ((void)0)
#else
#define LOGV_IF(cond, ...) \
   ( (CONDITION(cond)) \
   ? ((void)LOG(LOG_VERBOSE, LOG_TAG, __VA_ARGS__)) \
   : (void)0 )
#endif
#endif

webkit/WebCore/platform/brew/CursorBrew.cpp:#define LOG_TAG "WebCore"
webkit/WebCore/platform/android/CursorAndroid.cpp:#define LOG_TAG "WebCore“
```

Log in JAVA

```
DownloaderActivity.java:    private final static String LOG_TAG = "Downloader";

quake/src/com/android/quake/DownloaderActivity.java:            Log.i(LOG_TAG, "Versions match, no need to download.");
quake/src/com/android/quake/DownloaderActivity.java:          Log.e(LOG_TAG, "Unable to read local config file", e);
quake/src/com/android/quake/DownloaderActivity.java:        Log.i(LOG_TAG, "Download succeeded");
quake/src/com/android/quake/DownloaderActivity.java:        Log.e(LOG_TAG, "Download stopped: " + reason);
quake/src/com/android/quake/DownloaderActivity.java:            Log.i(LOG_TAG, "Network connectivity issue, retrying.");
quake/src/com/android/quake/DownloaderActivity.java:                Log.i(LOG_TAG, "Couldn't find local config.");
quake/src/com/android/quake/DownloaderActivity.java:                Log.i(LOG_TAG, "Local version out of sync. Wanted " +
quake/src/com/android/quake/DownloaderActivity.java:            Log.i(LOG_TAG, "Creating directory " + mDataPath);
```

I2C testing utility (using i2c-dev driver)

- Read/write registers in I2C client at userspace

/system/bin/i2c-util /dev/i2c-2 0x11 0x22 0
/system/bin/i2c-rw /dev/i2c-2 0x11 0x22 1 0xff

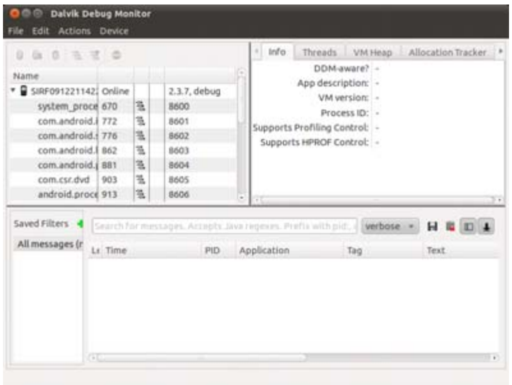
SPI testing utility (using spidev driver)

- Documentation/spi/spidev_test.c

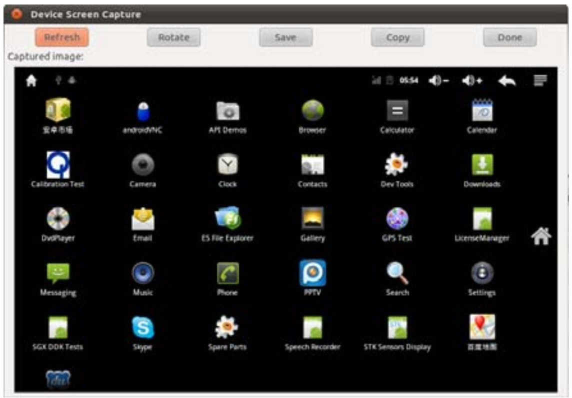
```
int main(int argc, char *argv[])
{
    fd = open(device, O_RDWR);
    ...

    ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);
    ret = ioctl(fd, SPI_IOC_RD_MODE, &mode);
    ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
    ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
    ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
    ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
    transfer(fd);
}
```

Android DDMS

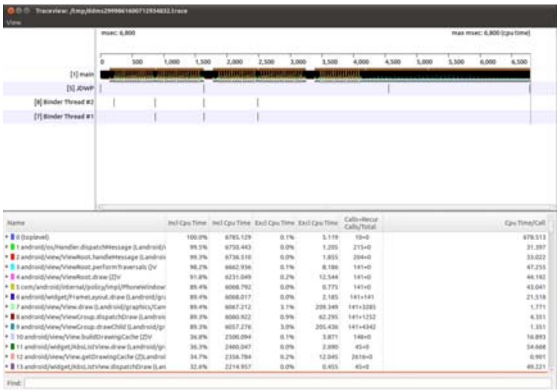
DDMS - Dalvik Debug Monitor Service



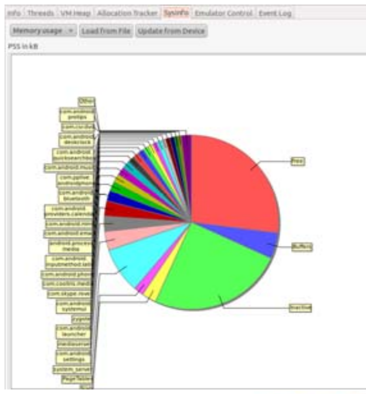
DDMS – Capture Framebuffer



DDMS – traceview



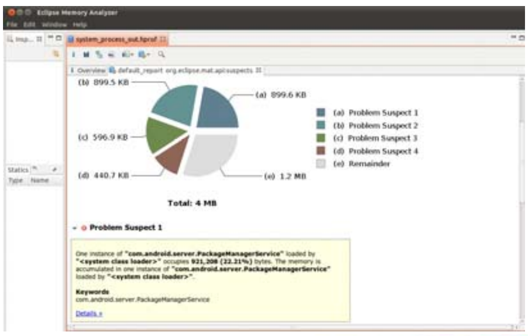
DDMS – sysinfo



DDMS-hprof

- vmuser@ubuntu:/data/android-sdk-linux/tools$ ./hprof-conv system_process.hprof system_process_out.hprof

Refer to: http://wenku.baidu.com/view/bf788a02b52acfc789ebc9c4.html



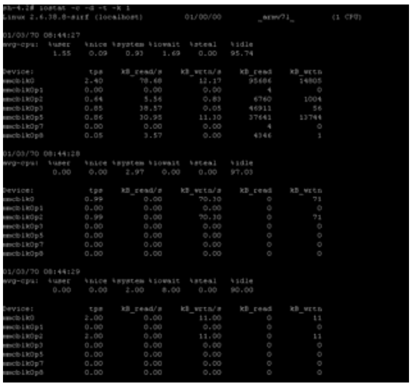
Performance profiling

top

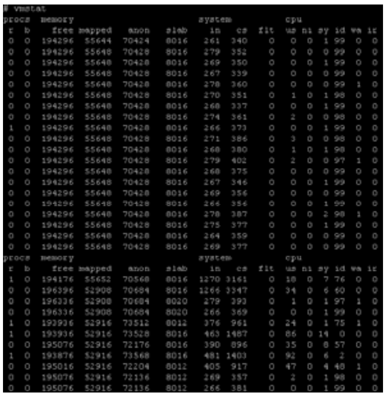
User 0%, System 0%, IOW 0%, IRQ 0%
User 2 + Nice 0 + Sys 3 + Idle 297 + IOW 1 + IRQ 0 + SIRQ 0 = 303

```
  PID CPU% S  #THR     VSS     RSS PCY UID      Name
 1017   1% R     1    964K    376K  fg root     top
  615   0% S    11  16076K   1796K  fg root     /usr/bin/gpsexe
  574   0% S    10  39756K   7576K  fg root     /system/bin/mediaserver
  555   0% S     1      0K      0K  fg root     jbd2/mmcblk0p2-
  584   0% S    10  14756K   2356K  fg root
/system/bin/synergy_service
    6   0% S     1      0K      0K  fg root     khelper
   13   0% S     1      0K      0K  fg root     suspend
   14   0% S     1      0K      0K  fg root     kworker/0:1
  198   0% S     1      0K      0K  fg root     sync_supers
  200   0% S     1      0K      0K  fg root     bdi-default
```

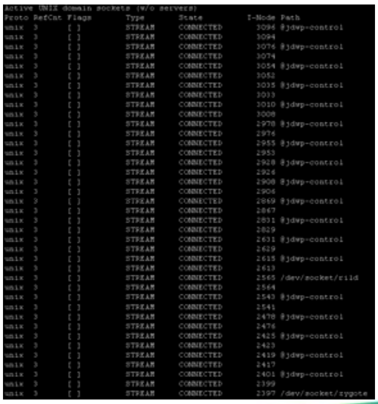
iostat - storage input and output statistics



vmstat - memory, processes, interrupts, paging



netstat – socket communication status



mpstat – show multi-core CPUs

```
barry@lianlab:~$ mpstat -P ALLLinux 2.6.32-39-server (lianlab)
          04/12/2012        _x86_64_          (16 CPU)01:54:06 PM  CPU    %usr
%nice    %sys %iowait    %irq   %soft  %steal  %guest   %idle01:54:06 PM  all
1.49    0.00    0.16    0.08    0.00    0.01    0.00    0.00   98.2701:54:06 PM    0
1.98    0.00    0.23    0.49    0.00    0.03    0.00    0.00   97.2701:54:06 PM    1
2.05    0.00    0.28    0.48    0.00    0.02    0.00    0.00   97.1701:54:06 PM    2
1.79    0.00    0.27    0.03    0.00    0.00    0.00    0.00   97.9101:54:06 PM    3
1.93    0.00    0.29    0.03    0.00    0.00    0.00    0.00   97.7401:54:06 PM    4
1.44    0.00    0.15    0.02    0.00    0.00    0.00    0.00   98.4001:54:06 PM    5
1.38    0.00    0.15    0.02    0.00    0.00    0.00    0.00   98.4501:54:06 PM    6
1.34    0.00    0.13    0.01    0.00    0.00    0.00    0.00   98.5201:54:06 PM    7
1.52    0.00    0.16    0.01    0.00    0.00    0.00    0.00   98.3001:54:06 PM    8
1.28    0.00    0.12    0.04    0.00    0.01    0.00    0.00   98.5501:54:06 PM    9
1.34    0.00    0.13    0.04    0.00    0.00    0.00    0.00   98.4901:54:06 PM   10
1.37    0.00    0.14    0.02    0.00    0.00    0.00    0.00   98.4701:54:06 PM   11
1.38    0.00    0.13    0.02    0.00    0.00    0.00    0.00   98.4601:54:06 PM   12
1.31    0.00    0.13    0.01    0.00    0.00    0.00    0.00   98.5501:54:06 PM   13
1.20    0.00    0.11    0.01    0.00    0.00    0.00    0.00   98.6701:54:06 PM   14
1.17    0.00    0.10    0.01    0.00    0.00    0.00    0.00   98.7201:54:06 PM   15
1.29    0.00    0.12    0.01    0.00    0.00    0.00    0.00   98.58
```

time – measures the runtime of a process

- – Three figures provided
- – real
- – user
- – sys

Benchmark tools

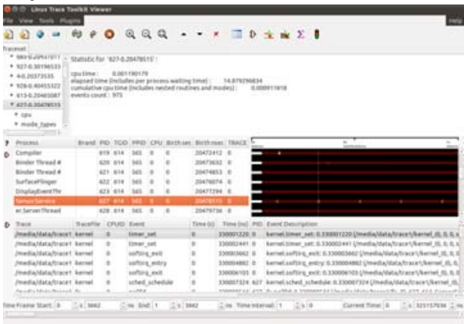
- lmbench
Microbenchmark for operating system functions

- Iozone
File system benchmark

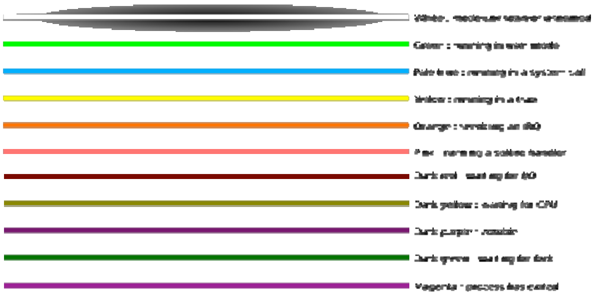
- Netperf
Network performance benchmark

LTTng - usage

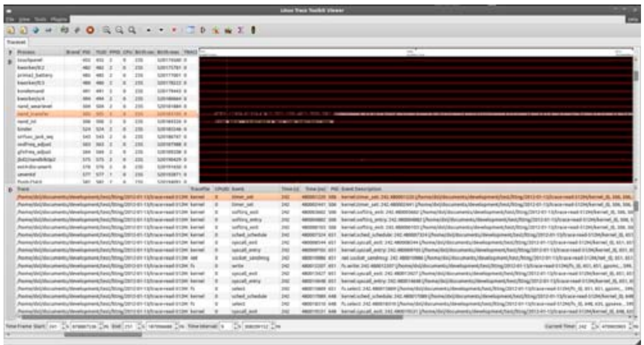
- # ltt-armall
- # lttctl -C -w /mnt/obb/trace1 trace1
- # lttctl -D trace1
- lttv -m batchAnalysis -s -m textDump -s -t trace
- lttv-gui



lttv-gui - color

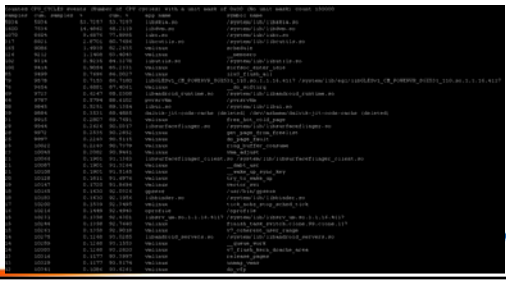


LTTng –NAND performance

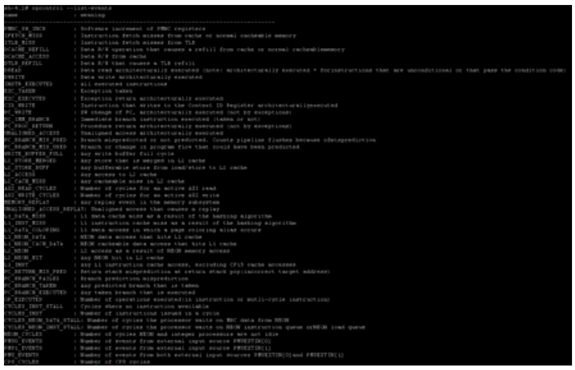


Oprofile -usage

- # modprobe oprofile
- # grep " _text" /proc/kallsyms
  c02d6000 T _text
- # grep " _etext" /proc/kallsyms
  c07bc000 A _etext
- # opcontrol --vmlinux=/vmlinux --kernel-range=0xc02d6000,0xc07bc000 --quick
- opcontrol --vmlinux=/vmlinux --kernel-range=0xc02d6000,0xc07bc000 --event=DCACHE_ACCESS:1000 --event=L1_DATA_MISS:1000 --event=L1_INST_MISS:1000
- # opcontrol --start
- # opcontrol --dump
- # opreport -la



Oprofile - events



gprof

- cc -g -c myprog.c utils.c -pg
- cc -o myprog myprog.o utils.o –pg
- gprof *options* [*executable-file* [*profile-data-files*...]] [> *outfile*]

DS-5 – streamline

- Target

modprobe gator

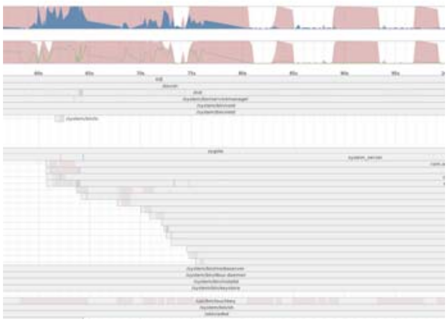
gatord 5039

- Host

adb forward tcp:5039 tcp:5039

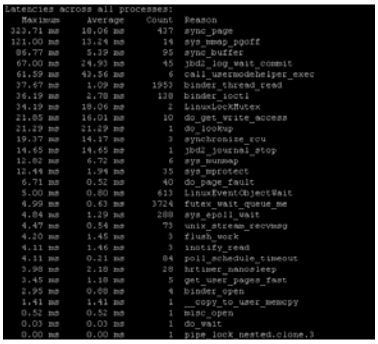
Use localhost:5039 to do capture

Bootchart



Latencytop

- /system/xbin/latencytop

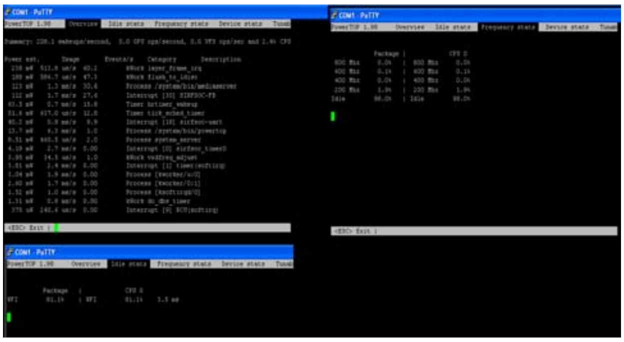


Latencies across all processes:

Maximum	Average	Count	Reason
323.71 ms	10.06 ms	437	sync_page
121.00 ms	13.24 ms	14	sys_mmap_pgoff
86.77 ms	5.39 ms	95	sync_buffer
67.00 ms	24.93 ms	45	jbd2_log_wait_commit
61.59 ms	43.56 ms	6	call_usermodehelper_exec
37.67 ms	1.09 ms	1953	binder_thread_read
36.19 ms	2.78 ms	138	binder_ioctl
34.19 ms	18.06 ms	2	LinuxLockMutex
21.85 ms	16.01 ms	10	do_get_write_access
21.29 ms	21.29 ms	1	do_lookup
19.37 ms	14.17 ms	3	synchronize_rcu
14.65 ms	14.65 ms	1	jbd2_journal_stop
12.82 ms	6.72 ms	6	sys_munmap
12.44 ms	1.94 ms	35	sys_mprotect
6.71 ms	0.52 ms	40	do_page_fault
5.00 ms	0.80 ms	613	LinuxEventObjectWait
4.99 ms	0.63 ms	3724	futex_wait_queue_me
4.84 ms	1.29 ms	288	sys_epoll_wait
4.47 ms	0.54 ms	73	unix_stream_recvmsg
4.20 ms	1.45 ms	3	flush_work
4.11 ms	1.46 ms	3	inotify_read
4.11 ms	0.21 ms	84	poll_schedule_timeout
3.98 ms	2.18 ms	28	hrtimer_nanosleep
3.45 ms	1.18 ms	5	get_user_pages_fast
2.95 ms	0.88 ms	4	binder_open
1.41 ms	1.41 ms	1	__copy_to_user_memcpy
0.52 ms	0.52 ms	1	misc_open
0.03 ms	0.03 ms	1	do_wait
0.00 ms	0.00 ms	1	pipe_lock_nested.clone.3

DVFS- powertop

- /system/bin/powertop



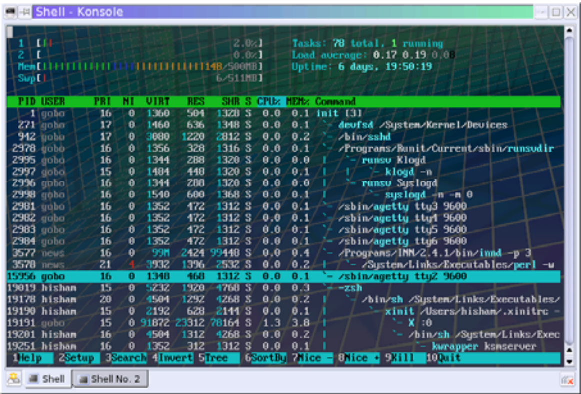
DVFS – cpufreq-bench

/usr/sbin/cpufreq-bench -l 50000 -s 100000 -x 50000 -y 100000 -g ondemand -r 5 -n 5 –v

Ondemand:
Round 1 - 39.74%
Round 2 - 36.35%
Round 3 - 47.91%
Round 4 - 54.22%
Round 5 - 58.64%

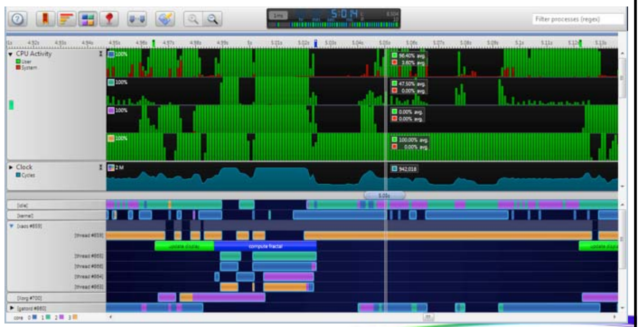
Interactive:
Round 1 - 72.95%
Round 2 - 87.20%
Round 3 - 91.21%
Round 4 - 94.10%
Round 5 - 94.93%

htop



streamline

- Streamline features per core performance counter charts and the X-Ray visualization mode, which maps process and thread activity per core



cyclictest

- *This tool acquires timer jitter by measuring accuracy of sleep and wake operations of highly prioritized realtime threads.*

```
# ./cyclictest -p 80 -t5 -n
1.58 1.61 1.62 3/68 4079
T: 0 ( 3131) P:80 I:    1000 C:16469865 Min:      8 Act:     13 Max: 62
T: 1 ( 3132) P:79 I:    1500 C: 9979903 Min:      8 Act:     18 Max: 75
T: 2 ( 3133) P:78 I:    2000 C: 7934887 Min:      9 Act:     22 Max: 83
T: 3 ( 3134) P:77 I:    2500 C: 6587910 Min:      9 Act:     25 Max: 81
T: 4 ( 3135) P:76 I:    3000 C: 5489925 Min:      9 Act:     27 Max: 86
```

Linux tracer

- # mount -t debugfs none /sys/kernel/debug
- # cd /sys/kernel/debug/tracing
- # cat events/sched/sched_process_fork/enable 0
- # echo 1 > events/sched/sched_process_fork/enable

```
# tracer: nop #
#           TASK-PID    CPU#     TIMESTAMP  FUNCTION
#              | |       |          |         |
             zsh-2667  [001]  6658.716936: sched_process_fork: comm=zsh
  pid=2667 child_comm=zsh child_pid=2748
```

Kernel programming

Barriers: Motivation

- The compiler can:
  - Reorder code as long as it correctly maintains data flow dependencies within a function and with called functions
  - Reorder the *execution* of code to optimize performance
- The processor can:
  - Reorder instruction execution as long as it correctly maintains register flow dependencies
  - Reorder memory modification as long as it correctly maintains data flow dependencies
  - Reorder the execution of instructions (for performance optimization)

90

Barrier Operations

- *barrier* – prevent only compiler reordering
- *mb* – prevents load and store reordering
- *rmb* – prevents load reordering
- *wmb* – prevents store reordering

- *smp_mb* – prevent load and store reordering only in SMP kernel
- *smp_rmb* – prevent load reordering only in SMP kernels
- *smp_wmb* – prevent store reordering only in SMP kernels
- *set_mb* – performs assignment and prevents load and store reordering

Atomic Operations

- Many instructions not atomic in hardware (smp)
  - Read-modify-write instructions: inc, test-and-set, swap
  - Unaligned memory access
- Compiler may not generate atomic code
  - Even i++ is not necessarily atomic!
- If the data that must be protected is a single word, atomic operations can be used. These functions examine and modify the word atomically.
- The atomic data type is **`atomic_t`.**

Atomic Ops

- Execute in a single instruction
- Can be used in or out of process context (i.e., softirqs)
- Never sleep
- Don't suspend interrupts

Atomic Operations

ATOMIC_INIT – initialize an *atomic_t* variable
atomic_read – examine value atomically
atomic_set – change value atomically
atomic_inc – increment value atomically
atomic_dec – decrement value atomically
atomic_add - add to value atomically
atomic_sub – subtract from value atomically
atomic_inc_and_test – increment value and test for zero
atomic_dec_and_test – decrement value and test for zero
atomic_sub_and_test – subtract from value and test for zero
atomic_set_mask – mask bits atomically
atomic_clear_mask – clear bits atomically

Atomic Bit Operations

Perform bit operations atomically. Can be done without disabling interrupts on most platforms.

set_bit	*change_bit*
clear_bit	*test_bit*
test_and_set_bit	*find_first_bit*
test_and_clear_bit	*find_first_zero_bit*
test_and_change_bit	*find_next_zero_bit*

There also exist non_atomic versions with __ prefix, e.g., *__set_bit.* These are slightly faster than the atomic versions.

Serializing with Interrupts

- Basic primitive in original UNIX
- Doesn't protect against other CPUs
- Intel: "interrupts enabled bit"
  - cli to clear (disable), sti to set (enable)
- Enabling is often wrong; need to restore
  - local_irq_save()
  - local_irq_restore()

Interrupt Operations

- Services used to serialize with interrupts are:
  - *local_irq_disable* - disables interrupts on the current CPU
  - *local_irq_enable* - enable interrupts on the current CPU
  - *local_save_flags* - return the interrupt state of the processor
  - *local_restore_flags* - restore the interrupt state of the processor
- Dealing with the full interrupt state of the system is officially discouraged. Locks should be used.

IRQ request and enable

- request_irq()
- disable_irq()
- disable_irq_nosync()
- enable_irq()

Interrupt bottom half

- softirq/ksoftirqd
- tasklet
- work queue

Threaded_irq

- int **request_threaded_irq** (unsigned int *irq*, irq_handler_t *handler*, irq_handler_t *thread_fn*, unsigned long *irqflags*, const char * *devname*, void * *dev_id*);

ARM GIC

- PPI : Private Peripheral Interrupt

specific to a single processor

- SPI : Shared Peripheral Interrupt

Distributor can route to any of a specified combination of processors

- Software-generated interrupt (SGI)

This is an interrupt generated by software writing to a GICD_SGIR register in the GIC. The system uses SGIs for interprocessor communication

- ID0-ID15 are used for SGIs
- ID16-ID31 are used for PPIs
- ID32+ are used for SPIs

gic_raise_softirq

- void gic_raise_softirq(const struct cpumask *mask, unsigned int irq)

- Platform:

set_smp_cross_call(gic_raise_softirq);

- Kernel:

smp_cross_call(cpumask_of(cpu), IPI_RESCHEDULE);

ARM Linux IPI

enum ipi_msg_type
{
 __IPI_WAKEUP,
 __IPI_TIMER,
 __IPI_RESCHEDULE,
 __IPI_CALL_FUNC,
 __IPI_CALL_FUNC_SINGLE,
 __IPI_CPU_STOP,
};

IRQ affinity

- Kernel space

irq_set_affinity(I2C0_IRQ, cpumask_of(0));
irq_set_affinity(GPIO0_IRQ, cpumask_of(1));

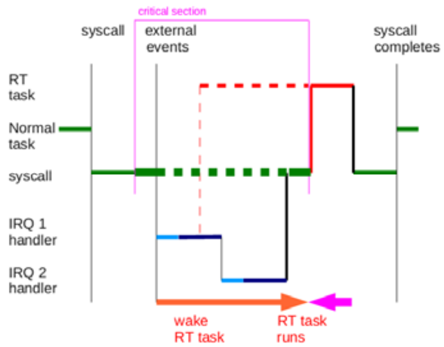
- Userspace

[root@boss ~]# echo 01 > /proc/irq/145/smp_affinity
[root@boss ~]# cat /proc/irq/145/smp_affinity
00000001

Spin Locks

- A spin lock is a data structure (*spinlock_t*) that is used to synchronize access to critical sections.
- Only one thread can be holding a spin lock at any moment. All other threads trying to get the lock will "spin" (loop while checking the lock status).
- Spin locks should not be held for long periods because waiting tasks on other CPUs are spinning, and thus wasting CPU execution time.

Spinlock in Preemptible Kernel



Spin Locks & Interrupts

- The spin lock services also provide interfaces that serialize with interrupts (on the current processor):
  - *spin_lock_irq* - acquire spin lock and disable interrupts
  - *spin_unlock_irq* - release spin lock and reenable
  - *spin_lock_irqsave* - acquire spin lock, save interrupt state, and disable
  - *spin_unlock_irqrestore* - release spin lock and restore interrupt state

High performance, low power spinlocks

```
static inline void _raw_spin_lock(spinlock_t *lock)
{
    unsigned long tmp;

    asm__ __volatile__(
" 1:      ldrex     %0, [%1]\n"                      ; exclusive read lock
"         teq       %0, #0\n"            ; check if free
"         wfene \n"                      ; if not, wait (saves power)
"         strexeq   %0, %2, [%1]\n"                  ; attempt to store to the lock
"         teqeq     %0, #0\n"            ; Were we successful ?
"         bne       1b"                              ; no, try again
   :      "=&r" (tmp)
   :      "r" (&lock->lock), "r" (1), "r" (0)
   :      "cc", "memory");

    rmb();          // Read data memory barrier, Stops WO reads << lock write
}                   // This is NOP on MPCore since dependent reads are sync'ed

static inline void _raw_spin_unlock(spinlock_t *lock)
{
    wmb();          // data write memory barrier, Ensure payload write visible
                    // Ensure data ordering, but does not necessarily wait
   _asm__ __volatile__(
"         str       %1, [%0]\n"                      ; Release, invalidates any LDREX
"         mcr       p15, 0, %1, c7, c10, 4\n"        ; DrainStoreBuffer (flushes to RAM)
"         sev       \n"                              ; Signal to any CPU waiting
   : "r" (&lock->lock), "r" (0)
   : "cc", "memory");
}
```

RW Spin Locks

- A read/write spin lock is a data structure that allows multiple tasks to hold it in "read" state or one task to hold it in "write" state (but not both conditions at the same time).
- This is convenient when multiple tasks wish to examine a data structure, but don't want to see it in an inconsistent state.
- A lock may not be held in read state when requesting it for write state.
- The data type for a read/write spin lock is *rwlock_t*.
- Writers can starve waiting behind readers.

RW Spin Lock Operations

- Several functions are used to work with read/write spin locks:
  - *rwlock_init* – initialize a read/write lock before using it for the first time
  - *read_lock* – get a read/write lock for read
  - *write_lock* – get a read/write lock for write
  - *read_unlock* – release a read/write lock that was held for read
  - *write_unlock* – release a read/write lock that was held for write
  - *read_trylock, write_trylock* – acquire a read/write lock if it is currently free, otherwise return error

RW Spin Locks & Interrupts

- The read/write lock services also provide interfaces that serialize with interrupts (on the current processor):
  - *read_lock_irq* - acquire lock for read and disable interrupts
  - *read_unlock_irq* - release read lock and reenable
  - *read_lock_irqsave* - acquire lock for read, save interrupt state, and disable
  - *read_unlock_irqrestore* - release read lock and restore interrupt state
- Corresponding functions for write exist as well (e.g., *write_lock_irqsave*).

spin_lock for MP

- Process context: spin_lock_irqsave
- IRQ context: spin_lock

```
static irqreturn_t xiic_isr(int irq, void *dev_id)
{
        struct xiic_i2c *i2c = dev_id;
         spin_lock(&i2c->lock);          (spin in other cores)
        /* disable interrupts globally */
        xiic_setreg32(i2c, XIIC_DGIER_OFFSET, 0);
        …
         spin_unlock(&i2c->lock);
        return IRQ_HANDLED;
}
```

```
static void xiic_start_xfer(struct xiic_i2c *i2c)
{
        unsigned long flags;

        spin_lock_irqsave(&i2c->lock, flags); (mask local irq in one core)
        xiic_reinit(i2c);
        /* disable interrupts globally */
        xiic_setreg32(i2c, XIIC_DGIER_OFFSET, 0);
         spin_unlock_irqrestore(&i2c->lock, flags);

        __xiic_start_xfer(i2c);
        xiic_setreg32(i2c, XIIC_DGIER_OFFSET, XIIC_GINTR_ENABLE_MASK);
}
```

Semaphores

- A *semaphore* is a data structure that is used to synchronize access to critical sections or other resources.
- A *semaphore* allows a fixed number of tasks (generally one for critical sections) to "hold" the semaphore at one time. Any more tasks requesting to hold the *semaphore* are blocked (put to sleep).
- A *semaphore* can be used for serialization only in code that is allowed to block.

Semaphore Operations

- Operations for manipulating semaphores:
  - *up* – release the semaphore
  - *down* – get the semaphore (can block)
  - *down_interruptible* – get the semaphore, but return whether we blocked
  - *down_trylock* – try to get the semaphore without blocking, otherwise return an error

Semaphores

- optimized assembly code for normal case (down())
  - C code for slower "contended" case (__down())
- up() is easy
  - atomically increment; wake_up() if necessary
- uncontended down() is easy
  - atomically decrement; continue
- contended down() is really complex!
  - basically increment sleepers and sleep
  - loop because of potentially concurrent ups/downs
- still in down() path when lock is acquired

115

Mutexes

- A *mutex* is a data structure that is *also* used to synchronize access to critical sections or other resources, introduced in 2.6.16.
- Core difference: only 1 owner, while semaphores can have multiple owners
- Historically, semaphores have been used in the kernel, but now mutexes are encouraged, unless counting feature is really required
- As of 2.6.26, major effort to eliminate semaphores completely, and may eventually disappear
- Replace remaining instances with *completions*

Why Mutexes?

`Documentation/mutex-design.txt`

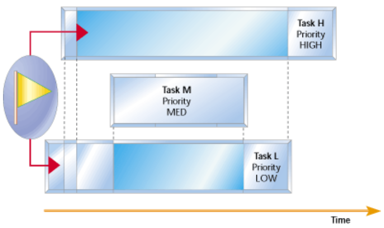
- Pros
  - Simpler (lighter weight)
  - Tighter code
  - Slightly faster, better scalability
  - No fastpath tradeoffs
  - Debug support – strict checking of adhering to semantics (if compiled in)
- Cons
  - Not the same as semaphores
  - Cannot be used from interrupt context
  - Owner must release

Mutex Operations

- Operations for manipulating mutexes:
  - *mutex_unlock* – release the mutex
  - *mutex_lock* – get the mutex (can block)
  - *mutex_lock_interruptible* – get the mutex, but allow interrupts
  - *mutex_trylock* – try to get the mutex without blocking, otherwise return an error
  - *mutex_is_locked* – determine if mutex is locked

Priority inversions

- when a medium-priority task preempts a lower-priority task using a shared resource on which the higher-priority task is pending. If the higher-priority task is otherwise ready to run, but a medium-priority task is currently running instead, a priority inversion is said to occur.



Priority ceiling protocol

- give each shared resource a predefined priority ceiling. When a task acquires a shared resource, the task is hoisted (has its priority temporarily raised) to the priority ceiling of that resource. It will not see whether the job has been blocked or not, simply it raises to the priority of the shared resource.

RT-mutex - priority inheritance

- RT-mutexes extend the semantics of simple mutexes by the priority inheritance protocol.
- A low priority owner of a rt-mutex inherits the priority of a higher priority waiter until the rt-mutex is released. If the temporarily boosted owner blocks on a rt-mutex itself it propagates the priority boosting to the owner of the other rt_mutex it gets blocked on. The priority boosting is immediately removed once the rt_mutex has been unlocked.

rt_mutex - APIs

- *void rt_mutex_init(struct rt_mutex *lock);*
- *void rt_mutex_destroy(struct rt_mutex *lock);*
- *void rt_mutex_lock(struct rt_mutex *lock);*
- *int rt_mutex_lock_interruptible(struct rt_mutex *lock, int detect_deadlock);*
- *int rt_mutex_timed_lock(struct rt_mutex *lock, struct hrtimer_sleeper *timeout,int detect_deadlock);*
- *int rt_mutex_trylock(struct rt_mutex *lock);*
- *void rt_mutex_unlock(struct rt_mutex *lock);*
- *int rt_mutex_is_locked(struct rt_mutex *lock);*

PI-futex - Lightweight PI-futexes

- *in the user-space fastpath a PI-enabled futex involves no kernel work (or any other PI complexity) at all. No registration, no extra kernel calls - just pure fast atomic ops in userspace.*
- *even in the slowpath, the system call and scheduling pattern is very similar to normal futexes.*
- *the in-kernel PI implementation is streamlined around the mutex abstraction, with strict rules that keep the implementation relatively simple: only a single owner may own a lock (i.e. no read-write lock support), only the owner may unlock a lock, no recursive locking, etc.*

PTHREAD_PRIO_INHERIT

- *A real-time system cannot be real-time if there is no solution for priority inversion, this will cause undesired latencies and even deadlocks. On Linux there is a solution for it in user-land since kernel version 2.6.18 together with Glibc 2.5 (PTHREAD_PRIO_INHERIT).*
- *int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr, int protocol);*
  - ✓ *PTHREAD_PRIO_NONE: A thread's priority and scheduling are not affected by the mutex ownership.*
  - ✓ *PTHREAD_PRIO_INHERIT: This protocol value affects a thread's (such as thrd1) priority and scheduling when higher-priority threads block on one or more mutexes owned by thrd1 where those mutexes are initialized with PTHREAD_PRIO_INHERIT. thrd1 runs with the higher of its priority or the highest priority of any thread waiting on any of the mutexes owned by thrd1.*

Completions

- Higher-level means of waiting for events
- Optimized for contended case

init_completion	*// replaces sema_init*
complete	*// replaces up*
wait_for_completion	*// replaces down*
wait_for_completion_interruptible	
wait_for_completion_timeout	
wait_for_completion_interruptable_timeout	

125

The Big Kernel Lock (BKL)

- For serialization that is not performance sensitive, the big kernel lock (BKL) was used
  - This mechanism is historical and should generally be avoided.
  - The function *lock_kernel* gets the big kernel lock.
  - The function *unlock_kernel* releases the big kernel lock.
  - The function *kernel_locked* returns whether the kernel lock is currently held by the current task.
  - The big kernel lock itself is a simple lock called *kernel_flag*.

2.6.39: Ding Dong, the Big Kernel Lock is Dead

- The Big Kernel Lock was almost removed in the 2.6.37 kernel. With the 2.6.39 kernel, the BKL is finally gone with a patch from Arnd Bergmann.
- The big kernel lock (BKL) is an old serialization method that we are trying to get rid of, replacing it with more fine-grained locking, in particular mutex, spinlock and RCU, where appropriate.

Jiffies

- In `/usr/src/linux/include/linux/jiffies.h`:
  - `unsigned long volatile jiffies`
- Until 2.6.21, jiffies was just a counter that was incremented every clock interrupt
- Tickless kernels removed need for kernels to process timer interrupts on idle systems
  - http://www.lesswatts.org/projects/tickless

HZ

- Determines how frequently the clock interrupt fires
- Default is 1000 on x86, or 1 millisecond
- Configurable at compile time or boot time
  - Other typical values are 100 (10 ms) or 25 (4)
- What's a good value for HZ?
  - Low values: less overhead
    - Good for idle systems (power)
    - Good for busy systems (thpt)
  - High values: better resolution
    - Ability to retake control in scheduler (clock interrrupt)
    - Good for interactivity
  - Tradeoff between responsiveness and throughput, running debate

Jiffies and HZ

- Again, incremented HZ times a second
- Jiffies can wrap around depending on platform
  - 32 bits, 1000 HZ: about 50 days
  - 64 bits, 1000 HZ: about 600 million years
- Jiffies_64:
  - On 32 bits, jiffies points to low-order 32 bits, jiffies_64 to high-order bits (be careful about atomicity!)
  - On 64 bit machines, jiffies == jiffies_64

Some Useful Functions

- Macros to compare jiffies values:
  - `time_after(a,b)`
  - `time_before(a,b)`
  - `time_after_eq(a,b)`
  - `time_before(a,b)`
- Macros to convert jiffies to/from others:
  - `timespec_to_jiffes(struct timespec * ts)`
  - `jiffies_to_timespec(unsigned long jiffies, struct timespec * ts)`
  - `timeval_to_jiffes(struct timeval * tv)`
  - `jiffies_to_timeval(unsigned long jiffies, struct timeval * tv)`
  - `jiffies_to_msecs(unsigned long)`
  - `msecs_to_jiffies(int)`
  - `jiffies_to_usecs(unsigned long)`
  - `usecs_to_jiffies(int)`

Inserting Delays

- `#include <linux/delay>`
- `void mdelay(unsigned long milliseconds)`
- `void udelay(unsigned long microseconds)`
- `void ndelay(unsigned long nanoseconds)`
- `void msleep(unsigned int milliseconds)`
- `unsigned long msleep_interruptible(unsigned int milliseconds)`
  - returns number of seconds left if interrupted

Timers

- Runs via softirq like tasklets, but at a specific time
- A timer is represented by a timer_list:

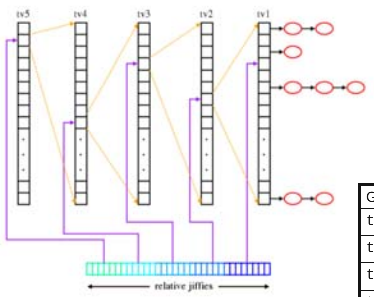
```
struct timer_list {
  struct list_head entry; /* dbly linked */
  unsigned long expires; /* In jiffies */
  void (*function)(unsigned long);
  unsigned long data; /* optional */
  struct tvec_t_base_s *base;
};
```

`expires` is an absolute value, not a relative one

Timer Operations

- `void init_timer(struct timer_list *timer);` — `// clears prev and next ptrs`
- `void add_timer(struct timer_list *timer);` — `// inserts into global timer list`
- `int del_timer(struct timer_list *timer);` — `// returns 1 if timer deleted, or 0 if too late`
- `int del_timer_sync(struct timer_list *timer);` — `// difference is makes sure timer is not running on any other CPUs`
- `void mod_timer(struct timer_list *timer, unsigned long expires);` — `// Changes expiry to new value`
- `int timer_pending(const struct timer_list *timer);` — `// 1 if timer is pending, 0 otherwise`

Timer Implementation



(from lwn.net)

Group	Ticks	Time
tv1	< 2^8	< .256 secs
tv2	< 2^14	< 16.4 secs
tv3	< 2^20	< 17.5 mins
tv4	< 2^26	< 18 hrs
tv5	< Inf	< Inf

High Resolution Timers

- Motivated by the observation of 2 types of timers:
  - *Timeout* functions, which we don't expect to actually happen (e.g., retransmisison timer for packet loss). Have low resolution and are usually removed before expiration.
  - *Timer* functions, which we do expect to run. Have high resolution requirements and usually expire
- Original timer implementation is based on jiffies and thus depends on HZ. Works well for timeouts, less so for timers.
  - Resolution no better than HZ (e.g., 1 millisecond)
- *High resolution timers,* introduced in 2.6.16, allow 1 nanosecond resolution
- Implemented in an red-black tree (rbtree)
  - Insert, delete, search in O(log *n*) time

ktime Kernel Time

```
#include <linux/ktime.h>

    union ktime {
      s64 tv64;             // in nanoseconds
    }
```

- `ktime_t ktime_add(ktime_t kt1, ktime_t kt2);`
- `ktime_t ktime_sub(ktime_t kt1, ktime_t kt2);`
- `ktime_t ktime_add_ns(ktime_t kt, u64 nanoseconds);`

- `ktime_t timespec_to_ktime(struct timespec tspec);`
- `ktime_t timeval_to_ktime(struct timeval tval);`
- `struct timespec ktime_to_timespec(ktime_t kt);`
- `struct timeval ktime_to_timeval(ktime_t kt);`
- `clock_t ktime_to_clock_t(ktime_t kt);`
- `u64 ktime_to_ns(ktime_t kt);`

High Res Timer Functions

- `extern void hrtimer_init(struct hrtimer *timer, clockid_t which_clock, enum hrtimer_mode mode);` — `// initialize hrtimer`
- `int hrtimer_cancel(struct hrtimer *timer);` — `// cancel timer (wait if necessary)`
- `int hrtimer_try_to_cancel(struct hrtimer *timer);` — `// cancel timer (don't wait)`
- `extern ktime_t hrtimer_get_remaining(const struct hrtimer *timer);` — `// time left before it expires`
- `extern int hrtimer_get_res(const clockid_t which_clock, struct timespec *tp);` — `// get resolution in ns`
- `static inline int hrtimer_active(const struct hrtimer *timer)` — `// 1 if active, 0 otherwise`

Restarting Timers

- Many timers want to restart
- Function passed to hrtimer_start needs to return a particular value:

```
      enum hrtimer_restart {
         HRTIMER_NORESTART,      /* Timer is not restarted */
         HRTIMER_RESTART,        /* Timer must be restarted */
      };
```

- **RESTART** used by functions that need a callback at a regular interval.
- The hrtimer code provides a function for advancing the expiration time to the next such interval:

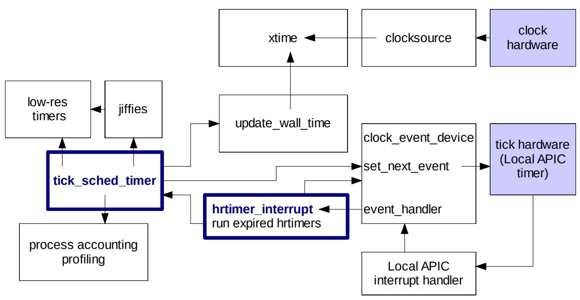
```
unsigned long hrtimer_forward(struct hrtimer *timer, ktime_t
   interval);
```

- Advances the timer's expiration time by the given interval.
- The need to add the interval more than once usually means that the system has overrun its timer period, perhaps as a result of high system load.
- The return value from **`hrtimer_forward()`** is the number of missed intervals, allowing code which cares to detect and respond to the situation.

Dyntick/``Tickless'' Kernel

- Observation: there is no need for a periodic tick in the system, particularly when the system is idle
  - Idle CPUs save power
  - Why wake up an idle CPU 1000 times a second?
  - Especially if it is virtualized?
- Instead, when going into idle loop, check next pending timer:
  - If event is further away than one tick, re-program clock timer to fire when that event is due
  - Other interrupts can still wake up the system
- Added in 2.6.21
  - **`CONFIG_NO_HZ`**
- New function added:
  - **`void init_timer_deferrable(struct timer_list *timer);`**
  - These timers are ignored for purposes of calculating wakeup time

Putting It All Together



hrtimer -users

- *The primary users of precision timers are user-space applications that utilize nanosleep, posix-timers and itimer interfaces. Also, in-kernel users like drivers and subsystems which require precise timed events(e.g. multimedia) can benefit from the availability of a separate high-resolution timer subsystem as well.*
- *The hrtimer patch converts the following kernel functionality to usehrtimers:*

- nanosleep
- Itimers
- posix-timers

The conversion of nanosleep and posix-timers enabled the unification of nanosleep and clock_nanosleep.

Is it still ticking?

- The timer tick is now an hrtimer
  - This allows it it coexist with the other hrtimers
  - Can be programmed in a flexible way, e.g. can be delayed for a while when the system is idle. (NO_HZ)
- So, is it still ticking?
  - hrtimers will make the hardware tick
  - Timer ticks, however, will happen only if needed, therefore less frequently

What' s Your Kernel Running?

- grep for HZ in the .config for your kernel
  - What is the value of CONFIG_HZ?
  - Is CONFIG_NOHZ set? (tickless kernel)
- cat /proc/interrupts
  - Look at the timer interrupt (irq 0). If you see a low number at irq 0, one of the local APICs is handling the global timer tick duties (local timer interrupts)
  - If CONFIG_NOHZ is not set, you can divide the number of timer or local timer interrupts by the number of seconds uptime, the result should be larger than HZ.
  - If CONFIG_NOHZ is set the number of timer interrupts will most likely be lower

Wait Queue

- ***Wait queues*** implement ***conditional waits*** on ***events***:
  - a process wishing to wait for a specific event
    - places itself in the proper wait queue

      and
    - relinquishes control.
- Therefore, a wait queue represents ***a set of sleeping processes***, which are woken up by the kernel when some condition becomes true.
- The condition could be related to:
  - an interrupt, such as for a disk operation to terminate
  - process synchronization
  - timing: such as a fixed interval of time to elapse

145

Wait Queue Implementation

- ***Wait queues*** are implemented as *doubly linked lists* whose elements include pointers to process descriptors.
- Each wait queue is identified by a ***wait queue head***, a data structure of type `wait_queue_head_t`:

```
struct __wait_queue_head
{     spinlock_t lock;
      struct list_head task_list;
};
typedef struct  __wait_queue_head   \
wait_queue_head_t;
```

for synchronization

the head of the list of waiting processes

146

Wait Queue Synchronization

- Since wait queues are modified
  - by ***interrupt handlers***

    as well
  - by major ***kernel functions***,

  the doubly linked lists must be protected from concurrent accesses, which could induce unpredictable results.
- Synchronization is achieved by the **`lock`** ***spin lock*** in the ***wait queue head***.

147

Declare a New Wait Queue Head

- A new wait queue head may be defined by using the `DECLARE_WAIT_QUEUE_HEAD(name)` macro, which
  - statically declares a new wait queue head variable called **`name`**

    and
  - initializes its **`lock`** and **`task_list`** fields.

```
#define __WAIT_QUEUE_HEAD_INITIALIZER(name) {             \
.lock = SPIN_LOCK_UNLOCKED,                                \
.task_list = { &(name).task_list, &(name).task_list } }

#define DECLARE_WAIT_QUEUE_HEAD(name)                      \
wait_queue_head_t name=__WAIT_QUEUE_HEAD_INITIALIZER(name)
```

148

Declare a New Wait Queue Element

- Alternatively, the `DEFINE_WAIT` macro:
  - declares a new **`wait_queue_t`** variable.
  - initializes it with the descriptor of the process currently executing on the **CPU.**

149

Functions That Can Put a Process to a Wait Queue

- A ***process*** wishing to wait for a specific ***condition*** can invoke any of the functions shown in the following list.
  - **`sleep_on( )`**
  - **`interruptible_sleep_on( )`**
  - **`sleep_on_timeout( )`**
  - **`interruptible_sleep_on_timeout( )`**
  - **`wait_event`** and **`wait_event_interruptible`** macros

150

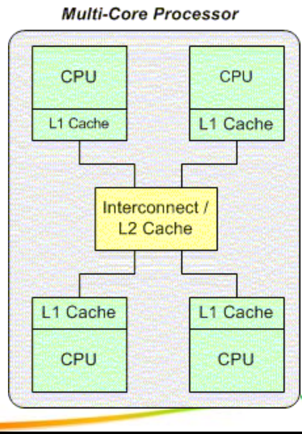
The sleep_on() function

```
void sleep_on(wait_queue_head_t *wq)
{ wait_queue_t wait;

  init_waitqueue_entry(&wait, current);
  current->state = TASK_UNINTERRUPTIBLE;
  add_wait_queue(wq, &wait);
  /* wq points to the wait queue head */
  schedule( );
  remove_wait_queue(wq, &wait);
}
```

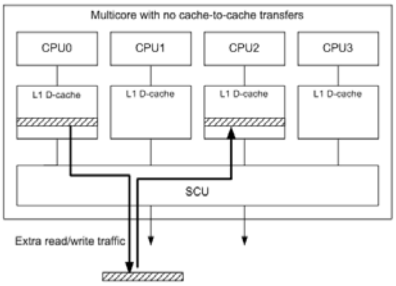
(1) Remove **current** from the runqueue.

(2) In order to make **schedule()** resume its execution, there must be some other kernel control path setting this process back to **TASK_RUNNING** state and putting it back to the runqueue after (1) is executed.

ARM SMP L1&L2 cache

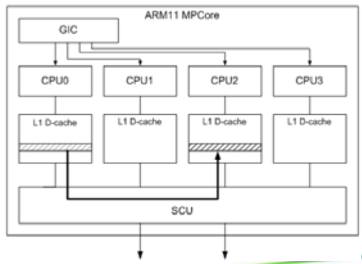


Multicore without cache-to-cache transfers



SCU(MESI protocol)

- Direct Data Intervention (DDI): The SCU keeps a copy of all cores caches' tag RAMs. This enables it to efficiently detect if a cache line request by a core is in another core in the coherency domain before looking for it in the next level of the memory hierarchy.
- Cache-to-cache Migration: If the SCU finds that the cache line requested by one CPU is present in another core, it will either copy it (if clean) or move it (if dirty) from the other CPU directly into the requesting one, without interacting with external memory.



DMA coherent(ARM11 MPCore)

- CP15 cache maintenance operations on the current CPU don't affect the cache of the other CPU
  - Another CPU writes the data to a cached memory buffer. The DMA restart operation is done by the current CPU which does a cache clean operation. None of the modified cache lines on the other CPU are cleaned, causing stale data to be transferred.
  - Current CPU invalidates the cache before an incoming DMA transfer writes new data. The CPU then reads the transferred data from the cacheable memory location. If the cache of another CPU contains an address within the DMA buffer, the SCU may take the stale data directly from the cache on that CPU with no access to external memory.
  - Current CPU writes to the uncached mapping but cache lines in the modified state on the other CPU may be evicted to Level 2 cache or main memory, corrupting part of the data written by the current CPU.
  - An external device writes to memory, but cache lines in the modified state on another CPU may be evicted, over-writing parts of the data written by the external device.

DMA coherent(Cortex-A9 MPCore)

- On Cortex-A9 MPCore, cache maintenance operations can be broadcast by hardware to other CPUs in the inner shareable domain.
- The Accelerator Coherency Port (ACP) and I/O Coherency
  - The Accelerator Coherency Port (ACP) is a optional feature of Cortex-A9, which provides an 64-bit AXI slave port that can be connected to a DMA engine, providing the DMA access to the SCU of Cortex-A9.
  - Addresses on the ACP port are physical addresses which can be snooped by the SCU to provide full I/O coherency.
  - Reads on the ACP port will hit in any CPU's L1 D-cache, and writes on the ACP port will invalidate any stale data in L1 and write through to L2.

Using large dma-coherent buffers

- void *dma_alloc_coherent(struct device *dev, size_t size, dma_addr_t *dma_handle, gfp_t flag)
- void dma_free_coherent(struct device *dev, size_t size, void *cpu_addr, dma_addr_t dma_handle)

Streaming DMA mappings

- dma_addr_t dma_map_single(struct device *dev, void *cpu_addr, size_t size, enum dma_data_direction direction)
- void dma_unmap_single(struct device *dev, dma_addr_t dma_addr, size_t size, enum dma_data_direction direction)
- int dma_map_sg(struct device *dev, struct scatterlist *sg, int nents, enum dma_data_direction direction)
- void dma_unmap_sg(struct device *dev, struct scatterlist *sg, int nhwentries, enum dma_data_direction direction)

Infrastructural APIs

- Device tree based BSP and drivers model
  - Device tree
  - Common board files
  - Platform device
  - I²C client
  - SPI client
- New Infrastructural APIs
  - Pinctrl API
  - Clock API
  - DMA API
  - Irq API
  - GPIO API

Linux system call

Linux System Calls

- System calls are low level functions the operating system makes available to applications via a defined API (Application Programming Interface)
- System calls represent the *interface* the kernel presents to user applications.
- In Linux all low-level I/O is done by reading and writing file handles, regardless of what particular peripheral device is being accessed—a tape, a socket, even your terminal, they are all *files*.
- Low level I/O is performed by making *system calls.*

From a Wrapper Routine to a System Call

- Unix systems include several ***libraries of functions*** that provide **API**s to programmers.
- Some of the **API**s defined by the **`libc`** standard **C** library refer to ***wrapper routines*** (routines whose only purpose is to issue a ***system call***).
- Usually, each system call has a corresponding wrapper routine, which defines the **API** that application programs should employ.

162

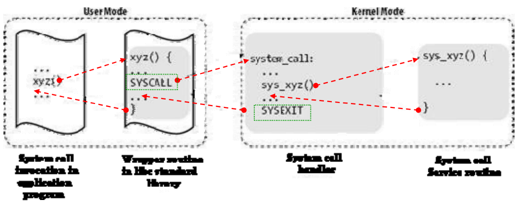
Example of Different APIs Issuing the Same System Call

- In Linux, the `malloc( )`, `calloc( )`, and `free( )` **API**s are implemented in the `libc` library.
- The code in this library keeps track of the allocation and deallocation requests and uses the `brk( )/mmap()` system call to enlarge or shrink the ***process heap***.

Operations Performed by a System Call

- The ***system call handler***, which has a structure similar to that of the other ***exception handlers***, performs the following operations:
  - Saves the contents of most registers in the Kernel Mode stack.
    - This operation is common to all system calls and is coded in assembly language.
  - Handles the system call by invoking a corresponding **C** function called the ***system call service routine***.
  - Exits from the handler:
    - the registers are loaded with the values saved in the Kernel Mode stack
    - the **CPU** is switched back from Kernel Mode to User Mode.
      - This operation is common to all system calls and is coded in assembly language.

Control Flow Diagram of a System Call



- The `arrows` denote the execution flow between the functions.
- The terms "**`SYSCALL`**" and "**`SYSEXIT`**" are placeholders for the actual assembly language instructions that switch the **CPU**, respectively, from User Mode to Kernel Mode and from Kernel Mode to User Mode.

System Call Dispatch Table

- To associate each ***system call number*** with its corresponding ***service routine***, the kernel uses a ***system call dispatch table***, which is stored in the `sys_call_table` array and has `NR_syscalls` entries.
- The **n**th entry contains the service routine address of the system call having number **n**.

Ways to Invoke a System Call on x86

- Applications can invoke a system call in two different ways:
  - By executing the `int $0x80` assembly language instruction; in older versions of the Linux kernel, this was the only way to switch from User Mode to Kernel Mode.
  - By executing the `sysenter` assembly language instruction, introduced in the Intel Pentium **II** microprocessors; this instruction is now supported by the Linux 2.6 kernel.

ARM EABI

- Old: The system call number was passed as part of the swi instruction, the kernel had to read and decode the swi instruction, polluting the data cache with instructions
- Now, the system call number is passed in r7
- CONFIG_OABI_COMPAT: In an EABIable kernel, provides compatibility with old ABI userspace binaries

Tracing System Calls

- Linux has a powerful mechanism for tracing system call execution for a compiled application
- Output is printed for each system call as it is executed, including parameters and return codes
- The ptrace() system call is used (same call used by debuggers for single-stepping applications)
- Use the “strace” command (man strace for info)
- You can trace library calls using the “ltrace” command

169

Slides derived from Phillip Hutto

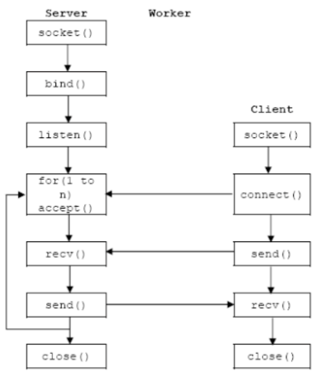
Linux socketcall

- **socketcall**() is a common kernel entry point for the socket system calls. *call* determines which socket function to invoke. *args* points to a block containing the actual arguments, which are passed through to the appropriate call. User programs should call the appropriate functions by their usual names. Only standard library implementors and kernel hackers need to know about **socketcall**().

int socketcall(int *call*, **unsigned long ***args***);**

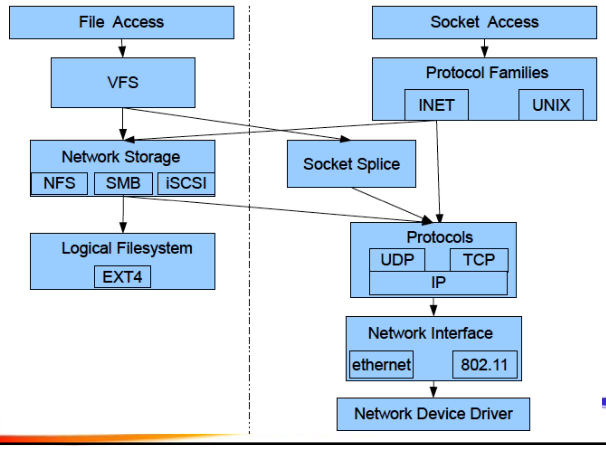
170

Network programming



171

Linux network architecture



Protocol family

- Implements different socket families INET, UNIX
- Socket Splice: Linux supports to send entire files between file descriptors, A descriptor can be a socket
- Protocols:Families have multiple protocols
  - INET: TCP, UDP
- pcap library for Linux uses PF_PACKET sockets

173