Linux 调试环境

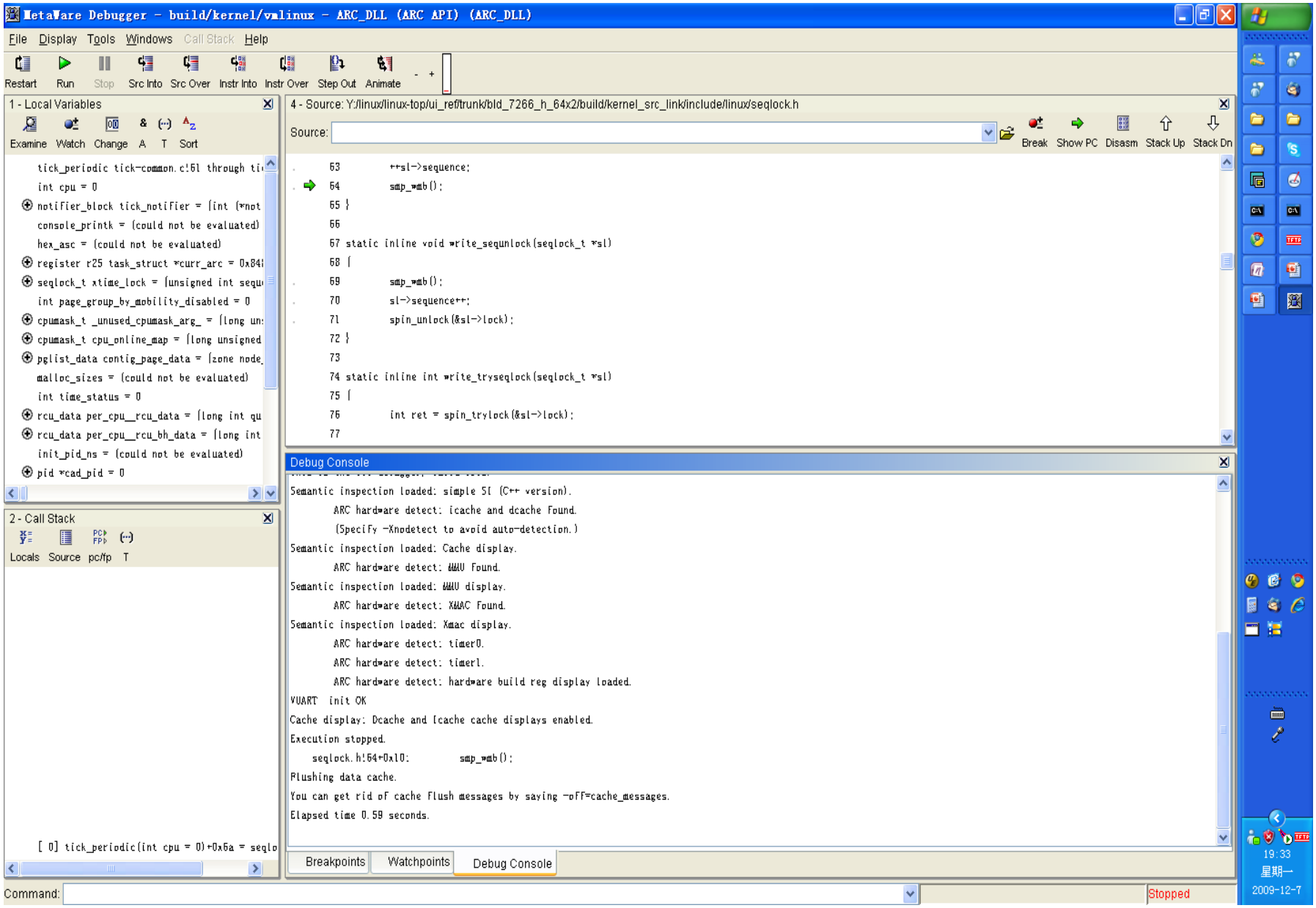
Amlogic Beijing
Zhouzhi
2009-12-7

主题

- Metaware 调试内核和驱动
- GDB/insight调试应用程序
- Procfs
- Oprofile优化代码

Metaware 调试内核和驱动

- 一般使用samba把编译的整个工程目录共享出来,然后在windows里面映射为一个网络驱动器;
- 确认安装好metaware和JTAG驱动,并使用JTAG连接到Metaware 调试内核和驱动板子上;
- 接通板子电源,并启动内核;
- 从windos上进入编译时的bld目录;运行wmake offdown 来启动metaware debugger

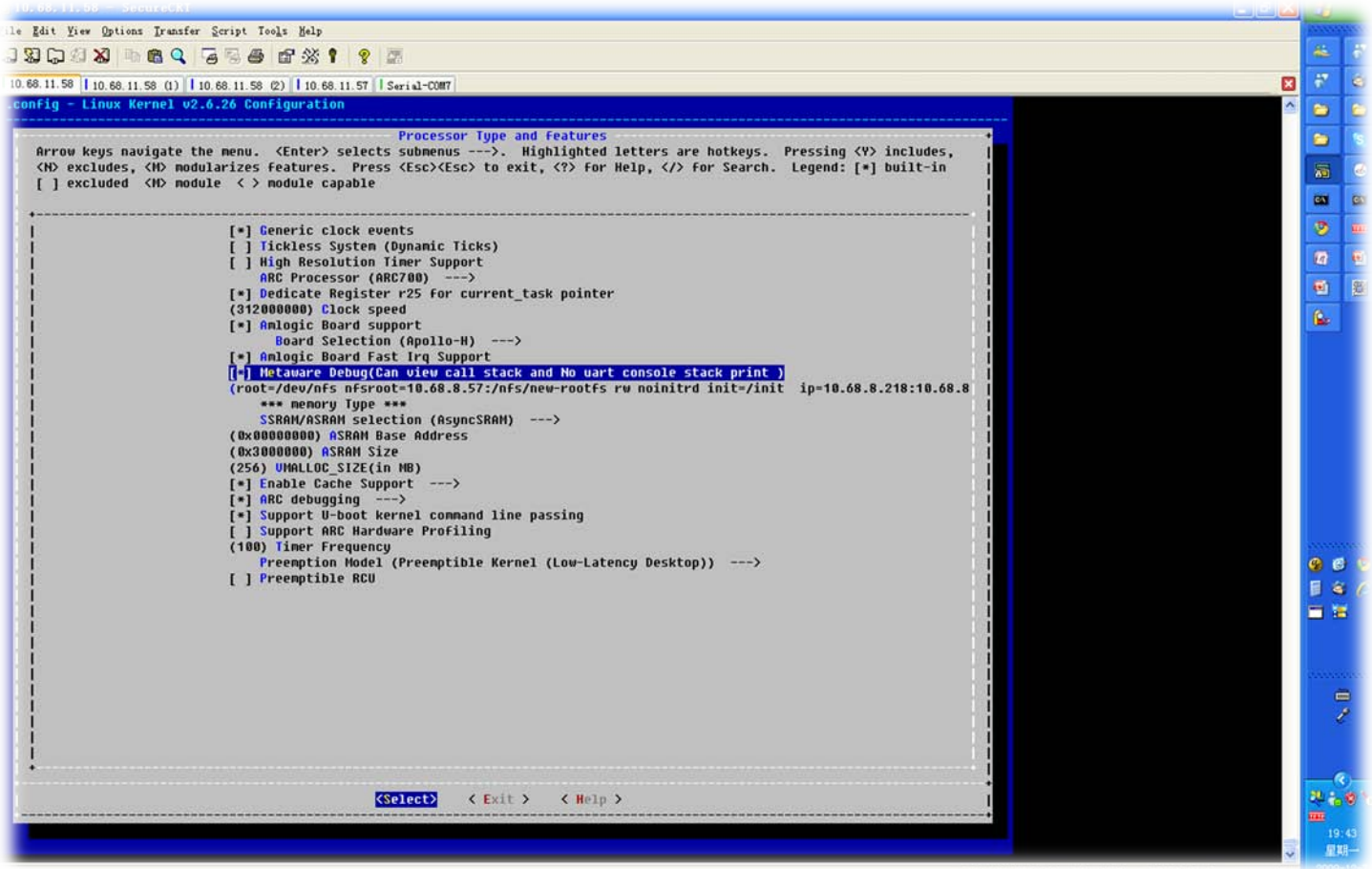

MetaWare Debugger - build/kernel/vmlinux - ARC_DLL (ARC API) (ARC_DLL)
File Display Tools Windows Call Stack Help
Restart Run Stop Src Into Src Over Instr Into Instr Over Step Out Animate
1 - Local Variables
Examine Watch Change A T Sort
tick_periodic tick-common.c!61 through ti
int cpu = 0
notifier_block tick_notifier = {int (*not
console_printk = {could not be evaluated}
hex_asc = {could not be evaluated}
register r25 task_struct *curr_arc = 0x84
seqlock_t xtime_lock = {unsigned int sequ
int page_group_by_mobility_disabled = 0
cpumask_t _unused_cpumask_arg_ = {long un
cpumask_t cpu_online_map = {long unsigned
pglist_data contig_page_data = {zone node_
malloc_sizes = {could not be evaluated}
int time_status = 0
rcu_data per_cpu__rcu_data = {long int qu
rcu_data per_cpu__rcu_bh_data = {long int
init_pid_ns = {could not be evaluated}
pid *cad_pid = 0
2 - Call Stack
Locals Source pc/fp T
[0] tick_periodic(int cpu = 0)+0x6a = seqlo
4 - Source: Y:/linux/linux-top/ui_ref/trunk/bld_7266_h_64x2/build/kernel_src_link/include/linux/seqlock.h
Source:
Break Show PC Disasm Stack Up Stack Dn
63 ++sl->sequence;
64 smp_wmb();
65 }
66
67 static inline void write_sequnlock(seqlock_t *sl)
68 {
69 smp_wmb();
70 sl->sequence++;
71 spin_unlock(&sl->lock);
72 }
73
74 static inline int write_tryseqlock(seqlock_t *sl)
75 {
76 int ret = spin_trylock(&sl->lock);
77
Debug Console
Semantic inspection loaded: simple SI (C++ version).
ARC hardware detect: icache and dcache found.
(Specify -Xnodetect to avoid auto-detection.)
Semantic inspection loaded: Cache display.
ARC hardware detect: MMU found.
Semantic inspection loaded: MMU display.
ARC hardware detect: XMAC found.
Semantic inspection loaded: Xmac display.
ARC hardware detect: timer0.
ARC hardware detect: timer1.
ARC hardware detect: hardware build reg display loaded.
VUART init OK
Cache display: Dcache and Icache cache displays enabled.
Execution stopped.
seqlock.h!64+0x10: smp_wmb();
Flushing data cache.
You can get rid of cache flush messages by saying -off=cache_messages.
Elapsed time 0.59 seconds.
Breakpoints Watchpoints Debug Console
Command:
Stopped
19:33
星期一
2009-12-7

Metaware 调试内核和驱动

- 使用标准的metaware可以查看和分析:
  - Step, assembe step
  - Source code
  - Assembler code
  - Local variable
  - Global variable
  - Watch Pointer　　#设置观测点,能够在变量或地址被访问或修改时,暂停执行,以便分析被错误修改情况;
  - Break Pointer　　#设置断点
  - HW Register　　#all the cpu registers,
  - Examine　　#可以查看分析变量,或把一个地址当作一个结构体地址来分析,以便很快的分析具体变量值;
  - Memory
  - 另外包括xymem,cache,timer,mmu,second cpu等分析;
  - Call Stack(分析Call stack 需要打开metaware支持,如下图)

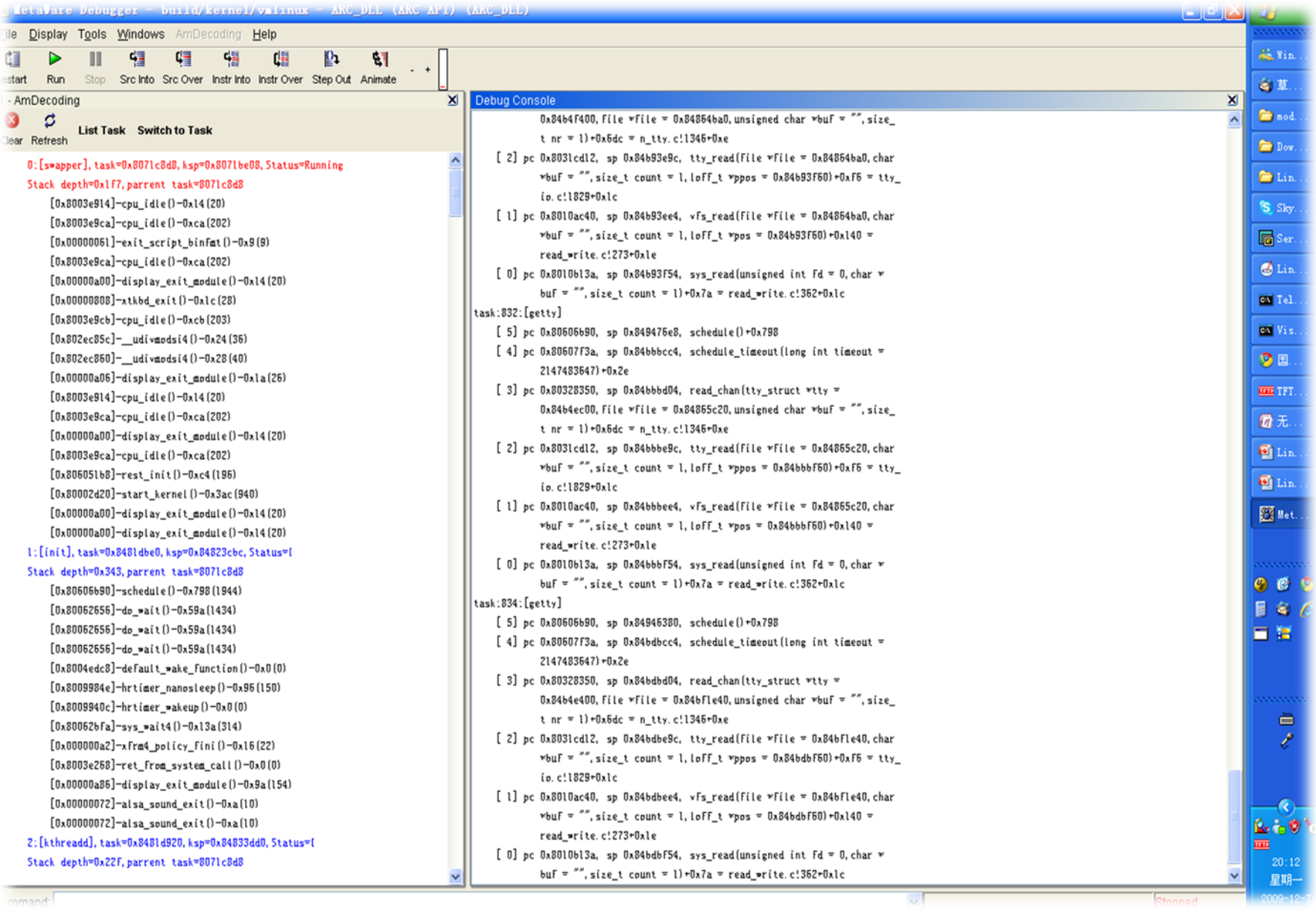
Metaware 调试内核和驱动

打开Metaware stack分析支持



Metaware 调试内核和驱动

- AmDecoding 分析内核thread的状态
  - 分析每个 thread的Call stack,所以需要在内核打开Metaware支持;
  - 分析各个thread的当前堆栈使用情况,
  - 获取各个thread的task_struct结构体地址,然后Examine窗口可以使用(struct task_struct *)(addr)来获取每个thread的详细信息;


MetaWare Debugger - build/kernel/vmlinux - ARC_DLL (ARC API) (ARC_DLL)
File Display Tools Windows AmDecoding Help
Start Run Stop Src Into Src Over Instr Into Instr Over Step Out Animate
- AmDecoding
Clear Refresh List Task Switch to Task
0:[swapper], task=0x8071c8d8, ksp=0x8071be08, Status=Running
Stack depth=0x1f7, parrent task=8071c8d8
[0x8003e914]-cpu_idle()-0x14(20)
[0x8003e9ca]-cpu_idle()-0xca(202)
[0x00000061]-exit_script_binfmt()-0x9(9)
[0x8003e9ca]-cpu_idle()-0xca(202)
[0x00000a00]-display_exit_module()-0x14(20)
[0x00000808]-xtkbd_exit()-0x1c(28)
[0x8003e9cb]-cpu_idle()-0xcb(203)
[0x802ec85c]-__udivmodsi4()-0x24(36)
[0x802ec860]-__udivmodsi4()-0x28(40)
[0x00000a06]-display_exit_module()-0x1a(26)
[0x8003e914]-cpu_idle()-0x14(20)
[0x8003e9ca]-cpu_idle()-0xca(202)
[0x00000a00]-display_exit_module()-0x14(20)
[0x8003e9ca]-cpu_idle()-0xca(202)
[0x806051b8]-rest_init()-0xc4(196)
[0x80002d20]-start_kernel()-0x3ac(940)
[0x00000a00]-display_exit_module()-0x14(20)
[0x00000a00]-display_exit_module()-0x14(20)
1:[init], task=0x8481dbe0, ksp=0x84823cbc, Status=[
Stack depth=0x343, parrent task=8071c8d8
[0x80606b90]-schedule()-0x798(1944)
[0x80062656]-do_wait()-0x59a(1434)
[0x80062656]-do_wait()-0x59a(1434)
[0x80062656]-do_wait()-0x59a(1434)
[0x8004edc8]-default_wake_function()-0x0(0)
[0x8009984e]-hrtimer_nanosleep()-0x96(150)
[0x8009940c]-hrtimer_wakeup()-0x0(0)
[0x80062bfa]-sys_wait4()-0x13a(314)
[0x000000a2]-xfrm4_policy_fini()-0x16(22)
[0x8003e268]-ret_from_system_call()-0x0(0)
[0x00000a86]-display_exit_module()-0x9a(154)
[0x00000072]-alsa_sound_exit()-0xa(10)
[0x00000072]-alsa_sound_exit()-0xa(10)
2:[kthreadd], task=0x8481d920, ksp=0x84833dd0, Status=[
Stack depth=0x22f, parrent task=8071c8d8
Debug Console
0x84b4f400, file *file = 0x84864ba0, unsigned char *buf = "", size_
t nr = 1)+0x6dc = n_tty.c!1346+0xe
[2] pc 0x8031cdl2, sp 0x84b93e9c, tty_read(file *file = 0x84864ba0, char
*buf = "", size_t count = 1, loff_t *ppos = 0x84b93f60)+0xf6 = tty_
io.c!1829+0x1c
[1] pc 0x8010ac40, sp 0x84b93ee4, vfs_read(file *file = 0x84864ba0, char
*buf = "", size_t count = 1, loff_t *pos = 0x84b93f60)+0x140 =
read_write.c!273+0x1e
[0] pc 0x8010b13a, sp 0x84b93f54, sys_read(unsigned int fd = 0, char *
buf = "", size_t count = 1)+0x7a = read_write.c!362+0x1c
task:832:[getty]
[5] pc 0x80606b90, sp 0x849476e8, schedule()+0x798
[4] pc 0x80607f3a, sp 0x84bbbcc4, schedule_timeout(long int timeout =
2147483647)+0x2e
[3] pc 0x80328350, sp 0x84bbbd04, read_chan(tty_struct *tty =
0x84b4ec00, file *file = 0x84865c20, unsigned char *buf = "", size_
t nr = 1)+0x6dc = n_tty.c!1346+0xe
[2] pc 0x8031cdl2, sp 0x84bbbe9c, tty_read(file *file = 0x84865c20, char
*buf = "", size_t count = 1, loff_t *ppos = 0x84bbbf60)+0xf6 = tty_
io.c!1829+0x1c
[1] pc 0x8010ac40, sp 0x84bbbee4, vfs_read(file *file = 0x84865c20, char
*buf = "", size_t count = 1, loff_t *pos = 0x84bbbf60)+0x140 =
read_write.c!273+0x1e
[0] pc 0x8010b13a, sp 0x84bbbf54, sys_read(unsigned int fd = 0, char *
buf = "", size_t count = 1)+0x7a = read_write.c!362+0x1c
task:834:[getty]
[5] pc 0x80606b90, sp 0x84946380, schedule()+0x798
[4] pc 0x80607f3a, sp 0x84bdbcc4, schedule_timeout(long int timeout =
2147483647)+0x2e
[3] pc 0x80328350, sp 0x84bdbd04, read_chan(tty_struct *tty =
0x84b4e400, file *file = 0x84bfle40, unsigned char *buf = "", size_
t nr = 1)+0x6dc = n_tty.c!1346+0xe
[2] pc 0x8031cdl2, sp 0x84bdbe9c, tty_read(file *file = 0x84bfle40, char
*buf = "", size_t count = 1, loff_t *ppos = 0x84bdbf60)+0xf6 = tty_
io.c!1829+0x1c
[1] pc 0x8010ac40, sp 0x84bdbee4, vfs_read(file *file = 0x84bfle40, char
*buf = "", size_t count = 1, loff_t *pos = 0x84bdbf60)+0x140 =
read_write.c!273+0x1e
[0] pc 0x8010b13a, sp 0x84bdbf54, sys_read(unsigned int fd = 0, char *
buf = "", size_t count = 1)+0x7a = read_write.c!362+0x1c
20:12
星期一

GDB/insight调试应用程序

- GDB是GNU开源组织发布的一个强大的UNIX下的程序调试工具

gdb a.out
(Gdb) set solib-absolute-prefix NFSROOT
(Gdb)target remote 10.68.11.128:1234
(gdb)list
(gdb)next
(gdb)step
(Gdb)bt
(gdb)print

Gdbserver host:1234 a.out

Gdbserver host:1234 --attach

GDB/insight调试应用程序

- 常用的gdb命令
  - backtrace 显示程序中的当前位置和表示如何到达当前位置的栈跟踪（同义词：where）
    breakpoint 在程序中设置一个断点
    cd 改变当前工作目录
    clear 删除刚才停止处的断点
    commands 命中断点时，列出将要执行的命令
    continue 从断点开始继续执行
    delete 删除一个断点或监测点；也可与其他命令一起使用
    display 程序停止时显示变量和表达时
    down 下移栈帧，使得另一个函数成为当前函数
    frame 选择下一条continue命令的帧
    info 显示与该程序有关的各种信息
    jump 在源程序中的另一点开始运行
    kill 异常终止在gdb 控制下运行的程序
    （即数据断点）
    whatis 显示变量或函数类型

GDB/insight调试应用程序

- list 列出相应于正在执行的程序的原文件内容
  next 执行下一个源程序行，从而执行其整体中的一个函数
  print 显示变量或表达式的值
  pwd 显示当前工作目录
  pype 显示一个数据结构（如一个结构或C++类）的内容
  quit 退出gdb
  reverse-search 在源文件中反向搜索正规表达式
  run 执行该程序
  search 在源文件中搜索正规表达式
  set variable 给变量赋值
  signal 将一个信号发送到正在运行的进程
  step 执行下一个源程序行，必要时进入下一个函数
  undisplay display命令的反命令，不要显示表达式
  until 结束当前循环
  up 上移栈帧，使另一函数成为当前函数
  watch 在程序中设置一个监测点

Procfs

- Procfs，内核提供的机制把内核信息映射成文件，一般我们我们把procfs mount到proc目录，这样我们可以通过普通的命令来访问；
  - /proc/kmsg #内核打印信息
  - /proc/iomem #内核io内存分配，可以看到decode，dsp，等得资源分配；
  - /proc/interrupts #中断信息，可以分析注册的中断号，和查看响应次数；
  - /proc/meminfo #内存信息，分析内存的使用情况；
  - /proc/cpuinfo #cpu信息，包含cache
  - /proc/1 #进程1的信息，里面包含有内存映射，线程信息，执行环境，堆栈信息等；
  - /proc/sys/ #内核信息和控制接口目录，我们可以通过这个里面的节点来控制内核；

Sysfs

- Linux2.6内核里面把proc文件里面关于驱动设备等信息独立出来，一般把sysfs mount到/sys目录；
  - /sys/block　　#块设备信息目录
  - /sys/bus　　#总线信息目录
  - /sys/class　　#逻辑设备信息目录
  - /sys/devices　#设备信息，以设备node区分；
  - /sys/firmware　#firmware信息
  - /sys/module　　#模块信息目录
  - /sys/power　　#电源信息目录