

Linux Support for Real-Time Applications

David L. Levine
Washington University, St. Louis
levine@cs.wustl.edu

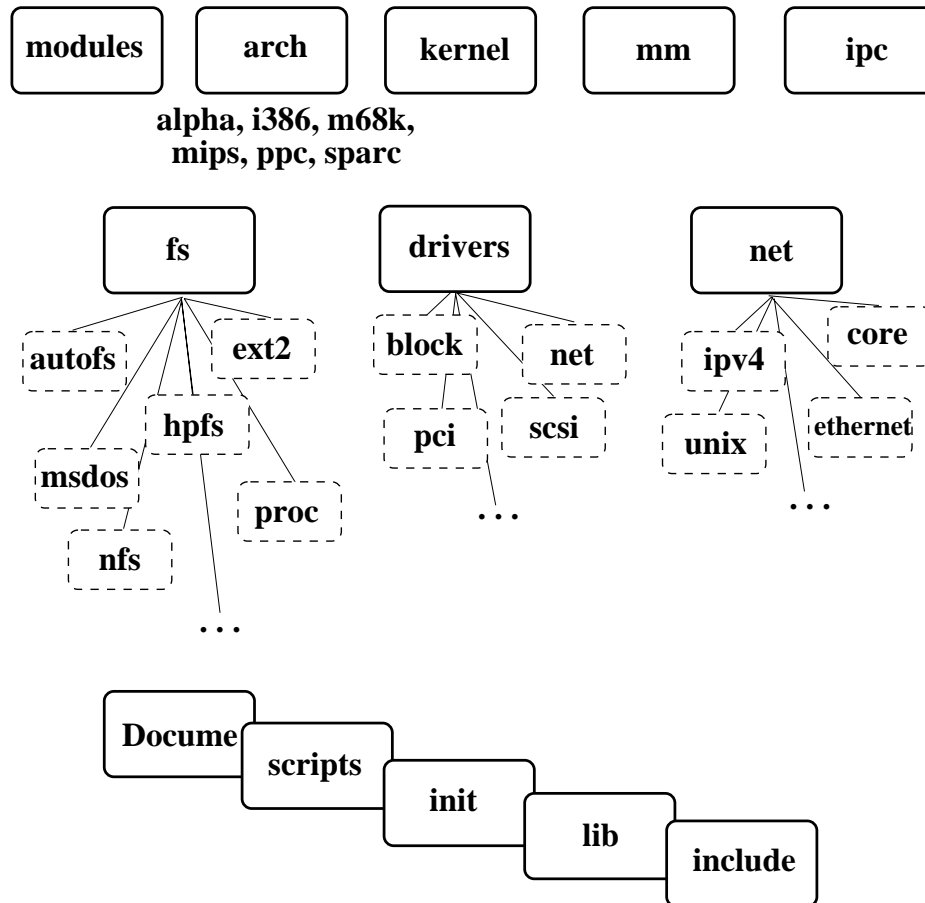
2 October 1998

<http://www.cs.wustl.edu/~levine/courses/cs6872/linux.pdf>

Linux 2.0 Real-Time Support

- Linux 2.0 kernel structure and mechanisms
- Linux scheduling
- Linux kernel threads
- RT-Linux
- KURT

Linux 2.0 Kernel Source Structure



- **Features**

- Separation of concerns/encapsulation
- Configuration support
- Modular (except for Alphas)

Linux Kernel Runtime Structure

Visible through `/proc` filesystem

1/	257/	318/	400/	cmdline	mounts
1476/	269/	337/	402/	cpuinfo	net/
1478/	278/	338/	414/	devices	pci
1479/	292/	341/	5/	dma	scsi/
193/	3/	342/	558/	filesystems	self@
2/	303/	356/	559/	interrupts	stat
202/	310/	357/	576/	ioports	sys/
213/	311/	375/	577/	kcore	uptime
224/	312/	376/	579/	kmsg	version
2262/	313/	394/	580/	loadavg	
235/	314/	396/	6/	locks	
246/	315/	398/	67/	meminfo	
2487/	317/	4/	7/	misc	

Linux System Info

```
/proc$ cat version
Linux version 2.0.34 (root@linux00.amt.tay1.dec.com) (gcc version 2.7.2.3) #2 Thu May 7 12:35:56 EDT

/proc$ cat cpuinfo
cpu                : Alpha
cpu model         : EV5
cpu variation     : 0
cpu revision      : 0
cpu serial number : Linux_is_Great!
system type       : EB164
system variation  : 0
system revision   : 0
system serial number : MILO-0000
cycle frequency [Hz] : 0
timer frequency [Hz] : 1024.00
page size [bytes]  : 8192
phys. address bits : 40
max. addr. space # : 127
BogoMIPS          : 497.02
kernel unaligned acc : 0 (pc=0,va=0)
user unaligned acc  : 244999 (pc=12005056c,va=12097b68b)
platform string    : N/A
```

Linux System Status

```
/proc$ cat loadavg  
0.00 0.00 0.00 2/56 2425
```

```
/proc$ cat meminfo  
          total:       used:        free:    shared: buffers:   cached:  
Mem:    262324224 111017984 151306240 63127552 18161664 55885824  
Swap:   534634496           0 534634496  
MemTotal:    256176 kB  
MemFree:     147760 kB  
MemShared:   61648 kB  
Buffers:     17736 kB  
Cached:      54576 kB  
SwapTotal:   522104 kB  
SwapFree:    522104 kB
```

Linux Kernel Support Mechanisms

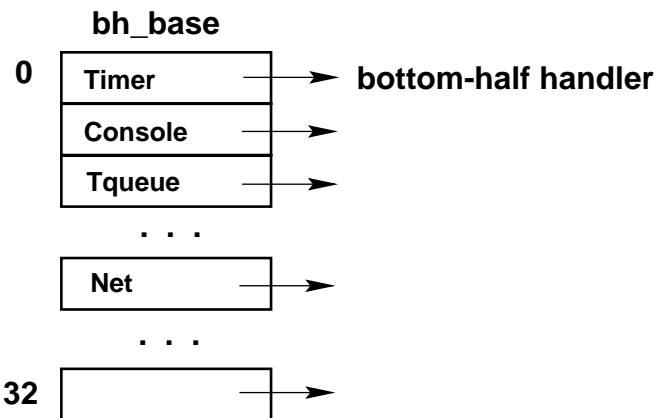
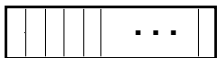
- Bottom-half (interrupt) handling
- Task queue
- Timer
- Wait queue
- Buzz (spin) lock
- Semaphore

Bottom-Half (Interrupt) Handling

bh_active:
queued for deferred execution



bh_mask:
shows installed handlers



- Minimize time spent in interrupt handlers
- Defers execution, via queues
- `bh_active` mask is checked at end of each system call
- Handlers called in order, 0 through 32

Task Queue

- Implements bottom-half handler queues
- Singly linked list of `tq_structs`
- `tq_struct`:

```
struct tq_struct {
    struct tq_struct *next;           /* linked list of active bh's */
    int sync;                         /* must be initialized to zero */
    void (*routine)(void *);         /* function to call */
    void *data;                       /* argument to function */
};
```

- Kernel uses for timer, immediate, and scheduler queues

Timer

- “Old” timer has static array of 32 pointers to `timer_structs`, plus `timer_active` mask. It is structurally similar to the bottom-half handler.
- “New” timer has linked list of `timer_lists`, maintained in ascending expiration order.

```
struct timer_list {
    struct timer_list *next;
    struct timer_list *prev;
    unsigned long expires;
    unsigned long data;
    void (*function)(unsigned long);
};
```

- Timer queues are processed by `TIMER` bottom-half handler.

Wait Queue

- Used by scheduler.
- Linked list of `task_structs`, which contains among other fields:

```
struct task_struct {
    volatile long state;      /* -1 unrunnable, 0 runnable, >0 stopped */
    long priority;
    unsigned long signal;
    unsigned long blocked;   /* bitmap of masked signals */
    unsigned long flags;     /* per process flags, defined below */
    int errno;
    /*
     * pointers to (original) parent process, youngest child, younger
     * sibling, older sibling, respectively. (p->father can be
     * replaced with p->p_pptr->pid)
     */
    struct task_struct *p_opptr, *p_pptr, *p_cptra, *p_ysptr, *p_osptr;
    struct wait_queue *wait_chldexit;      /* for wait4() */
};
```

Wait Queue, continued

```
/* file system info */
    int link_count;
    struct tty_struct *tty; /* NULL if no tty */
/* ipc stuff */
    struct sem_undo *semundo;
    struct sem_queue *semsleeping;
/* tss for this task */
    struct thread_struct tss;
/* filesystem information */
    struct fs_struct *fs;
/* memory management info */
    struct mm_struct *mm;
/* signal handlers */
    struct signal_struct *sig;
#ifdef __SMP__
    int processor;
    int last_processor;
    int lock_depth; /* Lock depth. We can context switch in and
                    out of holding a syscall kernel lock... */
#endif
```

Buzz (Spin) Lock

- Simple mutex implementation using single integer
 - Read the integer
 - If 1, busy wait
 - If 0, set to 1 and enter critical region
 - Reset to 0 after departing critical region
- Requires atomic test-and-set operation
- On Alphas, only used in SMP kernels; no-op on others.

Semaphore

- ```
struct semaphore {
 atomic_t count; /* Number of processes that can be granted
 access. If <= 0, accessors must wait. */
 atomic_t waking; /* Number of processes in wait_queue */
 int lock; /* [buzz lock] to make waking testing atomic */
 struct wait_queue *wait;
};
```
- `up ()`, `down ()`, and `down_interruptible ()` operations are provided.
- Other synchronization primitives are built on semaphores.

---

## Linux Scheduling

- POSIX interface (IEEE Std 1003.1b-1993, ISO/IEC 9945-1:1996)
- `SCHED_FIFO`, `SCHED_RR`, and `SCHED_OTHER` process scheduling classes
- `SCHED_OTHER` is (default) time-sharing, only allows priority 0. `SCHED_FIFO` and `SCHED_OTHER` allow priorities 1 through 99
- Linux kernel schedules processes, only, not threads.
- Threads implemented in library using `fork ()` (`clone ()`), with child memory shared copy-on-write with parent, and FDs shared)

## Linux Scheduling, continued

fork and thread creation performance:

| Platform                   | fork ( ) time,<br>$\mu\text{sec}$ | thread spawn time,<br>$\mu\text{sec}$ |
|----------------------------|-----------------------------------|---------------------------------------|
| Linux Alpha, 500 MHz       | 1000                              | 320                                   |
| Linux Pentium Pro, 200 MHz | 675                               | 310                                   |
| Solaris Ultra 30, 300 MHz  | 22000                             | 220                                   |

---

## Linux Scheduling Algorithm

- Check if called from an interrupt handler, and bail if so.
- Process the scheduler task queue.
- Disable interrupts.
- If the previous process was `SCHED_RR`, move to back of the runqueue.
- Set state of previous process.
- Enable interrupts.
- Pick the next process to run, based on scheduling class, priority, and age (`SCHED_OTHER`).

---

## Linux Scheduling Algorithm, continued

- If SMP, allocate the process to this CPU.
- Set the timeout timer.
- Run the processes using (assembler) CPU interface `switch_to`.

---

## Linux SMP Scheduling

- One scheduler per CPU
- One idle process per CPU
- `processor_mask` allows restriction of process to specified CPU(s).

---

## Linux Kernel Threads

- Eventual support for true kernel threads?
- Linux kernel threads developers page:  
`http://www.dstc.edu.au/linux-threads/`
- Last updated May 1996
- Asynchronous I/O isn't currently available, but is on the Linux 2.1/2.2 wish list:  
`http://www.cs.uml.edu/~acahalan/linux/21wishlist.html`

---

## RT-Linux

- Small RT executive runs Linux as lowest priority task
- RT kernel manages interrupts
  - Traps interrupt disable requests
  - Queues interrupts while they're supposed to be disabled
  - Delivers to Linux kernel when they're "re-enabled"
- Applications have a *real-time task* and a *user process*
  - Real-time task operates in kernel address space
  - Real-time task cannot make system calls
  - Communicate via a RT-FIFO
- Worst case interrupt latency on a 486/33Mhz PC is under 30  $\mu$ sec

---

## RT-Linux, continued

- RT kernel supports RMS and EDF, and shared memory; currently does not support protected memory or complex IPC mechanisms
- Task interface:

```
int rt_task_init(RT_TASK * task, void (*fn)(int data), int data,
 int stack_size, int priority);
int rt_task_make_periodic(RT_TASK *task, RTIME start_time,
 RTIME period);
```

and functions to suspend, wake up, wait, and delete a task.

- RT-FIFO interface includes functions for creating, destroying, writing to, reading from, and resizing RT-FIFOs.

---

## Kansas University Real-Time Linux (KURT)

- “Firm” real-time Linux
- Allows explicit scheduling of real-time events
- Supports periodic, very low jitter processing
- Two exclusive modes: *normal* and *real-time*
  - Real-time mode: executes real-time process *schedule* (only?)
  - When schedule is finished, switches to normal mode
- UTIME facility provides micro-second timing resolution

---

## KURT UTIME

- Programs timer chip to generate necessary interrupts. Both periodic and oneshot are supported.
- Replaces 10 msec tick by compensating for all OS code that assumes it.
- Interface:
  - Additional *usec* field in Linux `timer_list` struct.
  - `/usr/src/linux/include/linux/timer.h` has system calls to `init`, `add`, and `delete` timers. Upon expiration, the registered function is called.

---

## KURT Interface

- Mode switch commands:

```
int switch_to_rt (int timer_mode, unsigned long period,
 int rt_mode, char *cmdline, int length);
int switch_to_normal (int force);
```

- Define a real-time application interface, for requesting real-time module services:

```
int rtmod_cmd(int rtmod_id, int command, void *buf,
 unsigned long length);
```

- Set up the schedule:

```
int rt_schedule_events(struct timeval *start_time, int sched_mode,
 int num_times, char *filename);
```

---

## KURT Performance

- UTIME Provides 10 to 20  $\mu\text{sec}$  resolution. It does not increase clock drift.
- With 10 processes, measured  $\sim 1400 \mu\text{sec}$  time variation.
- With KURT, closer to 140  $\mu\text{sec}$ .

---

## Future Work

- Is Linux adequate for many “RT” applications?
- Compare native Linux latency/jitter with VxWorks 5.3.1, LynxOS 3.0.0, NetBSD, Solaris 2.6, and NT 4.0
- Compare KURT scheduling with native scheduling

---

## Information Sources

- David Rusling, The Linux Kernel:  
<http://www.redhat.com/linux-info/ldp/LDP/tlk/tlk.html>
- Doug Niehaus, *et al.*, KURT:  
<http://hegel.ittc.ukans.edu/projects/kurt/>
- Victor Yodaiken, Real-Time Linux:  
<http://luz.cs.nmt.edu/~rtlinux/>
- Eric S. Raymond, “The Cathedral and the Bazaar”:  
<http://earthspace.net/~esr/writings/cathedral-paper-1.html>