General structure of the Linux kernel

Geoffrey Thomas geofft@mit.edu

MIT Student Information Processing Board

November 11, 2008

The Linux Kernel

- Linux is a modular monolithic kernel
- Original design based on BSD and Minix
- Move towards modularity
  - /sys, VFS, ops, LKMs, syscalls, `binfmt_misc`, ...
- Extensive compile-time configuration mechanism

History

- April 1991: Usenet post
- March 1994: Linux 1.0.0
- June 1996: Linux 2.0.0
- January 1999: Linux 2.2.0
- January 2001: Linux 2.4.0
- December 2003: Linux 2.6.0
- Current: Linux 2.6.27.5

Resources

- The source!
  - http://www.kernel.org/
  - http://git.kernel.org/
  - http://lxr.linux.no/
- Linux Kernel Mailing List (http://www.lkml.org/)
- Linux Weekly News (lwn.net), KernelTrap.org
- Books, e.g., *LDD3*
  http://lwn.net/Kernel/LDD3/

Directory Tree (partial list)

- `arch`: boot, CPU management, ...
  - `arch/x86`
- `drivers`: bulk of the kernel
- `fs`
- `include`: structure definitions, etc.
  - `include/linux`
- `init`: bringing up the kernel after boot
- `kernel`: syscalls, scheduler, kernel features
- `lib`
- `mm`: memory management
- `net`
- `Documentation`

Process structure

- Processes are cheap (as UNIX requires)
- Threads are just a special type of process
- One kernel stack per process, so preemptible
- Kernel threads are processes with no userspace half

Completely Fair Scheduler

- Written by Ingo Molnar with ideas from Con Kolivas
- Emulates an "ideal multitasking CPU"
- `wait_runtime`
- Balanced tree, such that leftmost is next to be queued
- Options for desktop and server configs
- `cgroup` group schedule

Operations structures

```
const struct file_operations ext2_file_operations = {
        .llseek         = generic_file_llseek,
        .read           = do_sync_read,
        .write          = do_sync_write,
        .aio_read       = generic_file_aio_read,
        .aio_write      = generic_file_aio_write,
        .unlocked_ioctl = ext2_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = ext2_compat_ioctl,
#endif
        .mmap           = generic_file_mmap,
        .open           = generic_file_open,
        .release        = ext2_release_file,
        .fsync          = ext2_sync_file,
        .splice_read    = generic_file_splice_read,
        .splice_write   = generic_file_splice_write,
};
```

Using operations structures

```
struct file {
  ...
  unsigned int f_uid, f_gid;
  mode_t f_mode;
  struct path f_path;
  const struct file_operations *f_op;
  ...
}
  ret = file->f_op->read(file, buf, count, pos);
```

- inode_operations
- super_operations
- address_space_operations
- security_operations
- paravirt_ops

printk

```
printk(KERN_ERR "My pid is %d\n", current->tgid);
```

Linked Lists

Circular, doubly-linked

```
#include <linux/list.h>
struct my_struct {
  int data;
  struct list_head list;
};
LIST_HEAD(my_list);


void my_function(int n)
{
  struct my_struct foo = {.data = n};
  struct list_head *ptr;
  list_add(&my_struct->foo, &my_list);
  list_for_each(ptr, &my_list)
    printk("%d\n",
      list_entry(ptr, struct my_struct, list)->data);
}
```

Memory Allocation

- `kmalloc` and `kfree`
- Slab allocator
- `get_free_page`
- `vmalloc` and `vfree`
- `ioremap`
- per-cpu variables