Linux Kernel Implementation

Linus Torvalds
Computer Science Dept.
University of Helsinki

DECUS
May 1994
Contents

• Design Issues
• Basic Design
• Development Cycle
• Process Management
• Filesystem
• Memory Management
• Work in Progress
Design Issues

- **KISS**
  - Keep It Simple, Stupid
- **Compatibility**
- **Maximize performance**
  - don’t penalize good hardware
- **Keep it expandable**
  - avoid code or features locking us to a specific design
Basic Design

• **Monolithic Kernel**
  – Straightforward to code
  – modularization through coding

• **Non-pre-emptible**
  – avoids races and locking on kernel internal data structures

• **Machine-specific**
  – 176 000 lines of code
  – 88 000 of which are device drivers..
Development Framework

- Open development tree
  - ANSI C with 1TBS (K&R) layout
  - major parts in separate subdirectories

- Fast feedback
  - quick bugfixes

- Modularized development
  - hierarchy of responsible developers
  - but “outside” intervention encouraged
Process Management

- **Fundamental kernel routines**
  - Scheduling
  - Interrupt and DMA handling
  - Exception handling

- **ABI**
  - system call interface
  - signal handling
  - segmentation and v86-mode
Scheduling Example

• Traditional
  
  ```c
  sleep_on(&event_queue);  or
down(&event_counter);
  ```

• Linux
  
  ```c
  add_wait_queue(&event_queue);
current->state = TASK_SLEEPING;
if (!event)
  schedule();
current->state = TASK_RUNNABLE;
remove_wait_queue(&event_queue);
  ```
Filesystems

• Virtual File System layer
  – independent filesystems
  – virtualizes the filesystem interfaces
  – gives the user a unified filesystem layout
  – handles FS buffering

• Low-Level File System Components
  – UNIX filesystems: ext2, xfs, NFS, minix, sysv
  – data sharing: msdos, OS/2, iso9660
  – virtual: /proc
VFS Function Switch

• **Super-block operations**
  - `read_inode()`, `notify_change()`, `write_inode()`, `put_inode()`, `put_super()`, `write_super()`, `statfs()`, `remount_fs()`

• **Inode operations**
  - `create()`, `lookup()`, `link()`, `unlink()`, `symlink()`, `mkdir()`, `rmdir()`, `mknod()`, `rename()`, `readlink()`, `follow_link()`, `bmap()`, `truncate()`, `permission()`

• **File operations**
  - `lseek()`, `read()`, `write()`, `readdir()`, `select()`, `ioctl()`, `mmap()`, `open()`, `release()`, `fsync()`
Memory Management

- **Fundamental resource**
  - Kernel internal data
  - User process memory
  - IO buffering
  - shared memory

- **Maximize memory use**
  - minimize free memory
  - maximize memory re-use
Work in progress

- **Kernel threads**
  - separate filesystem, memory management and process information
- **Extended memory management**
  - improved allocation of physical memory
- **Filesystem optimizations**
  - name caching
  - improved data cache
- **Loadable modules**
- **Porting to other architectures**