What's new in Linux 2.6?

Dr. Ulrich Weigand
Linux for zSeries Development, IBM Lab Böblingen
Ulrich.Weigand@de.ibm.com

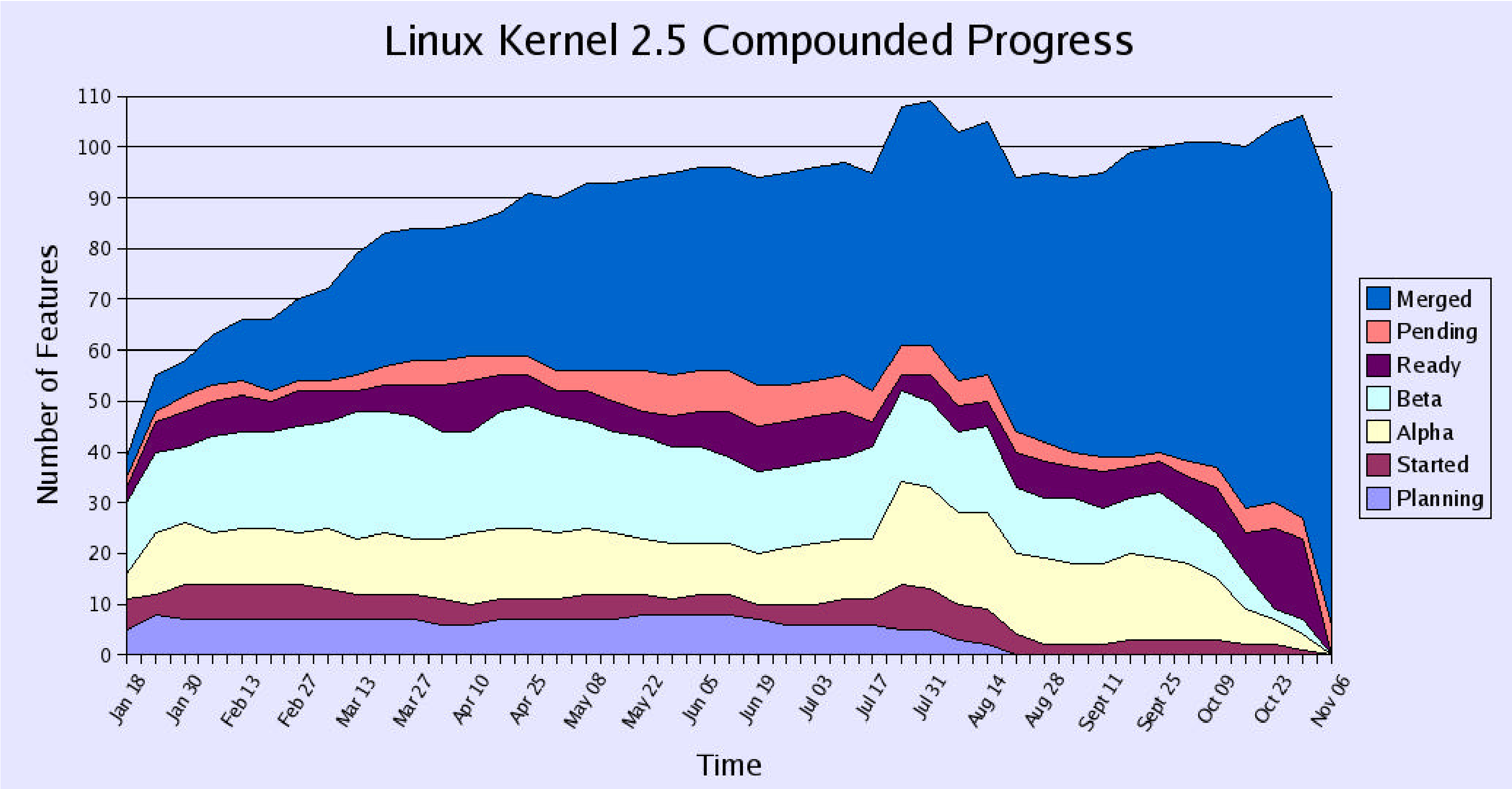
Agenda

- Timeline
- New features overview
- Scalability enhancements
- Threading model and futexes
- Device model and device configuration

Linux 2.6

- Timeline
  - January 1999: Linux 2.2.0 released
  - May 1999: Start of 2.3.x development (2.2.8)
  - January 2001: Linux 2.4.0 released
  - November 2001: Start of 2.5.x development (2.4.15)
  - October 2002: Feature freeze for 2.6
  - February 2003: Current version 2.5.61
  - Estimated release of Linux 2.6.0: mid-2003

Linux 2.6: Overvie



- Source: Guillaume Boissiere
  http://www.kernelnewbies.org/status/

Linux 2.6: Overview

- New features
  - Platform and device support
  - File systems and volume managers
  - Network protocols
  - Eliminate system limits
- Performance and scalability enhancements
  - Scheduler
  - Memory management
  - Block I/O layer
  - SMP scalability
- Backports to 2.4 kernels

Platform and device support

- New architectures
  - PowerPC 64-bit (ppc64)
  - AMD 64-bit (x86_64)
  - ucLinux (MMU-less processors: v850, m68knommu)
  - User Mode Linux
- New devices
  - New input device / frame buffer layers
  - ALSA (Advanced Linux Sound Architecture)
  - Video for Linux v2
  - New IDE layer, Serial ATA support

File systems

- Support for new file systems
  - IBM JFS
  - SGI XFS
  - NFS v4
  - Andrew File System (AFS)
  - ReiserFS v4 (planned)
- Other enhancements
  - Device mapper infrastructure (LVM2, EVMS)
  - Extended Attribute / Access Control List (ACL) support
  - Large directory support for ext2/ext3
  - Zero-copy NFS

Networking

- Networking enhancements
  - SCTP (Stream Control Transmission Protocol)
  - TCP segmentation offload
  - IPsec support and CryptoAPI
  - Improved IPv6 support
  - Bluetooth support

Scalability

- Removal of hard limits
  - Number of processes/threads: 64k -> 2G
  - Block device limit: 2TB -> 16TB / 8EB
  - Number of groups per process: 32 -> unlimited (planned)
  - Major/minor numbers: 256 -> 4k/1M (planned)
- SMP scalability
  - Reduce use of Big Kernel Lock
  - Eliminate global locks (I/O request, IRQ, task list)
  - Per-CPU data structures

O(1) scheduler

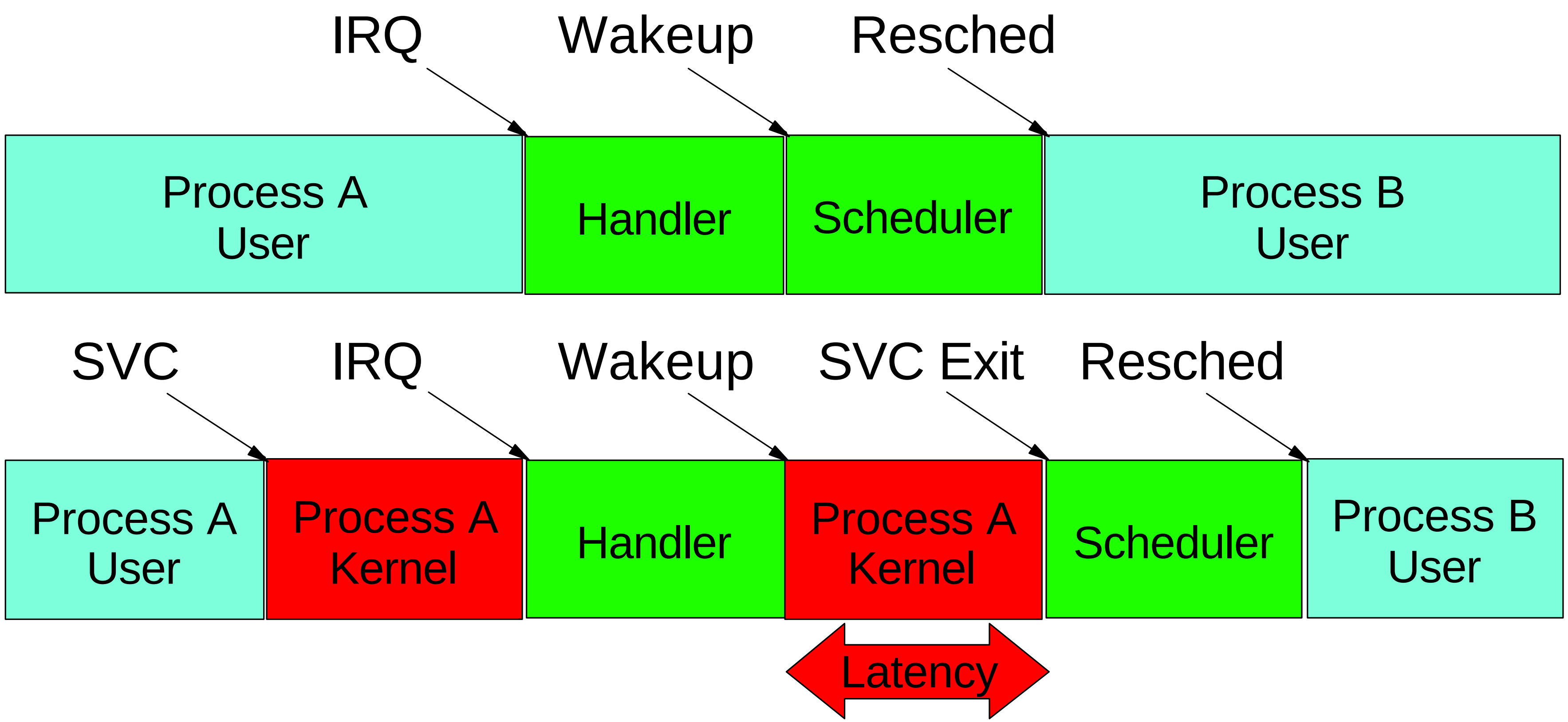
- Authors: Ingo Molnar et al.
- Design of old scheduler
  - Global run-queue holds all runnable processes
  - Reschedule scans full run-queue to find next process to run
  - Time-slice recalculation after all slices have been consumed
- Problems
  - Reschedule slow when run-queue is long
  - Recalculation loop slow, trashes cache
  - SMP scalability issues

O(1) scheduler (cont.)

- Design goals for new scheduler
  - O(1) algorithms: wakeup, schedule, timer interrupt
  - Scale to large number of processes/threads
  - Perfect SMP scalability
  - Processor affinity (incl. NUMA/SMT support)
  - Keep good interactive performance
  - Keep good performance with few runnable processes
  - Keep features: priorities, RT scheduling, CPU binding
- Implementation
  - Active/expired per-CPU priority arrays as run-queue
  - Load balancing between CPUs done by migration threads

Kernel preemption / low latency

- Authors: Robert Love, Andrew Morton, et al.
- Latency problem

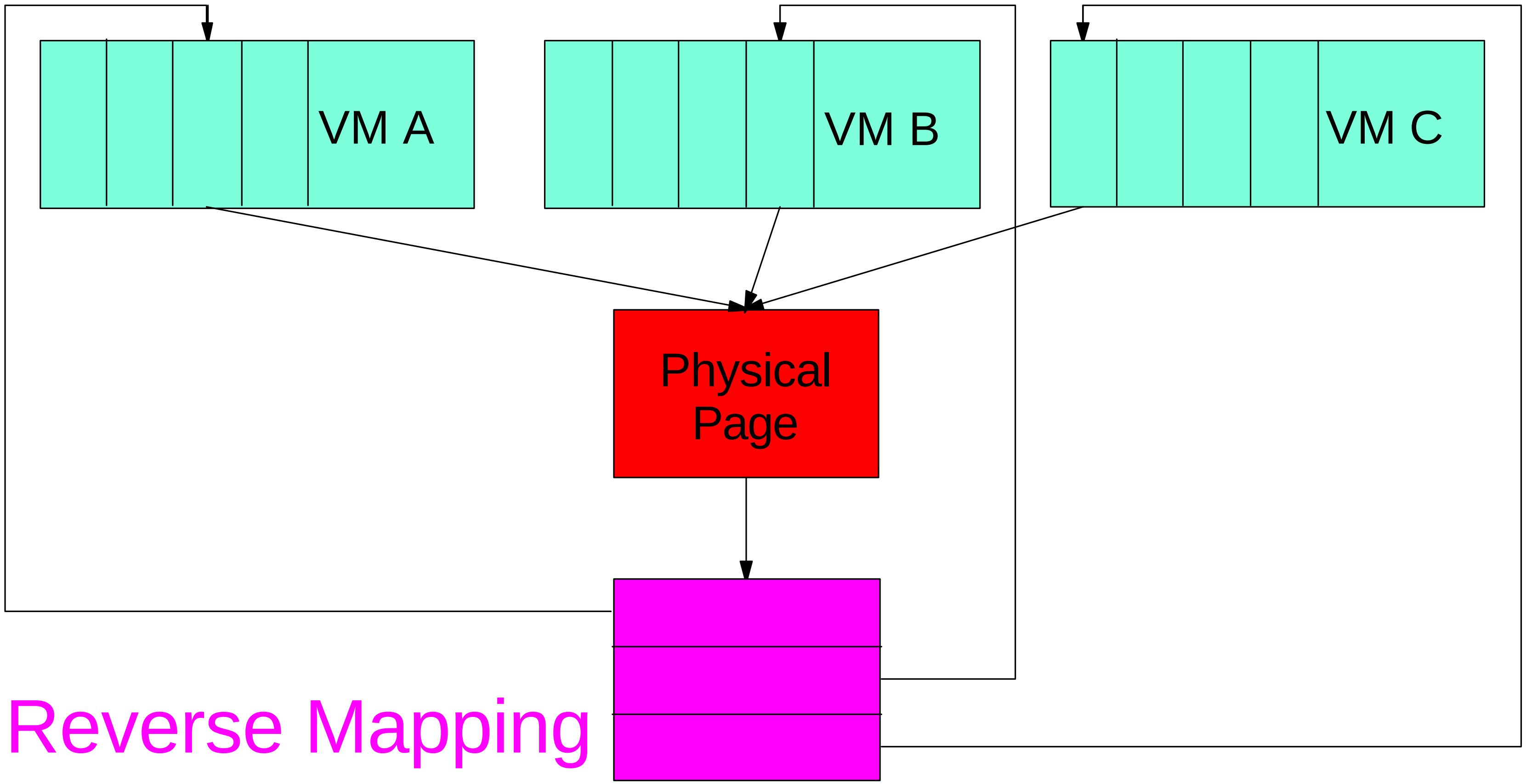


Kernel preemption / low latency (cont.)

- Proposed solutions
  - Kernel code yields voluntarily ('low latency patches')
  - Kernel code get preempted involuntarily
  - Currently implemented: both
- Preemption blockers
  - Interrupts (hard and soft)
  - Kernel SMP critical section (spinlock, per-CPU data etc.)
  - Scheduler (and other core routines)
- Design issues
  - Avoid large-scale code changes
  - Avoid throughput vs. latency trade-off

Memory management

- Authors: Rik van Riel, Andrew Morton, et al.
- Reverse mapping problem



Memory management (cont.)

- Advantages of reverse mappings
  - Easy to unmap page from all address spaces
  - Page replacement scans based on physical pages
  - Less CPU spent inside memory manager
  - Less fragile behaviour under extreme load
- Challenges with reverse mappings
  - Overhead to set up rmap structures
  - Out of memory while allocating rmap?

I/O scalability

- Authors: Jens Axboe, Andrew Morton, et al.
- Block I/O layer
  - Manages all access to block devices
  - Queues/merges/remaps block read/write requests
  - Implements 'buffer cache'
- Problems in 2.4
  - Shortcomings of 'buffer head' data structure
  - Large I/O, vectored I/O, raw I/O, async I/O inefficient
  - Global I/O request lock contention
  - Bounce buffer bottleneck on high-memory systems

I/O scalability (cont.)

- New BIO data structure
  - Efficiently unifies all types of I/O requests
  - Challenges
    - Rewrite much of the block I/O layer
    - Adapt all low-level drivers and remappers (MD, LVM)
    - Avoid deadlocks in out-of-memory situations
- Other enhancements
  - Eliminate global I/O request lock
  - Improved I/O scheduler
  - Merged buffer cache with page cache

Asynchronous I/O

- Authors: Ben LaHaise et al.
- Asynchronous I/O
  - I/O requests executed while application continues to run
  - Completion of I/O signalled to application
  - Goal: higher throughput, esp. for data bases etc.
- Implementation
  - Kernel provides async. I/O API (io_submit, io_getevents,...)
  - Synchronous kernel-internal interfaces switched to async.
  - Goal: everything in-kernel should be asynchronous
  - User space implements POSIX AIO on top

Networking scalability: epoll

- Authors: Davide Libenzi et al.
- Idle connection problem
  - Typical server load: many connections, few active
  - Event notification API (select, poll) performance degraded
- epoll: New notification mechanism
  - API: epoll_create / epoll_ctl / epoll_wait
  - Idle connections do not affect performance
  - Better performance, more robust than RT signals

Threading model

- Authors: Ulrich Drepper, Ingo Molnar
- Problems with LinuxThreads
  - POSIX non-compliance
    - One PID per process
    - POSIX signal handling
    - Inter-process synchronization primitives
  - Limited number of threads
  - Performance issues
  - Manager thread / heavy-weight library

Threading model (cont.)

- Design of new threading model
  - 1-on-1 model
  - No manager thread
  - Light-weight user space wrapper library
  - O(1) scheduler for large number of threads
  - Kernel/toolchain support for thread-local storage
  - Kernel awareness of 'thread groups'
  - Kernel support for fast thread start-up/exit
  - In-kernel POSIX signal handling
  - Synchronization primitives via 'futex'

Threading model (cont.)

- Thread-local storage support
  - New compiler feature (C/C++ language extension)
    - extern __thread int errno;
  - Compiler/Toolchain/Library support
    - Thread pointer via access register(s)
    - TLS relocations in assembler/linker
    - TLS support in dynamic linker and glibc
    - TLS access models to optimize performance
  - Kernel support
    - CLONE_TLS flag to clone()

Fast user space synchronization (Futex)

- Authors: Rusty Russell et al.
- Design goals
  - Intra-process and inter-process synchronization
  - Implement all POSIX synchronization primitives
  - Allow blocking and non-blocking wait
  - Allow multiple strategies (wake-one vs. wake-all etc.)
  - No administrative overhead (setup/cleanup etc.)
  - No system calls in the uncontended case
  - No unnecessary context switches
  - No limits (e.g. number of futexes)

Futex (cont.)

- Implementation
  - User space atomic operations on shared memory word
  - 'futex' system call to handle contention cases
- Futex system call
  - sys_futex (addr_t addr, int op, int val, struct timespec *timeout)
  - FUTEX_WAIT: If the lock word at 'addr' still contains 'val', sleep until a futex wakeup on 'addr' is performed or timeout.
  - FUTEX_WAKE: Wake up to 'val' processes sleeping on the futex 'addr'. Return number of processes actually woken.
  - FUTEX_FD: Return file descriptor usable for asynchronous wait on futex 'addr'. Optionally set up SIGIO signal 'val'.

Device model

- Authors: Patrick Mochel et al.
- Design goals
  - Represent physical device tree
    - Enables power-save suspend/resume operations
    - Simplifies device reference counting and locking
  - Enable dynamic device attach/detach
    - Automatically probe for devices, manual online/offline overrides
    - Interface with /sbin/hotplug user mode helper
  - Unified user interface
    - New file system: sysfs
    - Multiple subsystems provide 'views' into device tree

Device model (cont.)

- Devices Subsystem
  - Physical device interconnection tree
- Bus Subsystem
  - Top-level view of all device drivers by bus type
  - Links to connected devices
- Block Subsystem
  - Top-level view of all block devices and partitions
  - Links to underlying devices
- Net Subsystem
  - Top-level view of all network devices
  - Links to underlying devices

Device model (cont.)

- Bus and device types on zSeries
  - Channel Subsystem Bus / I/O Subchannel Devices
    - Identifier: Subchannel Number
    - Attributes: Channel Paths, PIM/PAM/POM
  - CCW Device Bus / CCW Devices
    - Identifier: Device Number
    - Attributes: Control Unit Type, Device Type, Online Status
  - CCW Device Group Buses
    - Group device: Multiple CCW devices used as a unit
    - Required for QETH, LCS, and CTC devices
    - Obsoletes 2.4 Channel Device Configuration layer
    - Identifier: First device in group
    - Attributes: Shared Online Status

Device model (cont.)

- User interface via sysfs: devices

```
/sys
   /devices
      /sys                        System Bus
         /channel_pathNN            Channel Path
      /css0                       Channel Subsystem Bus
         /0:NNNN                    Subchannel
            /0:NNNN                   CCW Device
      /qeth                       QETH Group Bus
         /0:NNNN                    CCW Device Group
   /bus
   /block
   /net
```

Device model (cont.)

- User interface via sysfs: device drivers

```
/sys
   /bus
      /css/drivers
         /io_subchannel     Subchannel Driver
            /0:NNNN           Links to /devices
      /css/devices          *All* devices
         /0:NNNN
      /ccw/drivers
         /dasd-eckd         DASD Driver
            /0:NNNN           Links to /devices
      /ccwgroup/drivers
         /qeth              QETH Driver
            /group            Group creation
            /0:NNNN           Links to /devices
```

Device model (cont.)

- User interface via sysfs: block devices

```
/sys
   /block
      /dasda                    DASD block device
        /device                   Link to /devices
        /dev                      Major/minor number
        ...
        /dasda1                   1st partition
          /dev                      Major/minor
        /dasda2                   2nd partition
          /dev                      Major/minor
```

Device model (cont.)

- Example: Install new QETH device

```
# Create QETH CCW group device
echo 0:5c00,0:5c01,0:5c02 \
   > /sys/bus/ccwgroup/qeth/group

# Set up portname parameter
echo portname:OSAPORT \
   > /sys/bus/ccwgroup/qeth/0:5c00/parameters

# Set device online
echo 1 \
   > /sys/bus/ccwgroup/qeth/0:5c00/online
```

Resources

- Linux for zSeries developerWorks page
  http://www.software.ibm.com/
  developerworks/opensource/linux390/index.html
- Linux for zSeries technical contact address
  linux390@de.ibm.com
- Linux for zSeries mailing list at Marist College
  http://www.marist.edu/htbin/wlvindex?LINUX-VM