Faculty of Computer Science Institute for System Architecture, Operating Systems Group

Construction of a Highly Dependable Operating System (J. Herder et al.)

presented by Bjoern Doebel

Dresden, 2008-02-27

Motivation

- Operating Systems crash
  - ~10 bugs per 1.000 LoC
  - device driver bugs are hard to debug, but: device drivers responsible for 85% of WinXP crashes
    - problem: device drivers are written by untrusted third-party developers (== device vendors)
- Idea: move device drivers to user space

User-level device drivers

- every driver isolated in a separate process
- no chance of corrupting other apps' data
- ideally, simply kill and restart the driver process

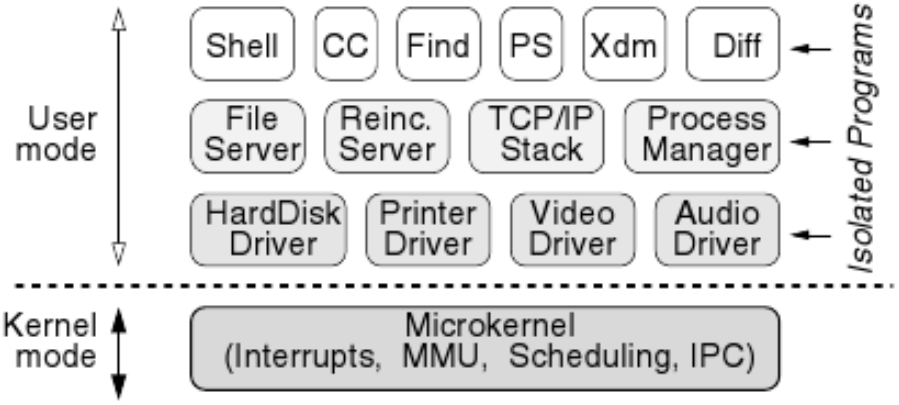

Figure 1: The design of our operating system is fully compartmentalized to properly isolate faults and enable recovery.

Approach

- Identify dependencies and come up with a solution
  - driver on kernel symbols
  - kernel depends on driver
  - drivers depend on each other
  - driver performs HW I/O
  - interrupt handler accesses driver data

Solutions

- Add new system calls
  - read/write I/O port vector
  - inter-AS copy
  - process privilege control
  - manage IRQs
- Redesign interrupt handlers
- IPC redesign
  - enforce combined send/receive operations (e.g., on kernel interaction)
  - asynchronous IPC

Solutions (2)

- IPC righs management
  - distinction between drivers – kernel – servers – users
- Restrict resource usage of drivers according to POLP – syscalls, IRQ lines, I/O ports
- Experiment with different file system block sizes
- Use a RAM disk to workaround block driver overhead... ?!

Recovery?

- Reincarnation server (-> *"Redesigning UNIX for reliability"*, ACSAC 2006)
  - parent of all drivers
  - upon termination receives signal and restarts driver
  - may periodically poll drivers for liveness and kill/restart them if they don't answer
- Helps for transient errors
  - aging errors
  - temporary attacks

Recovery (2)

- Data store
  - apps can store specific data
    - like priv_data[app_name]
    - can be used to implement name service, too
  - pub/sub system to get notifications
    - clients can subscribe to their drivers' data and get notified about changes
- Recovery/Retry after driver restart is left to the clients... !?

Comparison to L4(.Fiasco)

- no dedicated syscalls
  - Port I/O (-> IOPL)
  - inter-AS copy (-> IPC)
  - IRQ management (-> IPC, Omega0)
- more fine-grained rights management
  - I/O guard
    - I/O bitmap for ports
    - I/O dataspaces
  - IPCMon – IPC control with task granularity

Comparison to L4(.Fiasco)

- reuse legacy drivers instead of redesigning everything from scratch
  - DDEKit + DDE{Linux,FBSD}
  - easy support for a broad range of drivers
- Our Achilles' Heel: No restartable device drivers

Discussion

- Portability vs. platform-specific optimizations:
  "On most machines, doing I/O is not possible in user mode. On some machines there may be a way to map a page containing I/O registers to user space or map some of the I/O ports to user space, but this is not always possible and should not be relied on."
- I doubt that Reincarnation Server and Data Storage are enough for restartability.
- Why didn't we write a paper on DDE in 2001...?