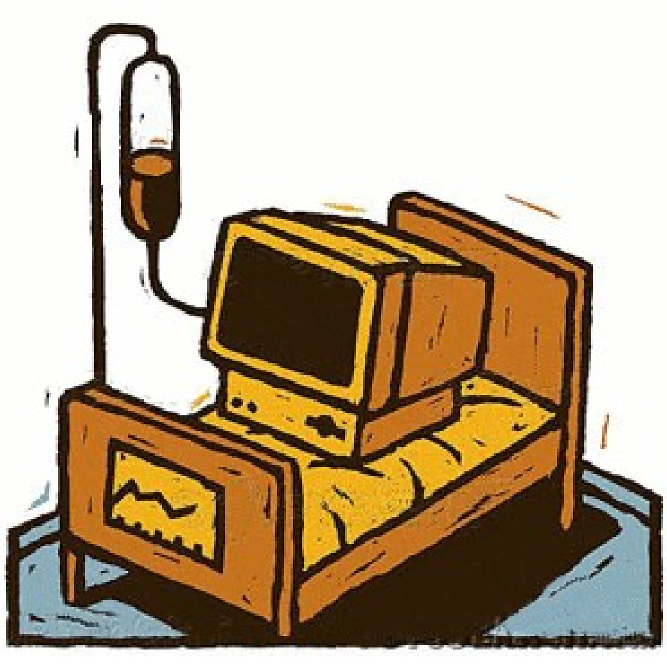
MINIX 3: A HIGHLY RELIABLE SELF-REPAIRING OPERATING SYSTEM

Research Summary

University of British Columbia
Vancouver, Canada
August 2, 2006



Jorrit N. Herder
Dept. of Computer Science
Vrije Universiteit Amsterdam

CENTRAL THEME

"Have no fear of perfection – you'll never reach it."

~ Salvador Dalí (1904-1989)

Jorrit N. Herder <jnherder@cs.vu.nl>

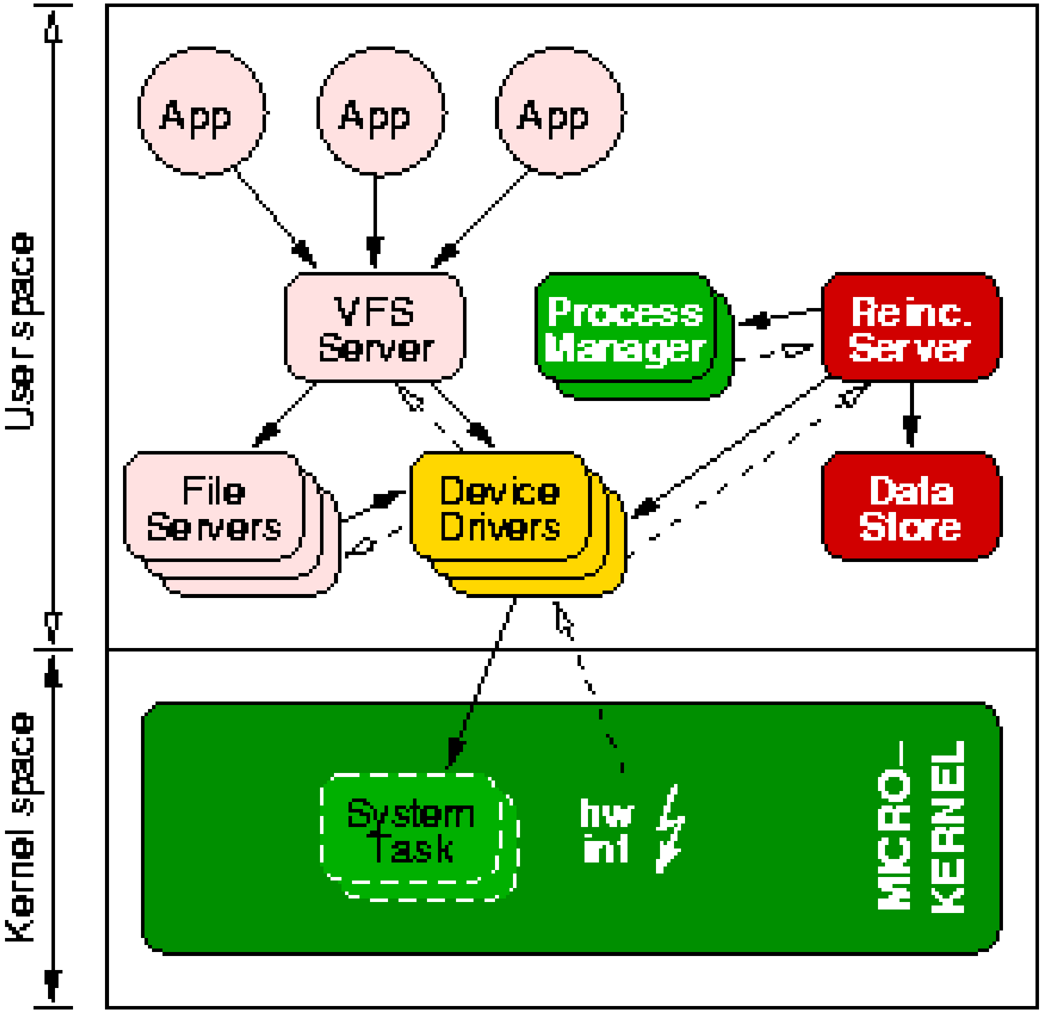
TALK SUMMARY

- **Problem Statement**
  - Bug-induced failures in critical OS components are inevitable
    - Getting all servers and drivers correct (or fault-resilient) is not practical
  - A single failure is potentially fatal in a commodity systems
    - Reboot is not always possible or wanted

- **Contribution**
  - Therefore, we have built a fault-resilient OS, MINIX 3
    - Fault resilience: ability to quickly recover from a failure
  - OS is compartmentalized to isolate faults and enable recovery
  - OS can automatically detect and repair certain defects

Jorrit N. Herder <jnherder@cs.vu.nl>

ARCHITECTURE OF A FAULT-RESILIENT OS



- **Reincarnation Server**
  - Manages drivers
  - Monitors system
  - Repairs defects
- **Data Store**
  - Publishes configuration
  - Allows to backup state

Jorrit N. Herder <jnherder@cs.vu.nl>

TALK OUTLINE

- Summary (done)
- Introduction (next)
- Fault Resilience
- Evaluation
- Discussion
- Conclusions

INTRODUCTION

Jorrit N. Herder <jnherder@cs.vu.nl>

PERCEIVED PROBLEMS

- **Weak security and reliability**
  - Computer crashes **<-- current focus**
  - Digital pests (viruses, worms, etc.)
- **Complexity**
  - Hard to maintain and configure
  - Too large for embedded and mobile computing

INHERENT PROPERTIES OF MONOLITHIC DESIGNS

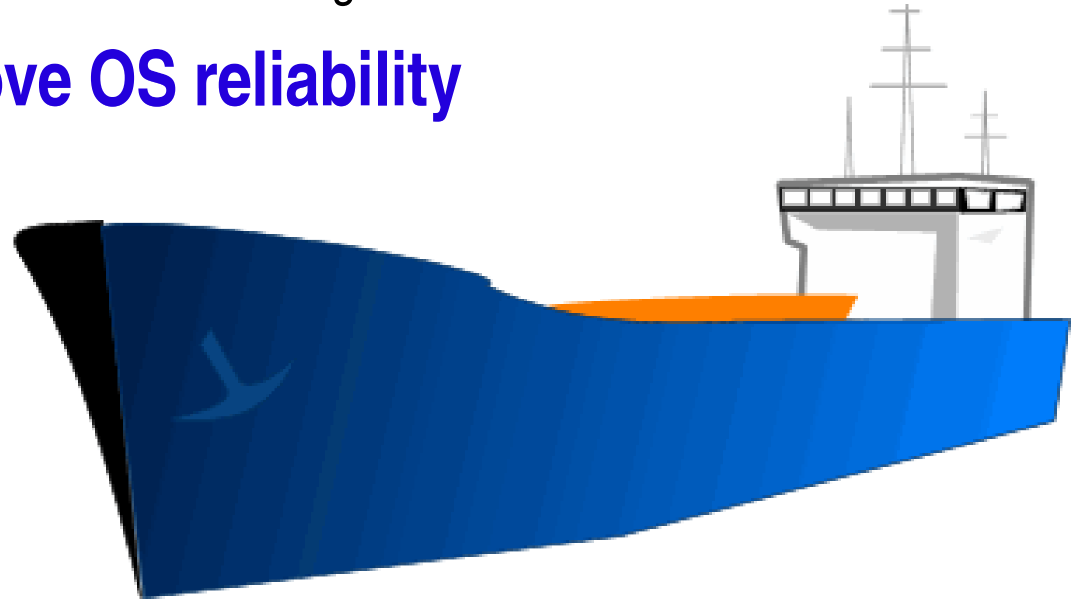
- **Fundamental design flaws in monolithic kernels**
  - All code runs at highest privilege level (breaches POLA)
  - No proper fault isolation (any bug can be fatal)
  - Huge amount of code *in* kernel (6-16 bugs per 1000 LoC)
  - Untrusted, 3rd party code in kernel (70% driver code)
  - Entangled code increases complexity (hard to maintain)

- Ok, the printer looks solid, but do you trust the driver?
- Why is the entire network stack in the kernel?
- Would you run my nifty kernel module?

Jorrit N. Herder <jnherder@cs.vu.nl>

HOW ABOUT MODULAR DESIGNS?

- **Modularity is commonly used in other engineering disciplines**
  - Ship's hull is compartmentalized to improve it's 'reliability'
    - If one compartment springs a leak, the others are not affected
  - Aircraft carrier is build out of many, well-isolated parts
    - Clogged toilet cannot affect missile launching facilities
- **Use modularity to improve OS reliability**



Jorrit N. Herder <jnherder@cs.vu.nl>

TOWARDS A FAULT-RESILIENT OS

- **We fully compartmentalized the operating system**
  - Transformation into a minimal kernel design (< 3800 LOC)
    - Kernel does minimal tasks to support user-mode operating system
  - All servers and drivers run in a separate user-mode process
    - Just like ordinary applications (with some minor exceptions)
- **We added mechanisms to detect and repair failures**
  - Privileged server can replace failed components
    - Crashed user processes can be restarted

Jorrit N. Herder <jnherder@cs.vu.nl>

THE MINIX 3 USER-MODE SERVERS AND DRIVERS

- **Device drivers, e.g.:**
  - Disk drivers
    - S-ATA, floppy, RAM disk
  - Terminal driver
    - Console, keyboard, serial
  - Fast Ethernet
    - Realtek, IntelPro, 3COM, NE2000.
  - Printer
  - Audio

- **Core servers**
  - File Server
  - Process Manager
  - Reincarnation Server
  - Data Store
- **Other services**
  - Network Server
  - Information Server
  - X Window System

REASONING BEHIND OUR APPROACH

- **Guarding drivers tackles most severe problem**
  - 70% of Linux source code consists of drivers
  - New hardware and drivers developed all the time
  - OS servers are more stable and tested over time
- **Key benefit over other approaches: simplicity**
  - Process model has been known for decades
  - No complex VM configuration management
  - No outdated wrappers with next kernel version

Jorrit N. Herder <jnherder@cs.vu.nl>

RELATED WORK IN FAULT RESILIENCE

- **Our work differs significantly from other approaches:**
  - Software-based isolation, interposition, and recovery of in-kernel drivers
    - Kernel mode limits isolation and aging of manually written wrappers
  - Run device drivers in dedicated user-mode virtual machines
    - More complex resource and configuration management
  - Minimal kernel designs running drivers in single-server OS
    - Still single point of failure and recovery is not possible
  - MMU-protected user-mode drivers without recovery mechanisms
    - Can benefit from our work by adding recovery mechanisms
  - Language-based protection and formal code verification
    - Complementary to our approach

FAULT RESILIENCE

Jorrit N. Herder <jnherder@cs.vu.nl>

FAULT ISOLATION

- **Limit consequences of faults to enable recovery**
- **All servers and drivers can fail independently**
  - Servers and drivers fully compartmentalized in user space
  - Private address spaces protected by kernel and MMU
    - Direct access of other process' memory is denied by MMU
    - Virtual copies between user processes require copy grant
    - Protection against DMA corruption requires I/O MMU
  - Privileges of each process reduced according to POLA
    - Unprivileged user and group ID
    - IPC primitives, possible IPC destinations, kernel calls
    - I/O ports and IRQ lines allowed

DEFECT DETECTION

- **Human user observes failure because of malfunctioning**
  - System crashes or becomes inresponsive
- **OS defect detection requires constant monitoring**
  - RS is parent of all servers and drivers and knows when one exits
    - RS immediately receives alert (SIGCHLD) from process manager upon exit
  - RS periodically checks drivers status using nonblocking IPC
    - Queried driver must respond within next period
    - Nonblocking notification messages prevent clogging the system

DEFECTS WE CAN DEAL WITH

- **Fault model**
  - Crashes, panics, or unexpected exits
  - Attack failures such as ping of death
  - Byzantine or logical failures are excluded
- **Assumptions**
  - Restart makes recovery possible
    - We cannot recover if hardware fails

RECOVERY PROCEDURE (1/3)

- **Fault-tolerant systems use redundancy to overcome failures**
- **Our fault-resilient design tries to automatically *repair* defects**

  (1) Malfunctioning component is identified

  (2) Associated policy script is run

  (3) Component can be replaced with a fresh copy

  - How to recover lost state?
  - How to deal with dependant components?

RECOVERY PROCEDURE (2/3)

- **Policy scripts**
  - Control recovery procedure
  - Full flexibility, e.g.:
    - Backup core dump and log error message
    - Send e-mail to remote administrator
    - Restart failed components
- **Restarting dead drivers**
  - Full restart through VFS
  - Lightweight execution by RS to bypass VFS
    - Disk drivers shadowed in RAM to allow recovery

Jorrit N. Herder <jnherder@cs.vu.nl>

RECOVERY PROCEDURE (3/3)

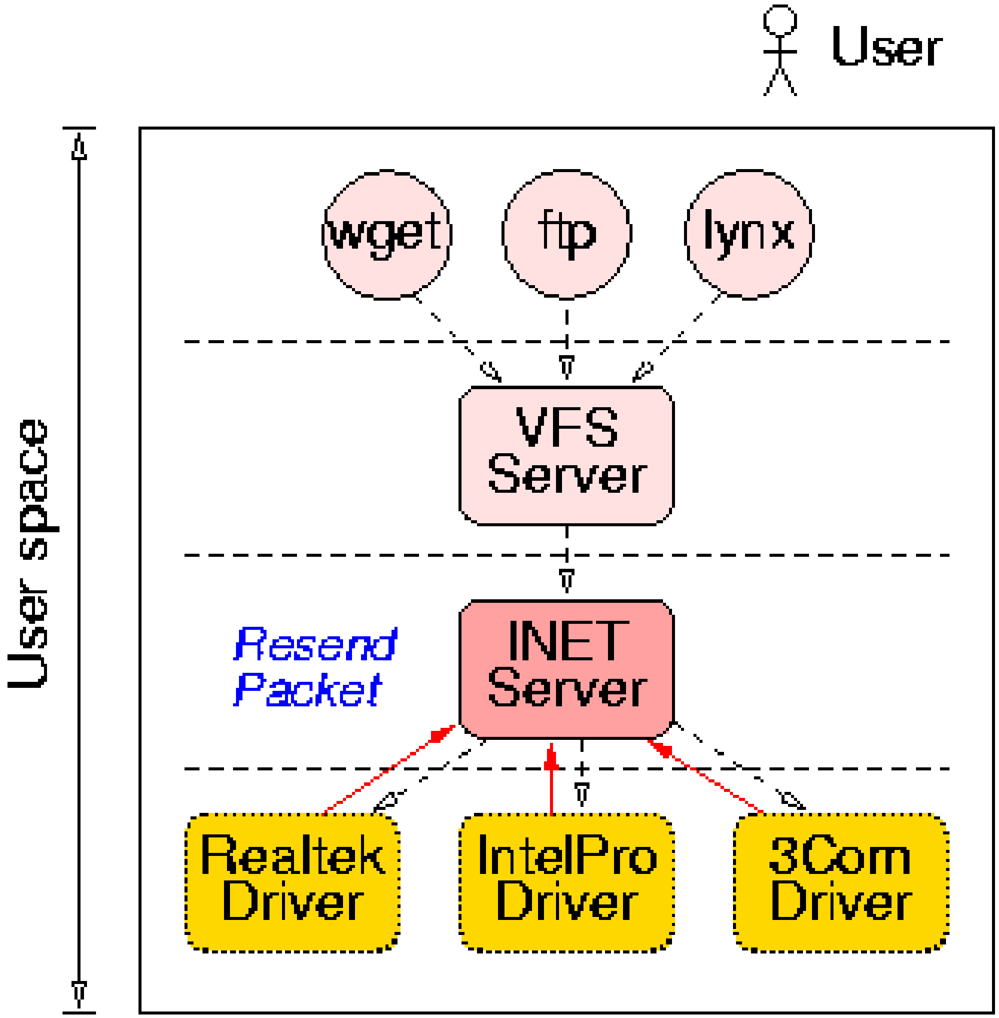
- **Recovering state**
  - Drivers mostly stateless; server-level does reinitialization
  - Some state can be privately stored at DS for local recovery
  - Restarting servers is problematic as (too) much state is lost
- **Dependant components**
  - RS publishes changes in system configuration at DS
  - IPC requests can fail, e.g., VFS request to driver
  - Errors are pushed up:
    - Recovery procedure starts at server level
    - Errors pushed to application level when recovery is not possible

Jorrit N. Herder <jnherder@cs.vu.nl>

EXAMPLES AND LIMITATIONS

- **Focus in on device drivers (worst problem)**
  - Ethernet driver recovery
  - Character driver recovery
  - Disk driver recovery
- **Recovery of failed servers**
  - Sometimes possible, depending on how much state is lost
    - Anything from user-supported recovery to transparent recovery
- **Limitations of our system**
  - Failures in the core servers are fatal

ETHERNET DRIVER CRASH



- **Transparent recovery**
  - Hidden in network server
    - Due to TCP/IP protocol

- **Recovery steps taken**

  (1) RS replaces dead driver

  (2) RS publishes update

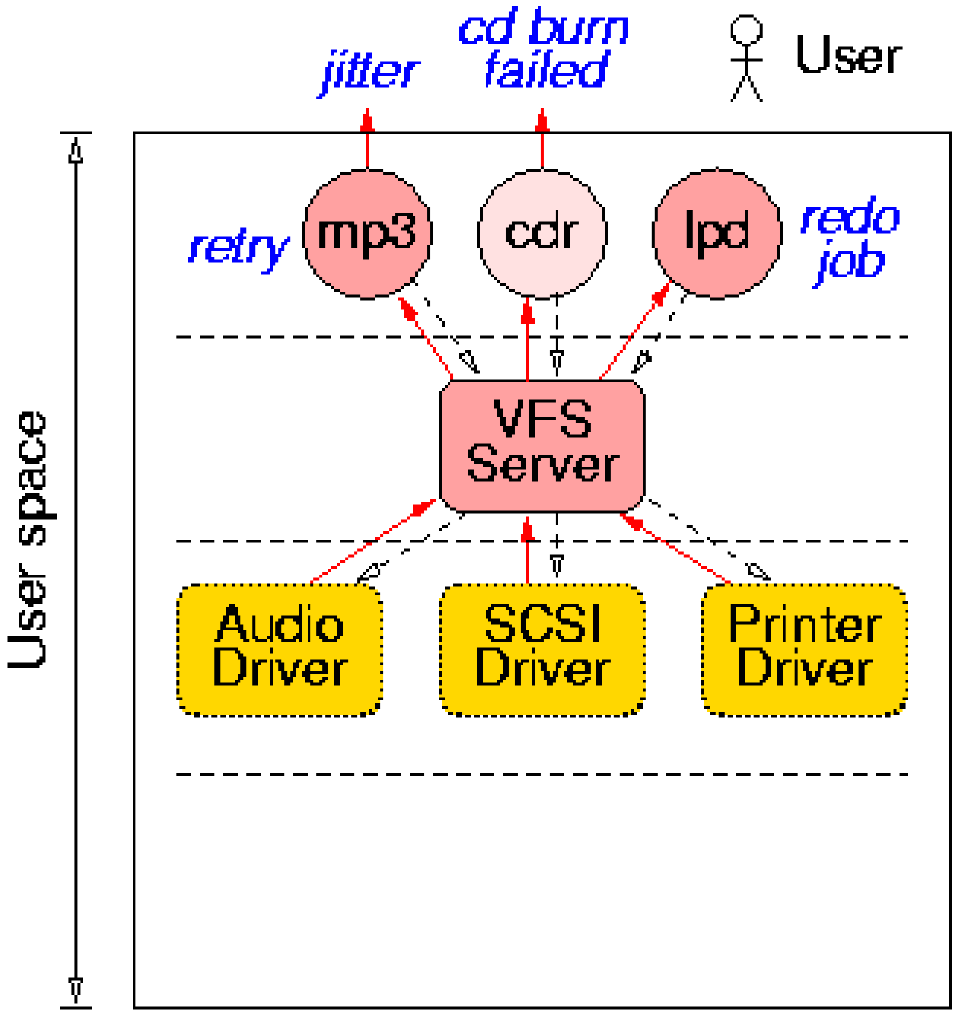
  (3) DS informs INET server

  (4) INET reinitializes driver

  (5) INET resends lost data

Jorrit N. Herder <jnherder@cs.vu.nl>

CHARACTER DEVICE DRIVER CRASH



- **No transparent recovery**
  - Recovery at application level
  - Error pushed back to user
    - Data stream interrupted
- **Recovery steps taken**

  (1) RS replaces dead driver

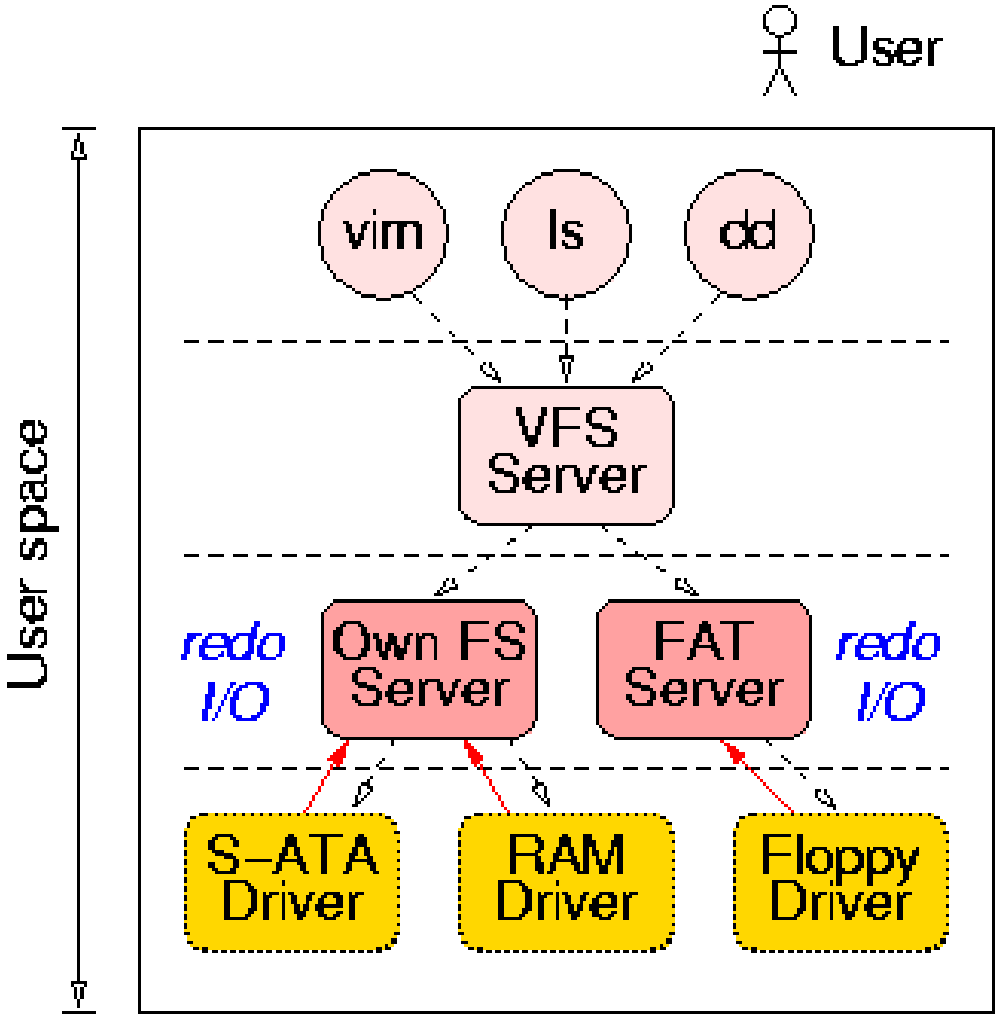
  (2) RS publishes update

  (3) DS informs VFS server

  (4) VFS returns I/O error to app

Jorrit N. Herder <jnherder@cs.vu.nl>

BLOCK DEVICE DRIVER CRASH



- **Transparent recovery**
  - Hidden in file server (FS)
    - Keep I/O requests pending
- **Recovery steps taken**

  (1) RS replaces dead driver

  (2) RS publishes update

  (3) DS informs FS server

  (4) FS retries pending request

INFORMATION SERVER CRASH

- **Handles formatted debug dumps of various data structures**
  - Data structures to be shown are in other servers
  - No state is lost when information server crashes
- **Recovery is transparent to the user and other servers**
  - Restarting information server simply does the job

NETWORK STACK (INET) CRASH

- **Suppose the INET server crashes, what would happen?**
  - All state, including all open TCP/IP sockets, is lost
  - All applications using the network server are affected
  - However, the system does not crash in its entirety!
- **Currently, manual recovery is possible**
  - Steps can be included in a policy script:
    - Restart INET server
    - Restart DHCP daemon
- **In future, data store may be used to backup state**

Jorrit N. Herder <jnherder@cs.vu.nl>

EVALUATION

Jorrit N. Herder <jnherder@cs.vu.nl>

DEPENDABILITY EVALUTION

- **Fault-injection experiments**
  - So far we have only manually injected faults
- **Measurements of the recovery overhead:**
  - Ethernet driver recovery:
    - Simulated repeated crashes with different time intervals
    - Transparent recovery was succeeded in all cases
    - Mean recovery time is 0.36 sec due to TCP retransmission timeout
      - 25% overhead with 1 crash every 1 sec
      - 8% overhead with 1 crash every 4 sec
      - 1% overhead with 1 crash every 25 sec
      - no overhead with no crashes

PERFORMANCE MEASUREMENTS

- **System feels fast and responsive**
  - Time from multiboot monitor to login is under 5 sec.
  - The system can do a full build of itself within 4 sec.
- **Overhead of user-mode drivers (without optimizations)**
  - Run times for typical applications: 6% overhead
  - File system and disk I/O performance: 9% overhead
    - Disk throughput (with fast disk and DMA) up to 70 MB/s
  - Networking performance: Fast Ethernet at full speed
    - Experiments show Gigabit Ethernet also works at full speed

Jorrit N. Herder <jnherder@cs.vu.nl>

SOURCE CODE STATISTICS

- **Kernel (including kernel tasks): < 4000 LoC**
- **Most important servers and drivers: ~2500 LoC**
- **Minimal POSIX-conformant system: ~20,000 LoC**
  - TCB reduced by 3 orders of magnitude compared to Windows
  - TCB depends on user's requirements and may be larger
    - Our TCP/IP networking server: ~20,000 LoC
    - The X Window System: ~80,000 LoC

Jorrit N. Herder <jnherder@cs.vu.nl>

DISCUSSION

Jorrit N. Herder <jnherder@cs.vu.nl>

USER VIEW OF MINIX 3

- **Using MINIX 3 is like using a normal multiuser UNIX system**
  - However, not as mature as FreeBSD or Linux
  - Only 18 months of development with small core of people
    - Nevertheless, over 400 UNIX applications available
    - Recently, the X Window System was ported
    - VFS infrastructure was also added
- **Currently Intel x86, but ports to other architectures underway**
  - Including, PowerPC, XScale
  - Future releases, may also target embedded devices

THE MOST IMPORTANT RELIABILITY FEATURES

1. Tiny kernel is easy to understand and get correct
2. OS bugs in user space are not necessarily fatal
3. Operating system can detect and repair driver failures
4. Infinite loops in servers and drivers detected
5. Memory protection through MMU hardware and kernel
6. Drivers cannot do I/O themselves, but need to ask kernel
7. IPC capabilities restricted according to POLA
8. IPC uses rendezvous with fixed-length messages
9. Interrupts and events use asynchronous notifications

LESSONS LEARNED

- **Recovering lost driver state is <u>not</u> the biggest problem**
  - In practice, only needed for some specific drivers
    - E.g., how to retrieve RAM disk regions after restart?
  - To restart servers, however, lost state becomes a key problem
    - Part of future research (e.g., file server recovery)
- **Integrated approach required for optimal results**
  - Servers and applications must be able to deal with driver errors
  - Recovery done at lowest possible layer, otherwise pushed up

Jorrit N. Herder <jnherder@cs.vu.nl>

GENERAL APPLICABILITY

- **Our techniques can be reused on other systems**
  - Trend towards user-mode drivers on other operating systems
    - User-mode drivers on Linux have been successfully tested
    - Next version of Windows (Vista) will also have user-mode drivers
  - User-mode drivers can be guarded similarly to what we have done
    - Reincarnation server and data store have to be ported
    - Minimal changes to device drivers; servers need to deal with failures
- **Performance overhead is not a real issue**
  - Trade-off between performance and dependability is changing
  - Penalty of ~10% negligible compared to hardware improvements

CONCLUSION

Jorrit N. Herder <jnherder@cs.vu.nl>

CONCLUSIONS (1/2)

- **We have built a highly reliable, self-repairing OS**
  - Full compartmentalization of the OS in user space
  - Explicit mechanisms to detect and repair failures
    - Deals with an important problem, namely device driver failures
    - Exceptions are caught and transparent recovery is often possible
- **Improvements over other operating systems**
  - Number of fatal (kernel) bugs is reduced
  - Compartmentalization limits bug damage
  - Recovery from common failures is possible

CONCLUSIONS (2/2)

- **Evaluation of MINIX 3**
  - Performance overhead of 5-10% compared to base system
  - Crash simulation experiments prove viability of approach
  - TCB (source code) reduced by up to 3 orders of magnitude
- **Practicality of our approach**
  - Our techniques can be applied to of other systems, such as Linux
  - Limited costs make real-world adoption attractive

ACKNOWLEDGEMENTS

- **Talk organization**
  - Andrew Warfield
- **The MINIX 3 team**
  - Mischa Geldermans
  - Ben Gras
  - Philip Homburg
  - Herbert Bos
  - Andy Tanenbaum

QUESTIONS & DISCUSSION

- **More information**
  - Web: www.minix3.org
  - News: comp.os.minix
  - E-mail: jnherder@cs.vu.nl
- **This talk's article**
  - ACM SIGOPS OSR July
- **Try it yourself!**
  - MINIX 3 Live CD-ROM