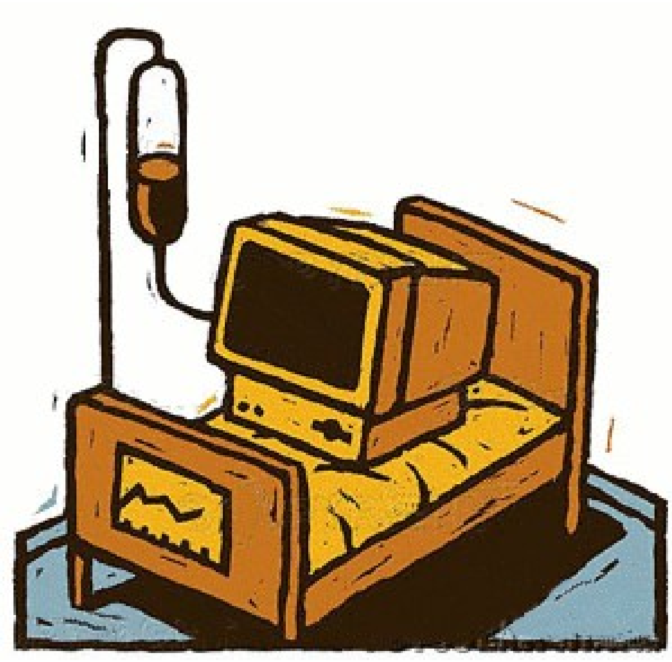
“REINCARNATION OF DEAD DEVICE DRIVERS”

Paper Proposal

1st EuroSys Authoring Workshop
April 2006 – Leuven, Belgium

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



BETTER TITLE:

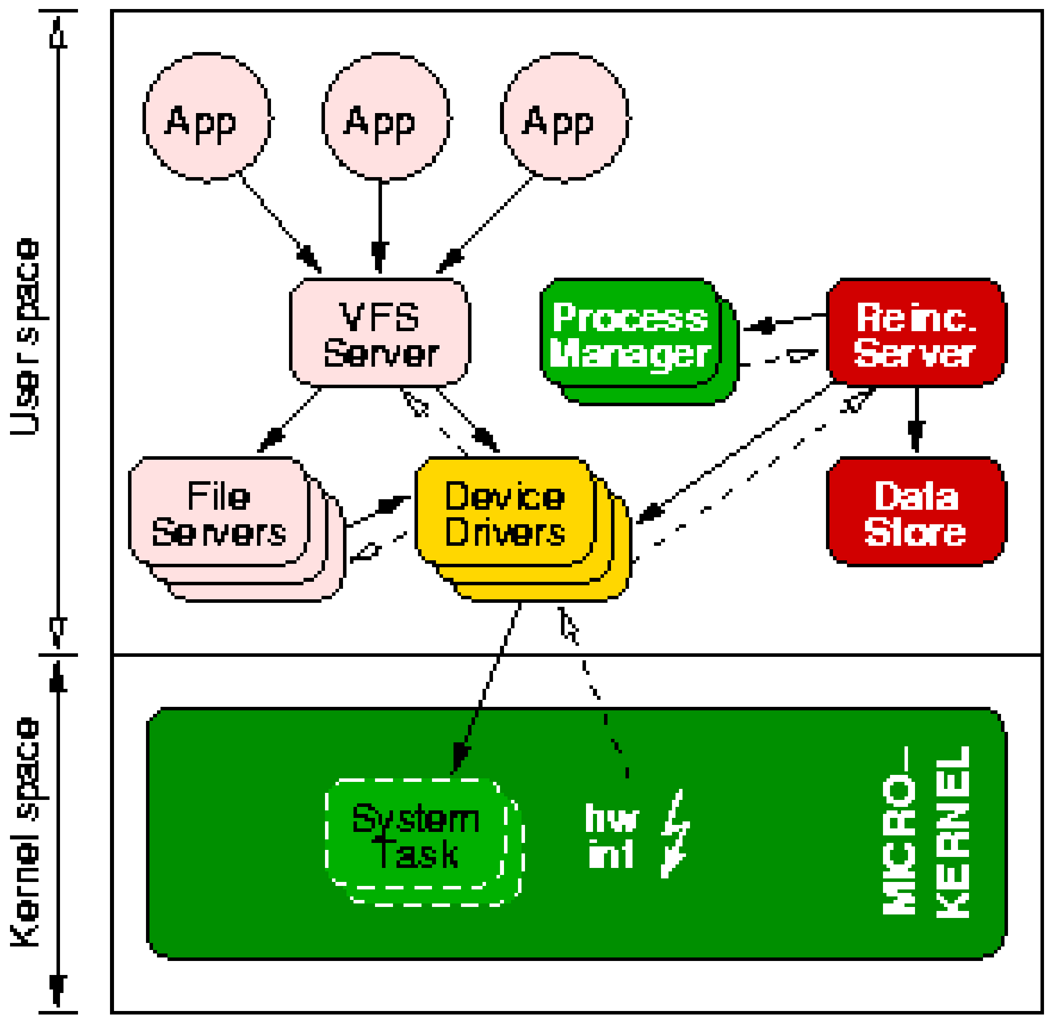
"TOWARDS A FAULT-RESILIENT OPERATING SYSTEM"

Fault resilience: ability to quickly recover from a failure

Sec. 1: INTRODUCTION

- **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
- **Sec. 1.1: Contribution**
  - Therefore, we have built a better OS that is <u>fault resilient</u>
- **Approach**
  - Compartmentalize the OS to enable recovery
  - Automatically detect and repair defects

ARCHITECTURE OF A FAULT-RESILIENT OS



- **Reincarnation Server**
  - Manage drivers
  - Monitor system
  - Repair defects
- **Data Store**
  - Publish configuration
  - Backup state

PAPER OUTLINE

- Sec. 1: Introduction (done)
- Sec. 2: Related work
- Sec. 3: Fault isolation
- Sec. 4: Defect detection
- Sec. 5: Recovery procedure
- Sec. 6: Examples and limitations
- Sec. 7: Dependability evaluation
- Sec. 8: Performance
- Sec. 9: Discussion
- Sec. 10: Conclusions
- Sec. 11: Acknowledgements
- Sec. 12: Availability

Sec. 2: 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 manually written wrappers required
  - 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
    - New and more effective recovery mechanisms are possible
  - Language-based protection and formal code verification
    - Complementary to our approach

Sec. 3: 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 MMU
    - Copies to/from applications require explicit permission
    - 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

Sec. 4: DEFECT DETECTION

- **System's well-being is constantly monitored**
  - RS periodically checks drivers status using nonblocking IPC
    - Queried driver must respond within next period
    - Nonblocking notification messages prevent clogging the system
  - RS immediately receives alert (SIGCHLD) from PM upon driver exit
    - RS is parent of all servers and drivers
- **Sec. 4.1: Fault model**
  - Crashes, panics, or unexpected exits
  - Attack failures such as ping of death
  - Byzantine or logical failures are excluded

Sec. 5: 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?

Sec. 5: RECOVERY PROCEDURE (2/3)

- **Sec. 5.1: 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
- **Sec. 5.2: Restarting dead drivers**
  - Full restart through VFS
  - Lightweight execution by RS to bypass VFS
    - Disk drivers shadowed in RAM to allow recovery

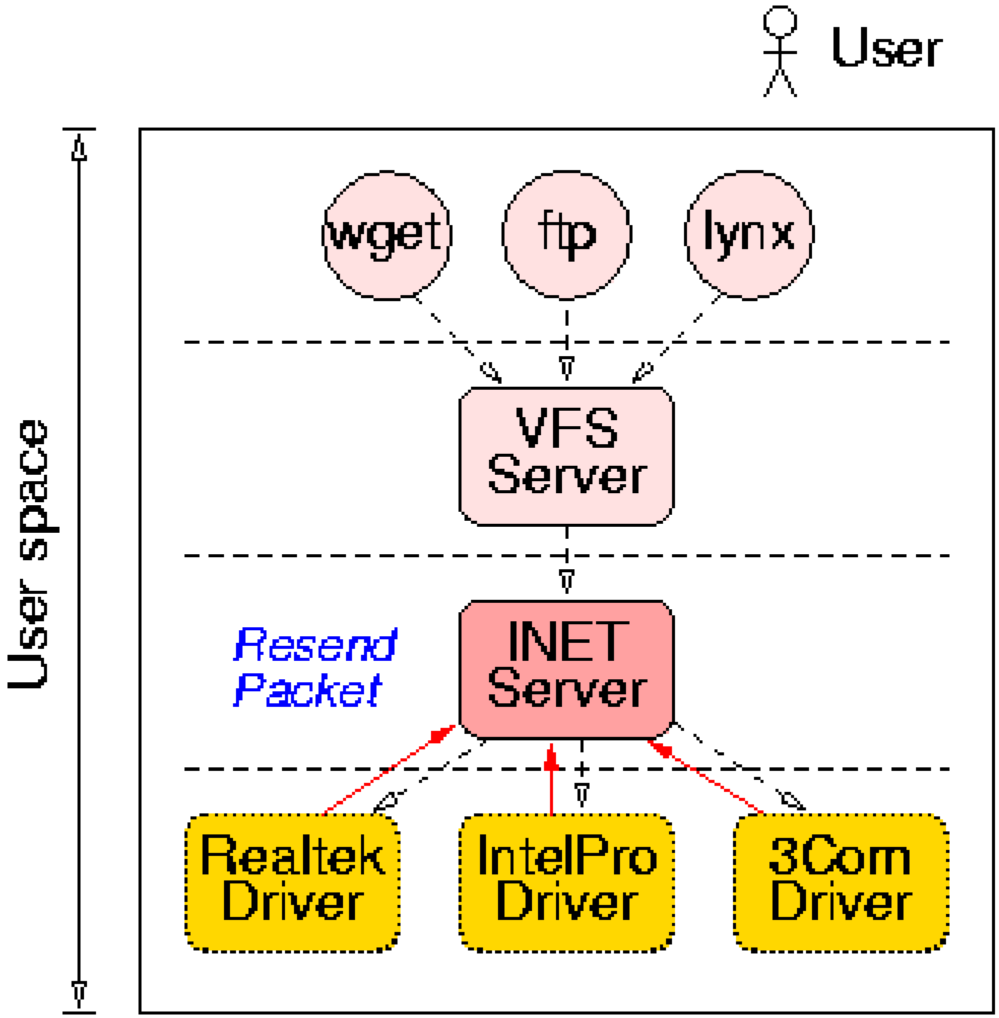
Sec. 5: RECOVERY PROCEDURE (3/3)

- **Sec. 5.3: 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
- **Sec. 5.4: 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

Sec. 6: EXAMPLES AND LIMITATIONS

- **Focus in on device drivers (worst problem)**
  - Sec. 6.1: Ethernet driver recovery
  - Sec. 6.2: Character driver recovery
  - Sec. 6.3: Disk driver recovery
- **Sec. 6.4: Recovery of failed servers**
  - Sometimes possible, depending on how much state is lost
    - Anything from user-supported recovery to transparent recovery
- **Sec. 6.5: Limitations of our system**
  - Failures in the core servers are fatal

Sec. 6.1: ETHERNET DRIVER RECOVERY



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

- **Recovery steps taken**

  (1) RS replaces dead driver

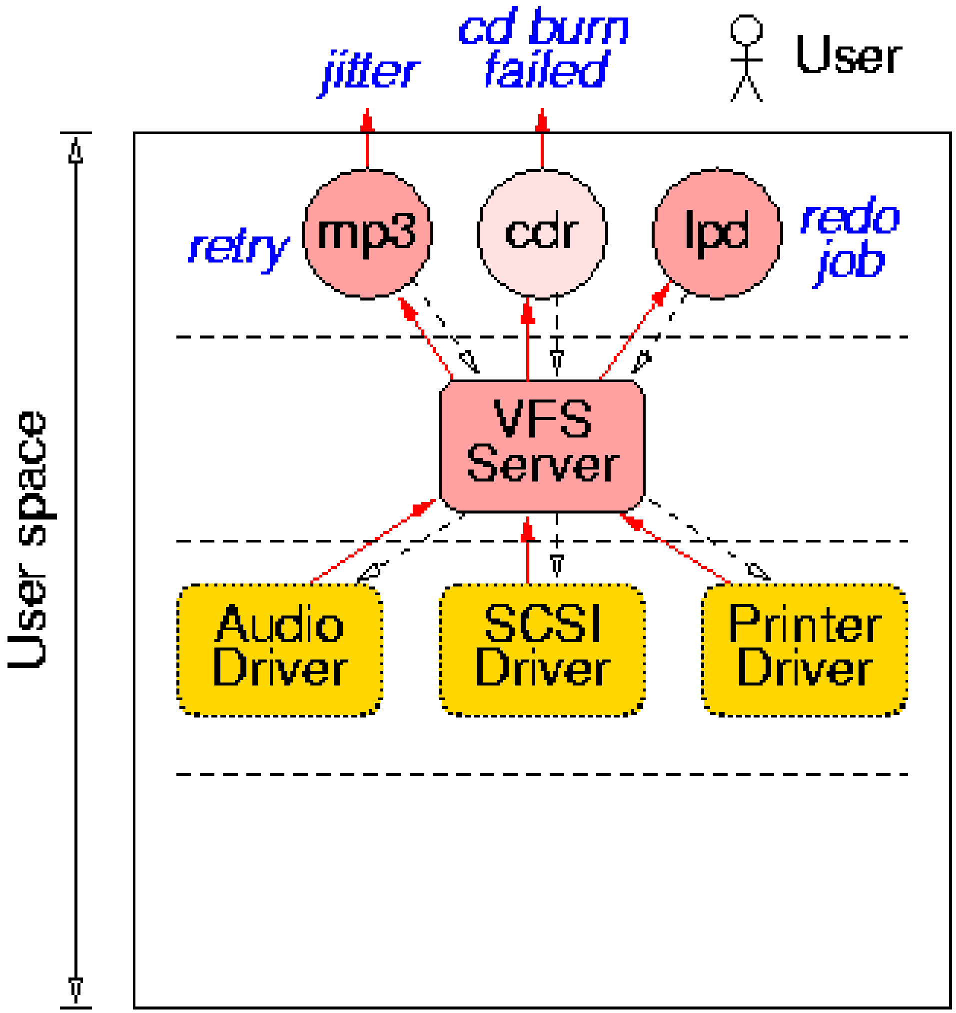
  (2) RS publishes update

  (3) DS informs INET server

  (4) INET reinitializes driver

  (5) INET resends lost data

Sec. 6.2: CHARACTER DRIVER RECOVERY



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

- **Recovery steps taken**

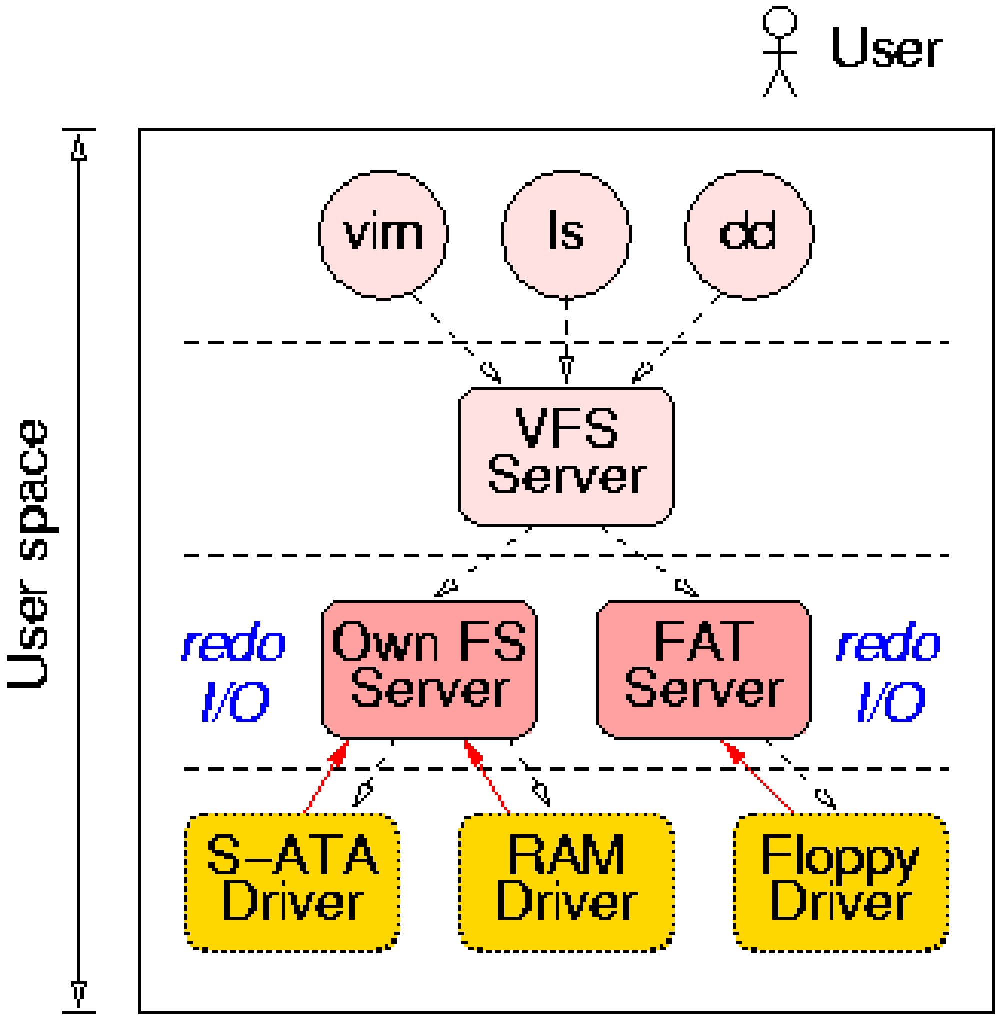
  (1) RS replaces dead driver

  (2) RS publishes update

  (3) DS informs VFS server

  (4) VFS returns I/O error to app

Sec. 6.3: BLOCK DRIVER RECOVERY (work-in-progress)



- **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

Sec. 7: DEPENDABILITY EVALUTION

- **Sec. 7.1: Fault-injection experiments**
  - To be done
- **Sec. 7.2: Recovery-overhead measurements**
  - 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

Sec. 8: PERFORMANCE

- **Performance measurements**
  - Time from multiboot monitor to login is under 5 sec.
  - The system can do a full build of itself within 4 sec.
  - Run times for typical applications: 6% overhead
  - File system and disk I/O performance: 9% overhead
  - Networking performance: Ethernet at full speed
- **Code size statistics**
  - Kernel is 3800 LOC; rest of the OS is in user space
  - Minimal POSIX-conformant system is 18,000 LOC

Sec. 9: DISCUSSION

- **Lessons learned**
  - Recovering lost state is one of the key problems
  - Integrated approach required for optimal results
    - E.g., servers and applications need to do recovery as well
- **General applicability**
  - User-mode drivers on Linux have been successfully tested
  - Our techniques can be applied to further improve dependability
  - Performance overhead is not a real issue

Sec. 10: CONCLUSIONS

- **We have built a fault-resilient OS**
  - Deals with an important problem, namely device driver failures
  - Defects are no longer fatal and transparent recovery is often possible
- **We have provided a concrete evaluation**
  - Fault-injection experiments and crash simulation prove viability
  - Performance overhead of 5-10% compared to base system
- **We have shown practicality of our approach**
  - Our techniques can be applied to of other systems, such as Linux
  - Limited costs make real-world adoption attractive

Sec. 11: ACKNOWLEDGEMENTS

- **John Wilkes (shepherd)**
- **The MINIX 3 team**
  - Ben Gras
  - Philip Homburg
  - Herbert Bos
  - Andy Tanenbaum

TIME FOR QUESTIONS & DISCUSSION

- **Sec. 12: Availability**
  - On the spot: MINIX 3.1.2 CD-ROM
  - Web: www.minix3.org
  - News: comp.os.minix
  - E-mail: jnherder@cs.vu.nl