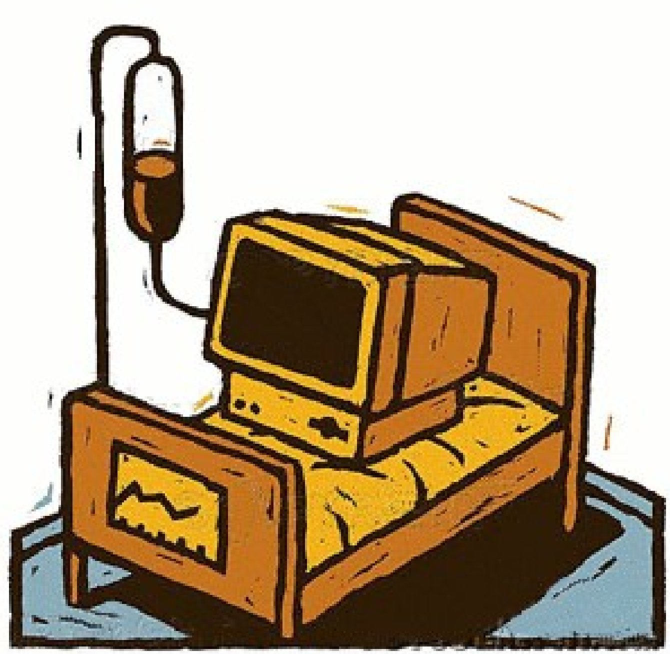
THE ARCHITECTURE OF A FAULT-RESILIENT OPERATING SYSTEM

ASCI 2006 Conference

Lommel, Belgium
June 15, 2006

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



WHAT'S IN FOR YOU?!

- **Reality check: *your* computer is broken ...**
  - Weak security and reliability
    - Application failures
    - Operating system failures **<-- current focus**
    - Digital pests (spyware, viruses, worms, etc.)
  - Enormous complexity
    - Hard to maintain and configure (even for IT professionals)
    - Too large for embedded and mobile computing

TALK OUTLINE

- Reality check (done)
- Introduction (next)
- Fault resilience
- Conclusion
- Questions

INTRODUCTION

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
- **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

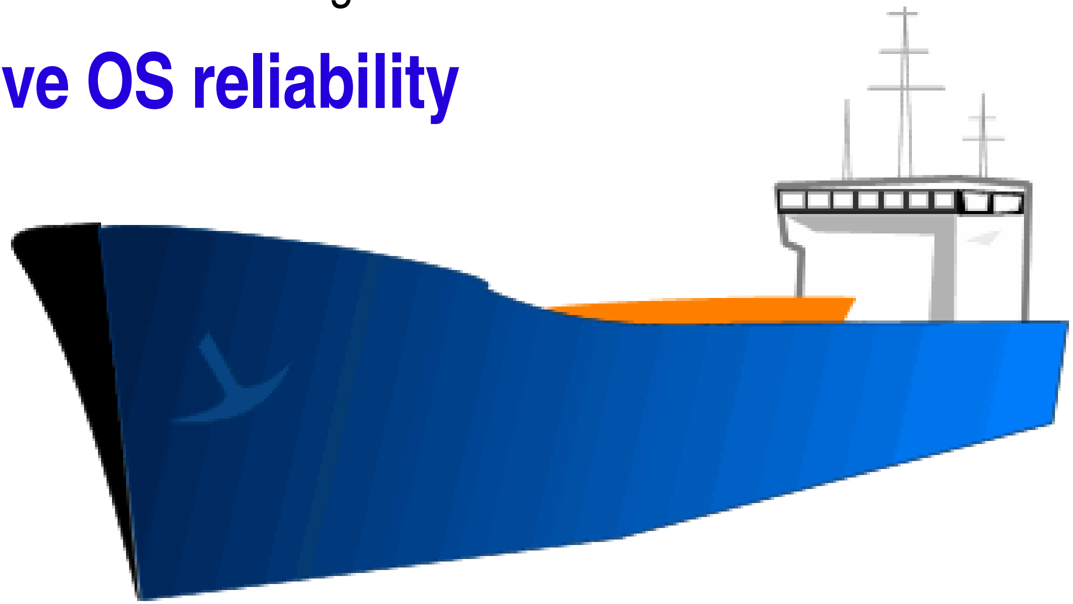
INHERENT PROBLEMS OF MONOLITHIC DESIGNS

- **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?

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**

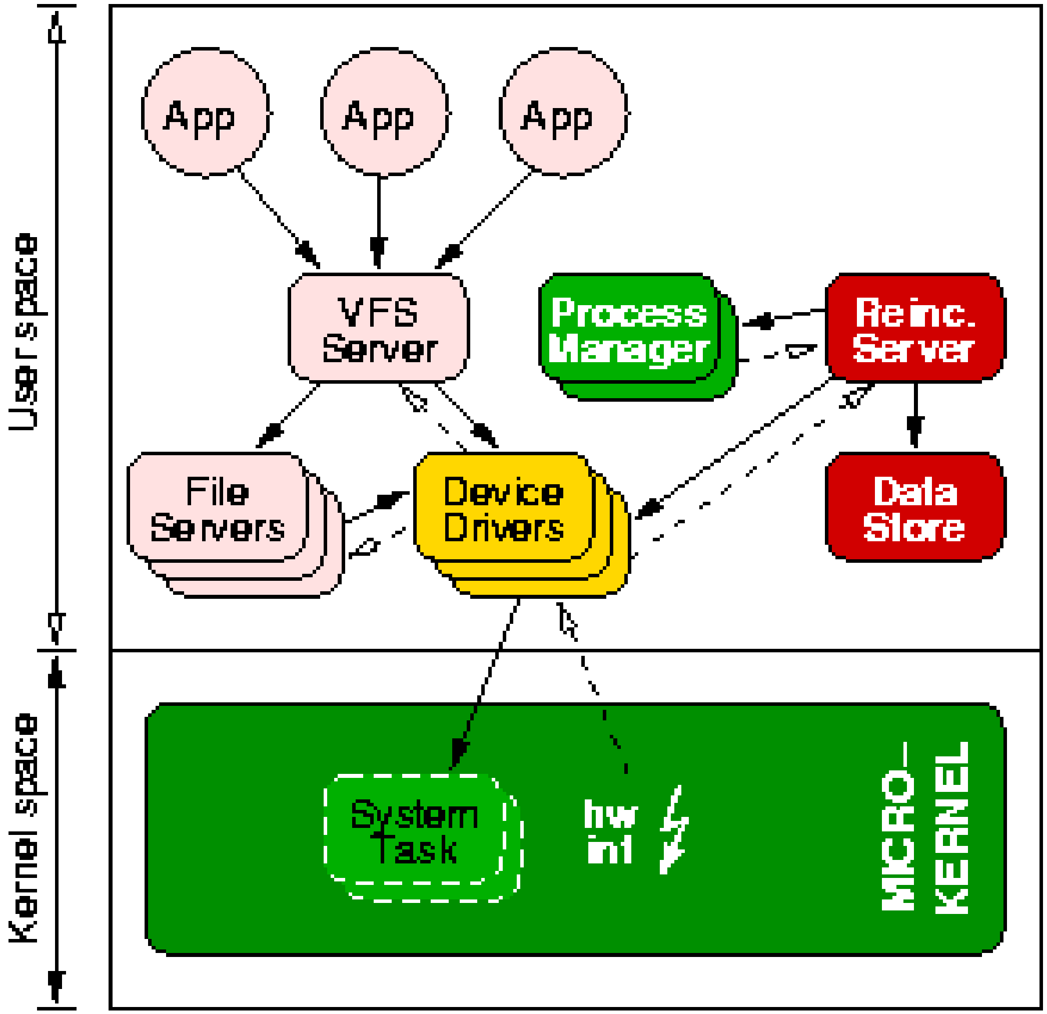


FAULT RESILIENCE

MINIX 3: A FAULT-RESILIENT OPERATING SYSTEM

- **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

HIGH-LEVEL DESIGN OF MINIX 3



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

RECOVERY PROCEDURE

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

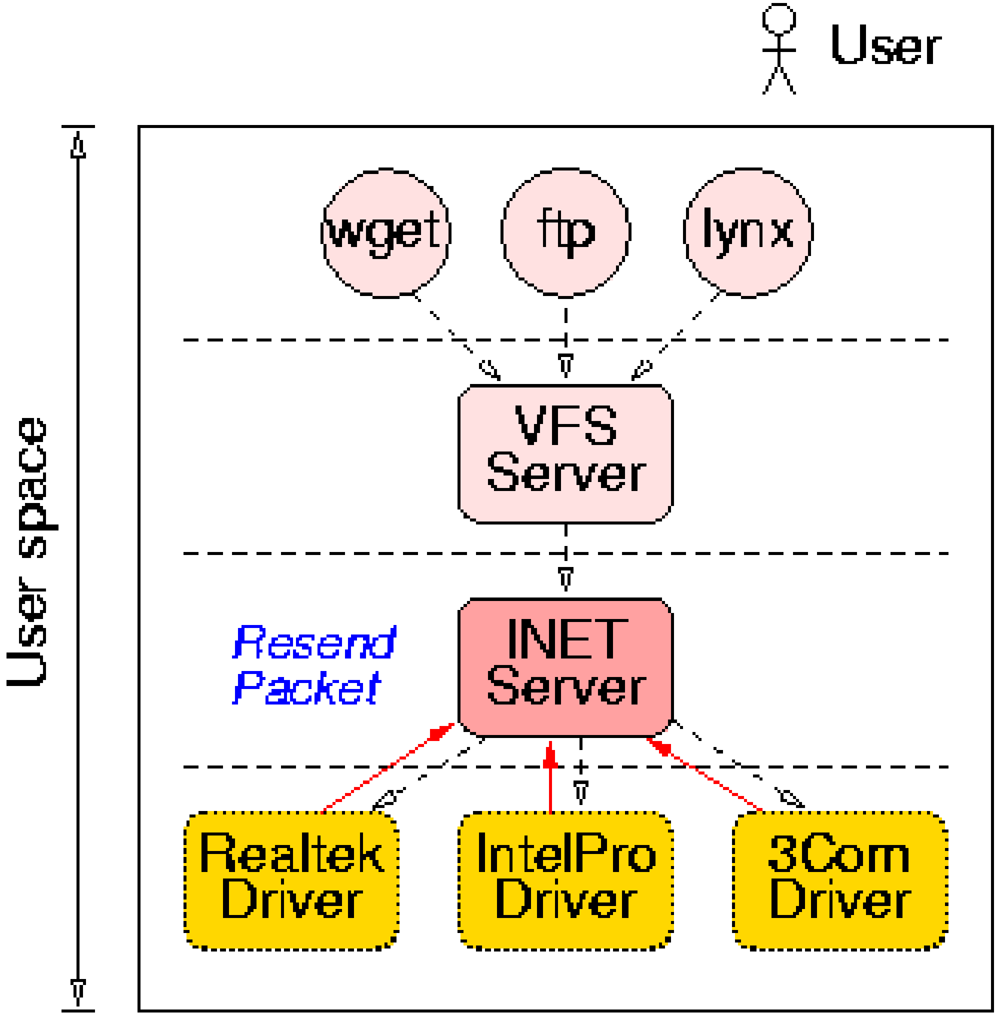
(1) Malfunctioning component is identified

(2) Associated recovery script is executed

(3) Component can be replaced with a fresh copy

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

EXAMPLE: ETHERNET DRIVER CRASH



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

- **Recovery steps taken**

  (1) Replace dead driver

  (2) Publish new configuration

  (3) INET notices update

  (4) INET reinitializes driver

  (5) INET resends lost data

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., recover from a file server failure
- **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

CONCLUSION

DISCUSSION

- **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

CONCLUSION

- **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

TIME FOR QUESTIONS & DISCUSSION

- **Try it yourself!**
  - MINIX 3 Live CD-ROM
- **More information**
  - Web: www.minix3.org
  - News: comp.os.minix
  - E-mail: jnherder@cs.vu.nl

- **The MINIX 3 team**
  - Ben Gras
  - Philip Homburg
  - Herbert Bos
  - Andy Tanenbaum