Dealing with Driver Failures in the Storage Stack

4th Latin-American Symposium on Dependable Computing

Jorrit N. Herder, David C. van Moolenbroek,
Raja Appuswamy, Bingzheng Wu,
Ben Gras, and Andrew S. Tanenbaum

vrije Universiteit amsterdam

Outline

- Device driver bugs and MINIX3
- Problem definition
- Solution: the “filter driver”
- Evaluation
- Conclusion

Outline

- **Device driver bugs and MINIX3**
- Problem definition
- Solution: the “filter driver”
- Evaluation
- Conclusion

Device driver bugs (1/2)

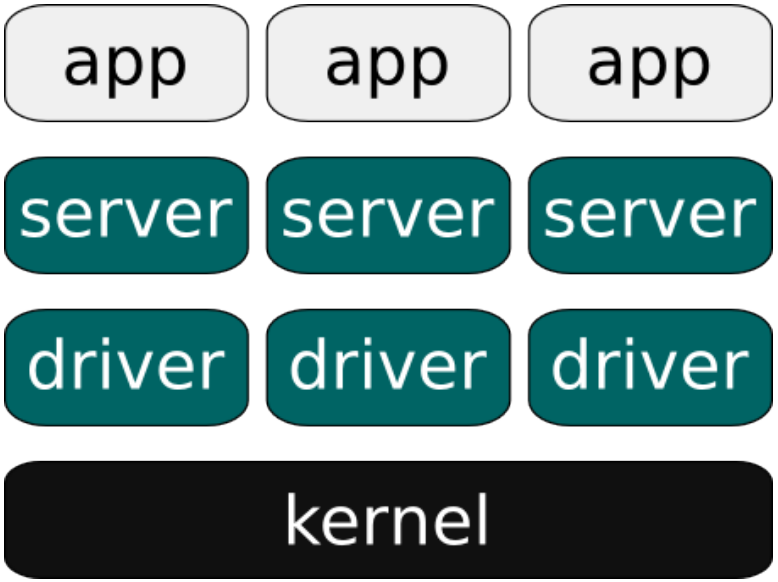
- Studies found 1 to 70 bugs per 1000 LoC
  - E.g., 3.35 bugs/1000 LoC for FreeBSD
- Large systems will always be buggy
  - Application bugs: bad
  - OS bugs: very bad
- Device drivers: main source of OS bugs
  - Drivers comprise up 70% of the OS
  - Drivers have 3-7x higher fault density

Device driver bugs (2/2)

- Driver bugs typically take down OS
  - In-kernel drivers have all powers of machine
  - Faults can propagate and cause global failure
- Up to 85% of all Windows XP crashes caused by drivers!

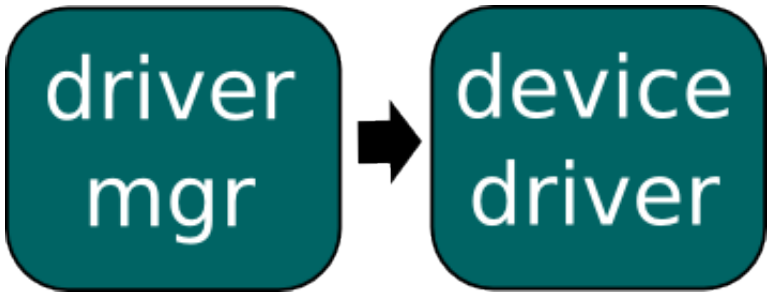
The MINIX3 OS

- MINIX3 multiserver operating system
  - Tiny (<5000 LoC) privileged microkernel
  - Ordinary processes make up rest of system
  - Message passing
- Focus: dependability



Device-driver management

- Device drivers are just processes
  - No more privileges than they need
  - May crash or hang
- Driver manager restarts failed drivers
  - Where possible, transparent to application



Outline

- Device driver bugs and MINIX3
- **Problem definition**
- Solution: the “filter driver”
- Evaluation
- Conclusion

Problem: restarts not enough!

- Today: focus on disk driver
  - Driver for hard disk
  - Reads and writes data blocks
  - Client is file server (FS)
- Bugs: more than crash/hang
  - Silent data corruption
  - Corrupted requests and replies
  - Timeouts

Driver threats (1)

- Driver may respond to request with “OK”
  - But.. may not have read/written the block
  - But.. may have read/written the wrong block
  - But.. may have garbled data
  - And some other variants

- These problems must be detected!

Driver threats (2)

- Driver may respond with “sorry, failure”
  - Legitimate failure or not?
  - What is expected driver behavior?
- Driver may not respond at all
  - Always bad

Missing infrastructure

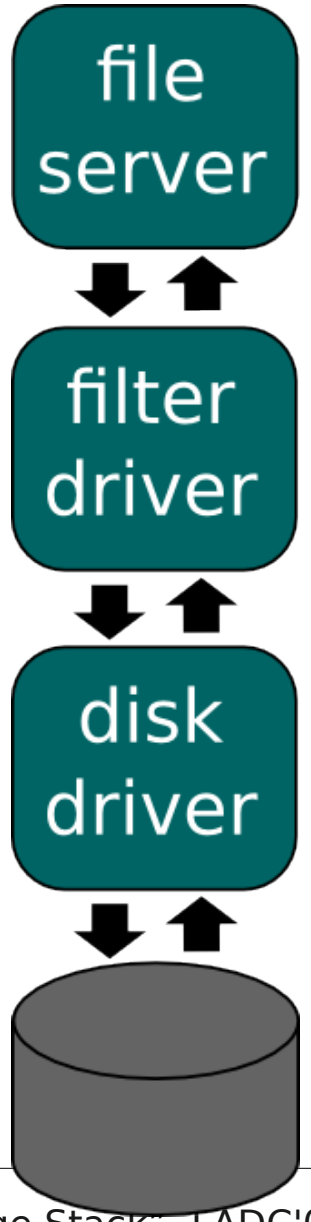
- We accept that a driver is buggy
- Driver client must handle threats
  - FS server cannot do this currently
- What is needed is:
  - End-to-end integrity for file system data
  - Semantic model of driver's working

Outline

- Device driver bugs and MINIX3
- Problem definition
- **Solution: the “filter driver”**
- Evaluation
- Conclusion

Our solution: the “filter driver”

- New driver process
  - Sits between FS and disk driver
  - Monitors all driver requests
- Purpose: detect and recover from disk driver bugs
- Can be (re)used with different file servers (FAT, ext-2, etc.)

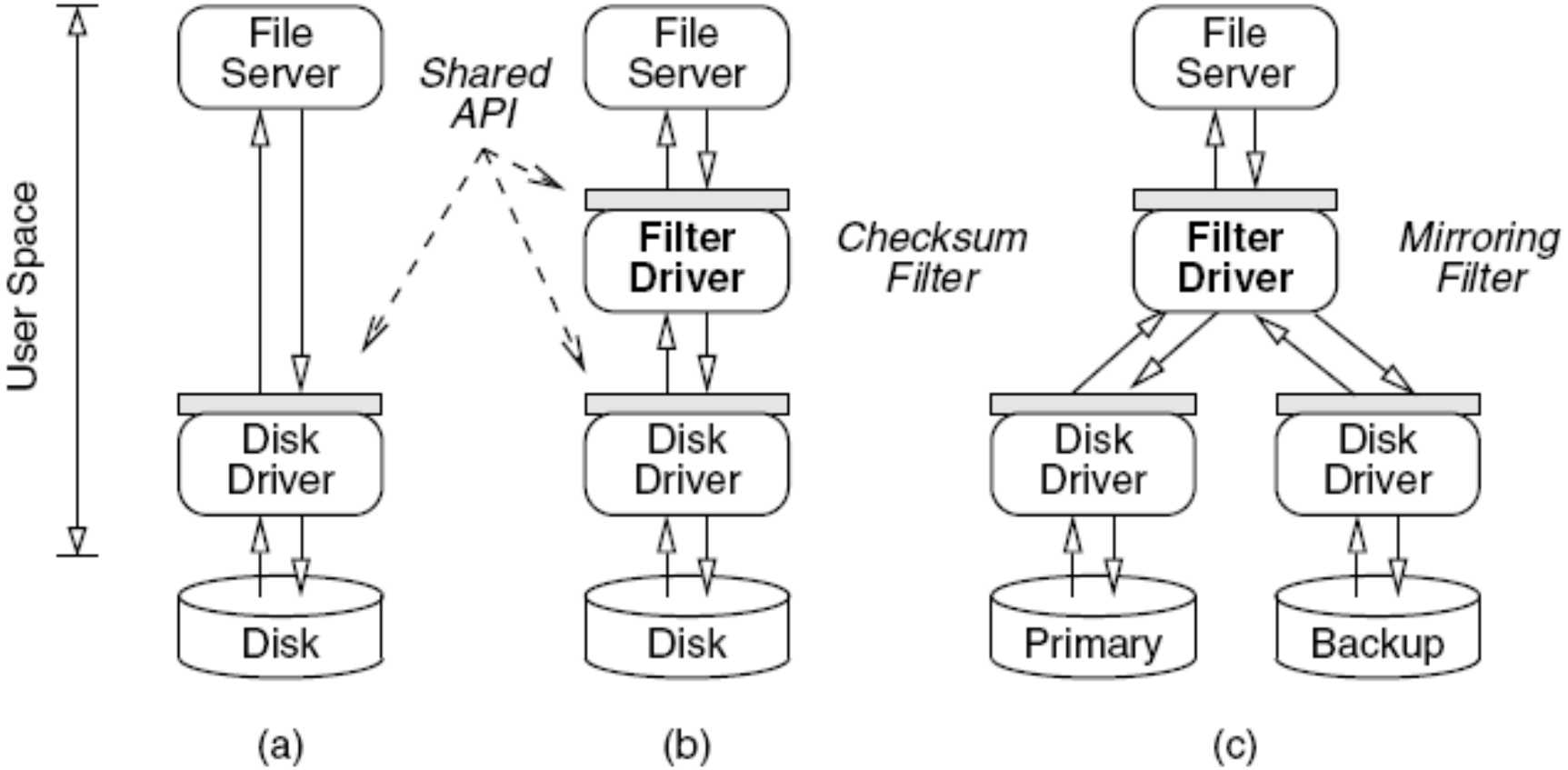


Working of the filter driver

- Filter driver transparently operates between file server and disk driver
  - Implements same API as disk driver
- Supports different protection strategies
  - Set alarm for timeouts
  - Check for expected result
  - Checksumming & mirroring

Filter driver configurations

- Checksumming, mirroring, or both

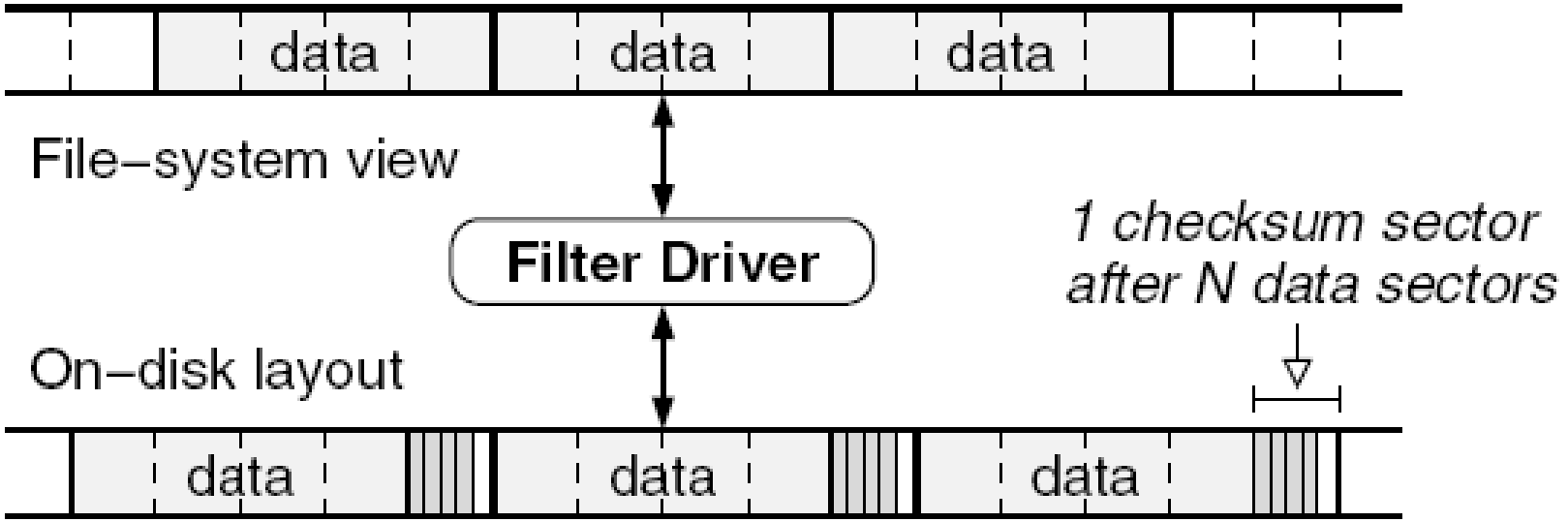


Checksumming (1/2)

- Checksum over each data sector
  - Sector number included in checksum
- Read request from FS:
  - Read both data and checksum
  - Check checksum
- Write request from FS:
  - Write both data and checksum
  - Read back checksum (hard requirement!)
  - Compare checksums

Checksumming (2/2)

- Filter driver presents virtual disk to FS
  - Virtual disk is smaller than real disk
  - Difference allows for storage of checksums



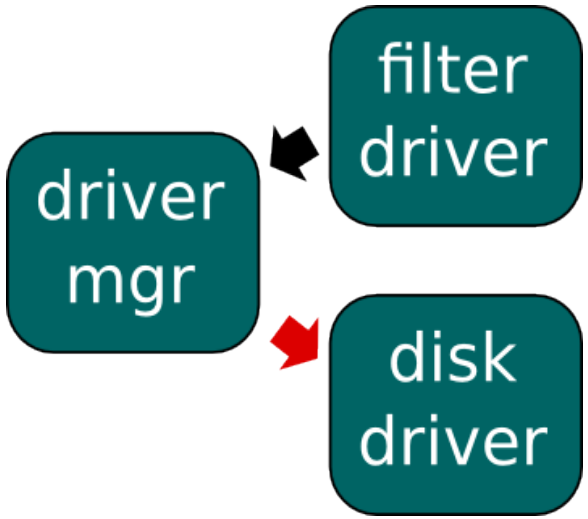
Detection of driver bugs

- Checksumming detects bad “OK” replies
- “Failure” replies only allowed if they are:
  - really invalid (persistent errors), or
  - requesting unaligned data, or
  - requesting beyond disk end
- No reply at all: schedule timeout alarm

- Filter driver detects ***all*** buggy behavior!

Recovery procedure

- If a problem is detected, what to do next?
- Three levels:
  1) retry request
  2) restart driver
  3) switch to mirror
- Done transparently to FS
- Only if **nothing** helps, tell FS



Outline

- Device driver bugs and MINIX3
- Problem definition
- Solution: the “filter driver”
- **Evaluation**
- Conclusion

Experimental evaluation

- Implemented prototype in MINIX3
  - Part of Google Summer of Code 2008
- Experiments run on the prototype:
  - Performance measurements
  - Fault-injection testing
  - Analysis of engineering effort

Performance measurements

- Analyzed different filter configurations
  - No filter, null filter (only forward data), mirror, checksum, mirror+checksum
    - Checksumming costs most visible
- Analyzed different workloads
  - Writes show more overhead than reads
    - Read back of data for verification costly
  - CPU-bound jobs outperform I/O-bound jobs
    - Filter costs amortized over other processing

Application benchmarks

- Application workload is dominating factor
- User-perceived overhead is 0-28%

Benchmark	No Filter		Mirror+Checksum	
Copy root FS	14.89	(1.00)	18.34	(1.23)
Find and touch	2.75	(1.00)	2.91	(1.06)
Build libraries	28.84	(1.00)	28.72	(1.00)
Build MINIX 3	14.26	(1.00)	14.86	(1.04)
Copy source tree	2.54	(1.00)	3.26	(1.28)
Find and grep	5.16	(1.00)	5.67	(1.10)
File system check	3.46	(1.00)	3.91	(1.13)
Delete root FS	10.72	(1.00)	13.07	(1.22)

Fault-injection testing

- Manual fault injection
  - Small number of manually inserted faults
  - Check for specified behavior
- Automated fault injection using Software-implemented fault injection (SWIFI)
  - Faults representative for programming errors
  - Injected faults into binary of running driver
  - Driver workload: read/write data from/to disk

Effects of fault injection

- Faults in disk driver induce failures
- Filter driver retries failed operations
- Driver manager also may be involved
  - Unexpected disk driver exit or panic
  - Disk driver crashed due to exception
  - Missing disk driver heartbeat
  - Filter driver complains about problem
    - Only if retry attempt was not successful

SWIFI test results (1/2)

SWIFI test run	**#1**	**#2**	**#3**	**SUM**
- Total faults injected	1000	1000	1000	3000
SATA driver restarts	33	31	30	94
- Driver exit due to panic	5	7	5	17
- Crashed due to exception	9	5	8	22
- Missing driver heartbeat	1	4	1	6
- Filter-driver complaint	18	15	16	49
Problems detected by filter	92	88	14	194
- Request undeliverable	0	1	1	2
- Timeout receiving reply	18	14	17	33
- Unexpected IPC reply	24	33	33	60
- Legitimate request failed	35	40	38	78
- Bad checksum detected	15	0	6	21
- Read-after-write failed	0	0	0	0

SWIFI test results (2/2)

- Filter driver software was very successful
- Disk controller hardware was not:
  - Sitecom Serial ATA PCI RAID controller had problems handling the fault load
    - Controller confused after driver restart
    - Controller caused PCI bus hang
  - Full system reset required in these cases
- Note: this is a hardware shortcoming rather than a problem with our design

Engineering effort

- One-time effort: filter driver works with different file servers (FAT, ext-2, etc.)
- Trusted code base reduced by >50%

Part	Total Lines	Code Lines	Complexity
at_wini.c	2726	1923	223
libdriver.c	740	428	6
filter.c	1573	1030	209

Outline

- Device driver bugs and MINIX3
- Problem definition
- Solution: the “filter driver”
- Evaluation
- **Conclusion**

Conclusion

- Novelty: disk driver no longer trusted
- Filter driver transparently implements various protection strategies
  - End-to-end integrity for file system data
  - Semantic model of disk driver's working
- Detect and recover from all driver bugs
- Sacrifice disk space and performance for improved dependability

EOF

Thank you! Questions?