Ivannikov Memorial Workshop
Yerevan, Armenia
03 May 2018

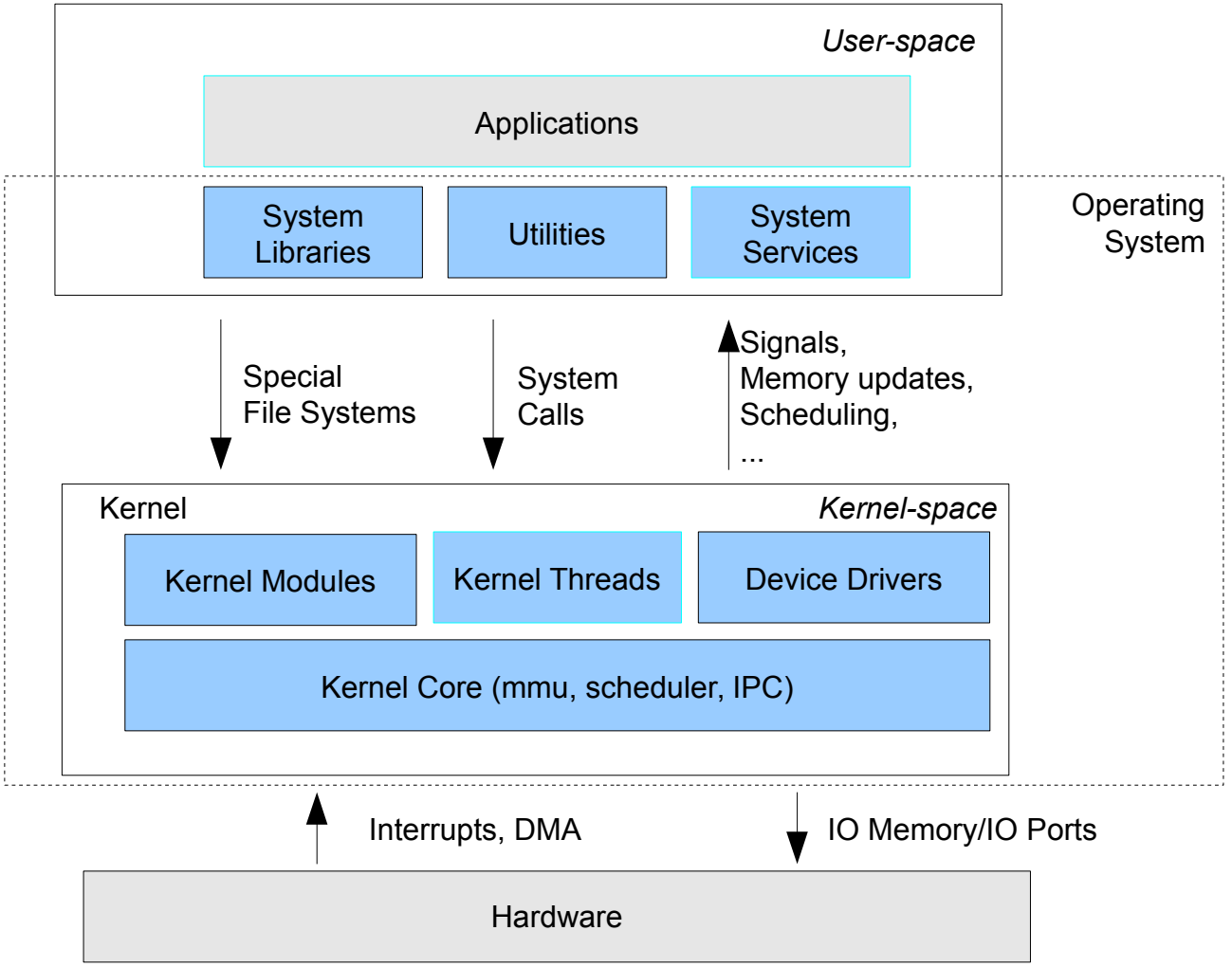
LinuxTesting.org

Key Aspects of Operating System Testing

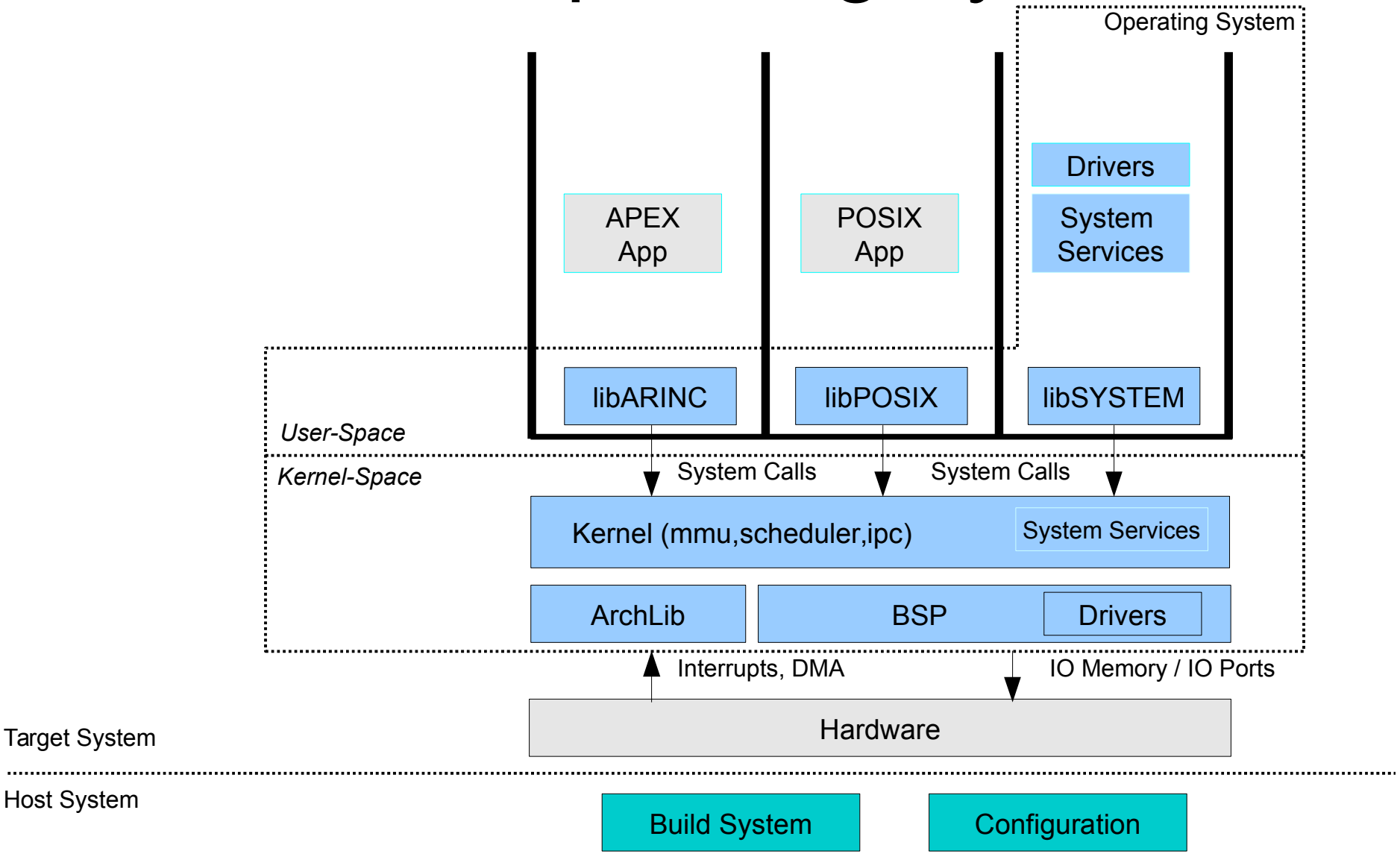
Alexey Khoroshilov
khoroshilov@ispras.ru

ISP RAS
Ivannikov Institute for System Programming
of the Russian Academy of Sciences

Operating Systems



Embedded Operating Systems



Operating System Specifics

- HW manager
  - dependence on HW and its configurations
  - internal activity
  - internal parallelism
- Cornerstone of software system
  - correct handling of any input/userspace behaviour
  - tolerance to unusual events
    - e.g. resource exhaustion
  - long run time
    - => resources leaks are unacceptable
  - minimal overhead

Operating System Specifics (2)

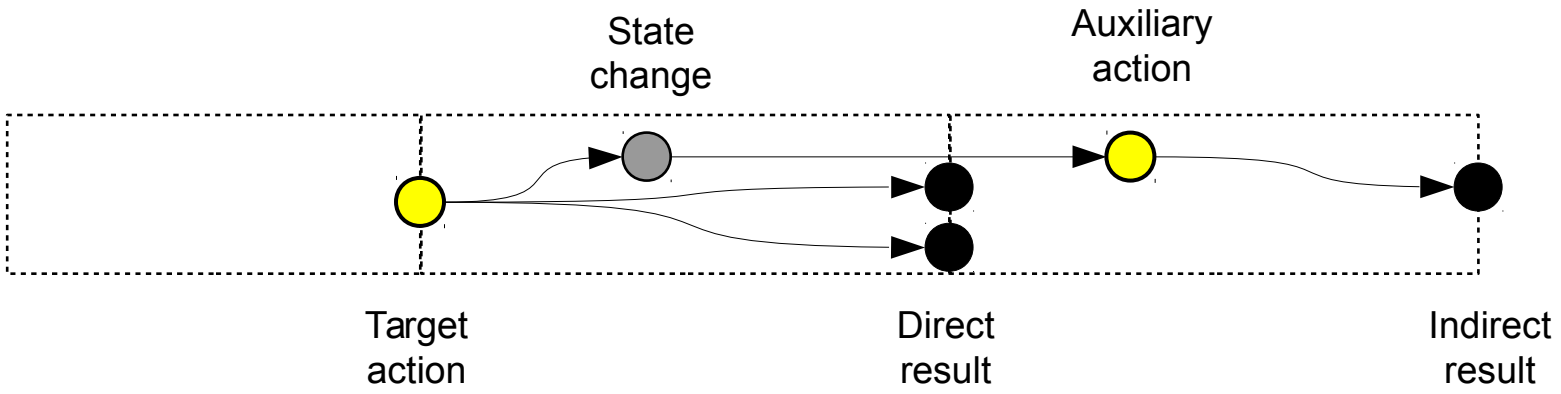
- Environment for application software
  - compliance to standard API specifications
  - compliance to API documentation
  - API/ABI forward/backward compatibility
- Execution environment for test system
  - minimal influence of test system to functionality under test
  - faults in OS should not be lost

Goals of Testing

- Requirements checking
  - Functional requirements
  - Information flow restrictions
  - Probabilistic requirements
- Anomaly detection
  - Assertion failed
  - Programming language/HW bad event
    - Invalid memory access
    - Unspecified behaviour
    - ...
  - Resource leak
  - Data race

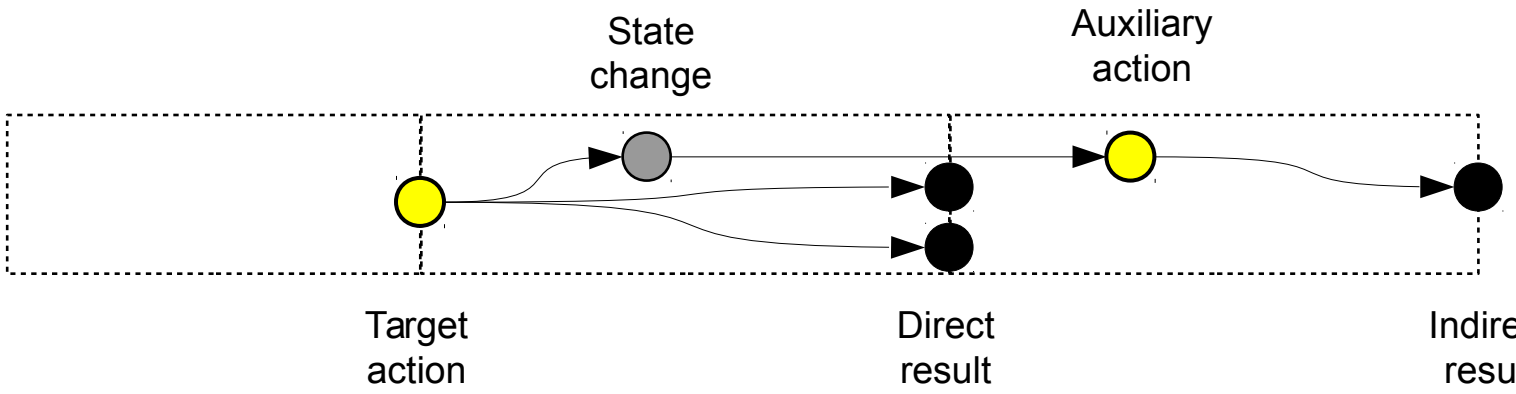
Functional Requirement Model

If event 'Target action' under some conditions happens, then SUT **have to** do something.



Functional Requirement Model

If event 'Target action' under some conditions happens,
then SUT **have to** do something.



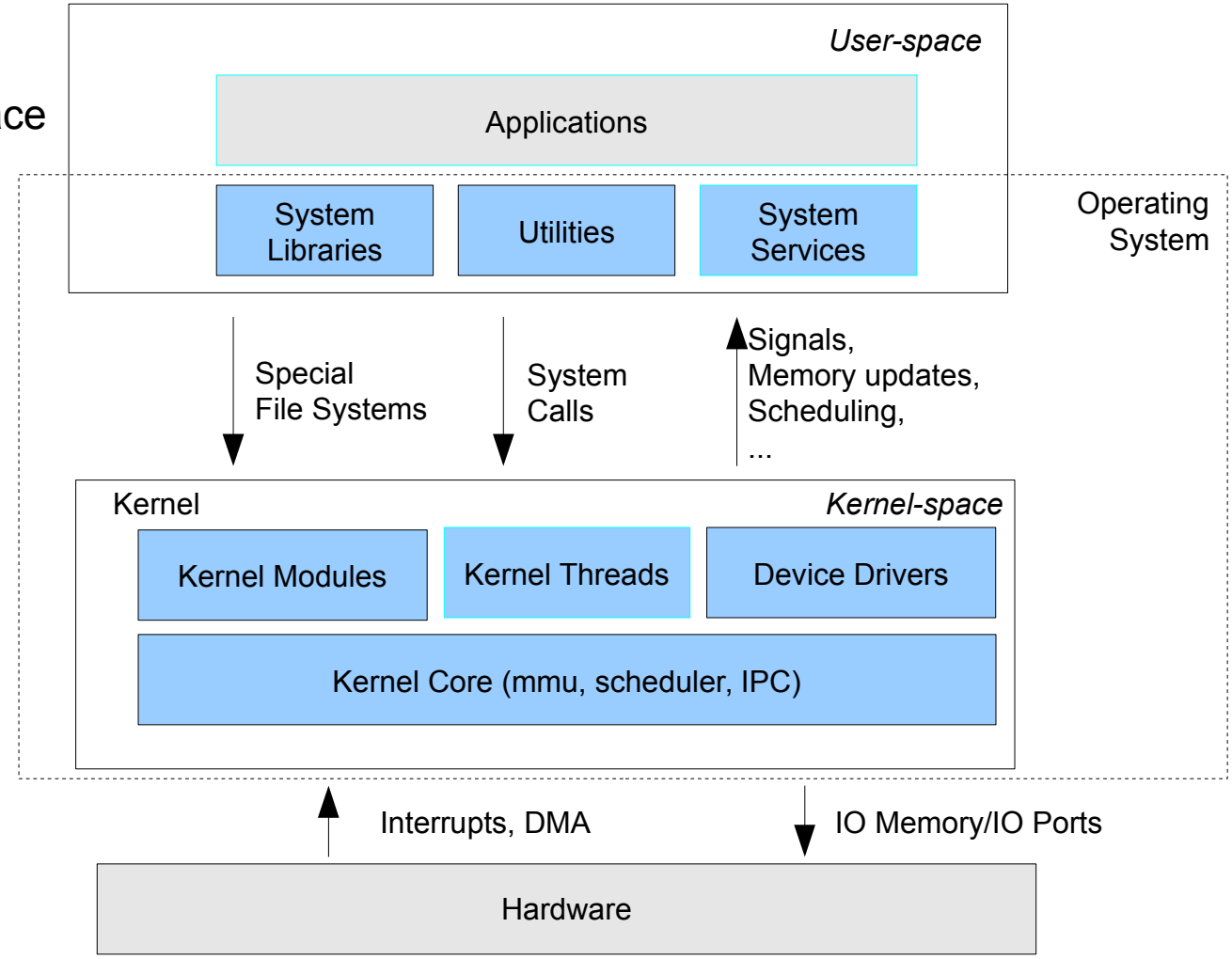
Prepare test situation
Iterate test situations

Influence SUT, e.g.:
- fault injection
- interrupt injection
- context switch

Post actions

Kinds of Test Actions

- Test Actions
  - application interface
  - HW interface
  - internal actions
    - inside
    - outside

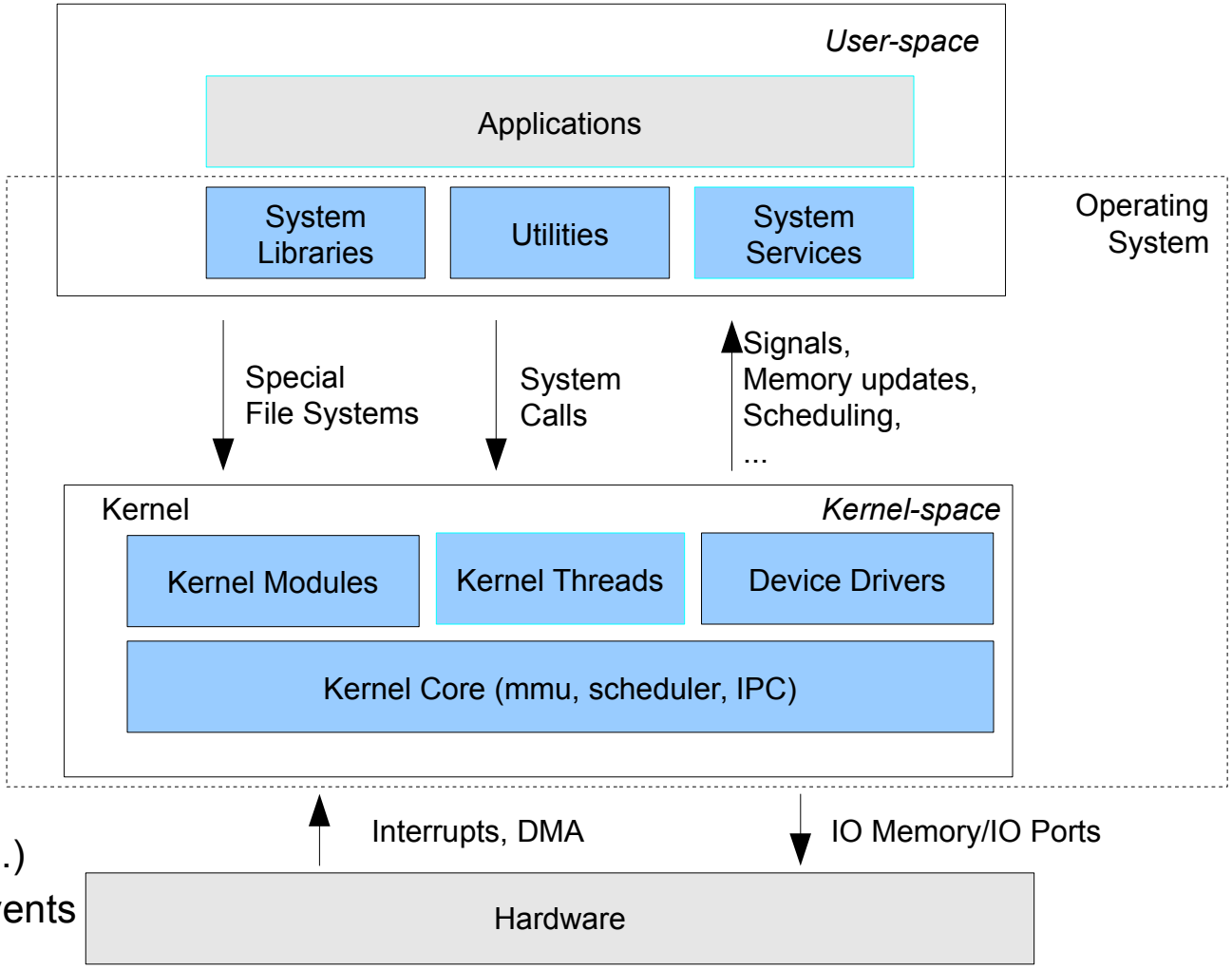


Active Aspects

- Target Test Situations Set
  - requirements coverage
  - class equivalence coverage
  - model coverage (of SUT or reqs)
  - source code structure coverage
  - data flow coverage
- Test Situations Setup/Set Generation
  - passive
  - fixed scenario
    - manual
    - pre-generated
      - coverage driven
      - random
  - adapting scenario
    - coverage driven
      - source code coverage
      - model/... coverage
    - random
- Test Actions
  - application interface
  - HW interface
  - internal actions
    - inside
    - outside

Monitoring Aspects

- Kinds of Observable Events
  - interface events
  - internal events
- Events Collection
  - internal
  - external
  - embedded
- Events Analysis
  - online
    - in-place
    - outside
  - offline
- Requirements Specification
  - in-place (local, tabular)
  - formal model (pre/post+invariants,...)
  - assertions/prohibited events



Monitoring Aspects

- Kinds of Observable Events
  - interface events
  - internal events
- Events Collection
  - internal
  - external
  - embedded
- Events Analysis (for verdict, for coverage)
  - online
    - in-place
    - outside
  - offline
- Requirements Specification
  - in-place (local, tabular)
  - formal model (pre/post+invariants,...)
  - assertions/prohibited events

LinuxTesting.org

Robustness Testing

ISP RAS
Ivannikov Institute for System Programming
of the Russian Academy of Sciences

Fault Handling Code

- Is not so fun
- Is really hard to keep all details in mind
- Practically is not tested
- Is hard to test even if you want to
- **Bugs seldom(never) occurs**

  => low pressure to care

Why do we care?

- It beats someone time to time
- Safety critical systems
- Certification authorities

Run-Time Testing of Fault Handling

- Manually targeted test cases
  - \+ The highest quality
  - – Expensive to develop and to maintain
  - – Not scalable
- Random fault injection on top of existing tests
  - \+ Cheap
  - – Oracle problem
  - – No any guarantee
  - – When to finish?

Systematic Approach

- Hypothesis:
  - Existing test lead to more-or-less deterministic control flow in kernel code
- Idea:
  - Execute existing tests and collect all potential fault points in kernel code
  - Systematically enumerate the points and inject faults there

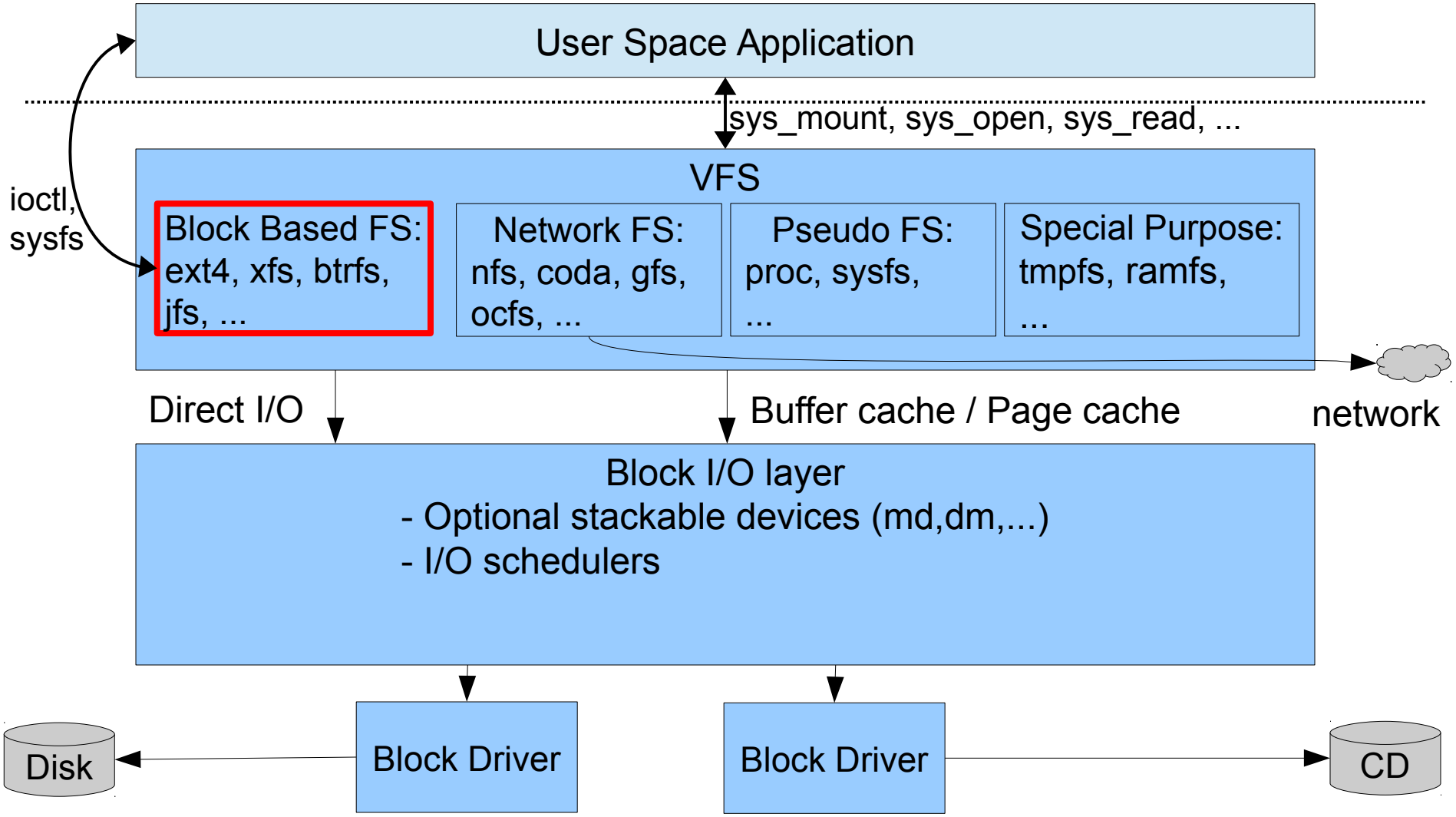
Experiments – Outline

- Target code
- Fault injection implementation
- Methodology
- Results

Experiments – Target

- Target code: file system drivers
- Reasons:
  - Failure handling is more important than in average
    - Potential data loss, etc.
  - Same tests for many drivers
  - It does not require specific hardware
  - Complex enough

Linux File System Layers



File System Drivers - Size

File System Driver	Size, LoC
JFS	18 KLOC
Ext4	37 KLoC *with jbd2*
XFS	69 KLoC
BTRFS	82 KLoC
F2FS	12 KLoC

File System Driver – VFS Interface

- file_system_type
- super_operations
- export_operations
- inode_operations
- file_operations
- vm_operations
- address_space_operations
- dquot_operations
- quotactl_ops
- dentry_operations

~100 interfaces in total

FS Driver – Userspace Interface

File System Driver	ioctl	sysfs
JFS	6	-
Ext4	14	13
XFS	48	-
BTRFS	57	-

FS Driver – Partition Options

File System Driver	mount options	mkfs options
JFS	12	6
Ext4	50	~30
XFS	37	~30
BTRFS	36	8

FS Driver – On-Disk State

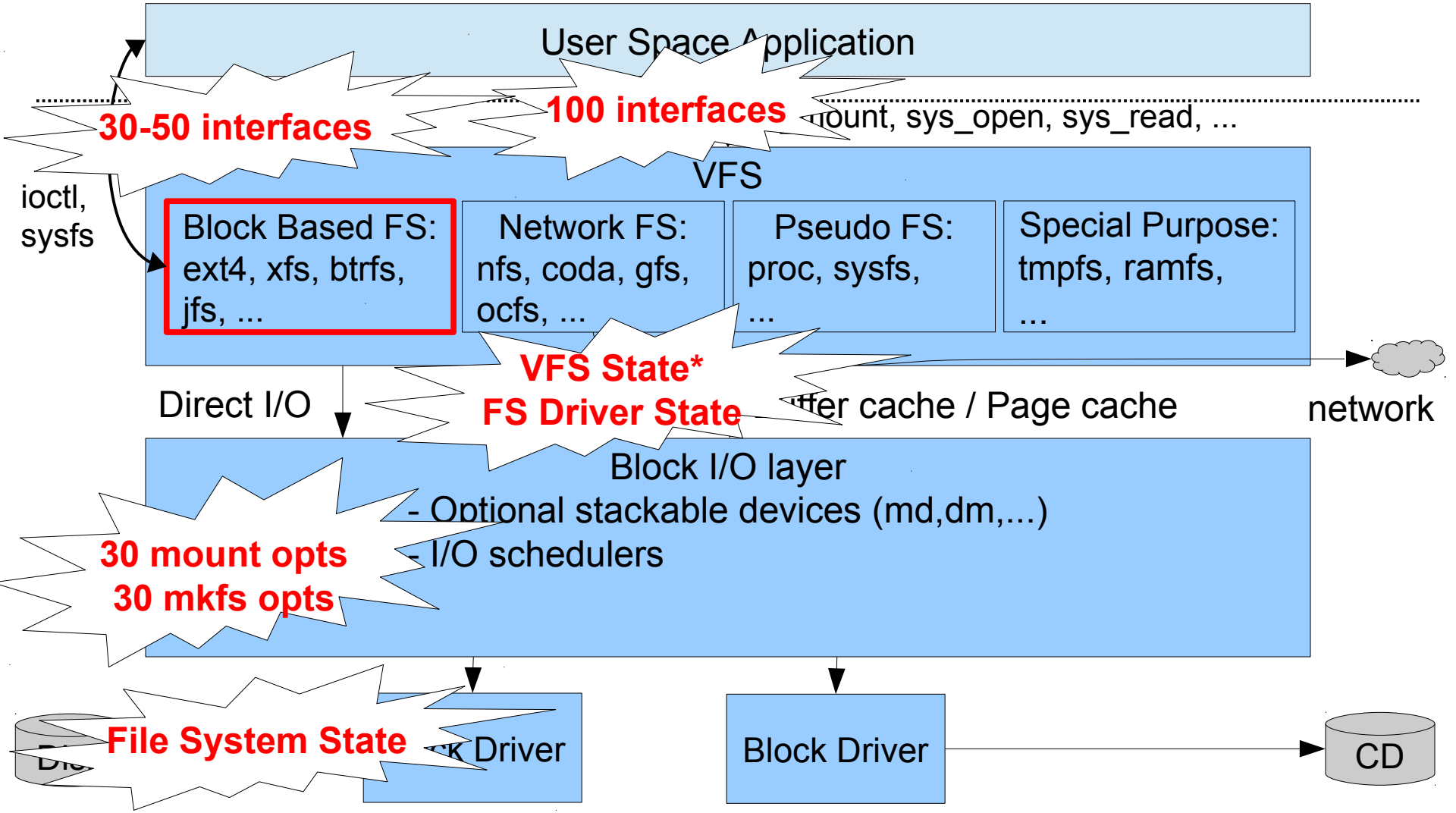
File System Hierarchy

* File Size
* File Attributes
* File Fragmentation
* File Content (holes,...)

FS Driver – In-Memory State

- Page Cache State
- Buffers State
- Delayed Allocation
- ...

Linux File System Layers



FS Driver – Fault Handling

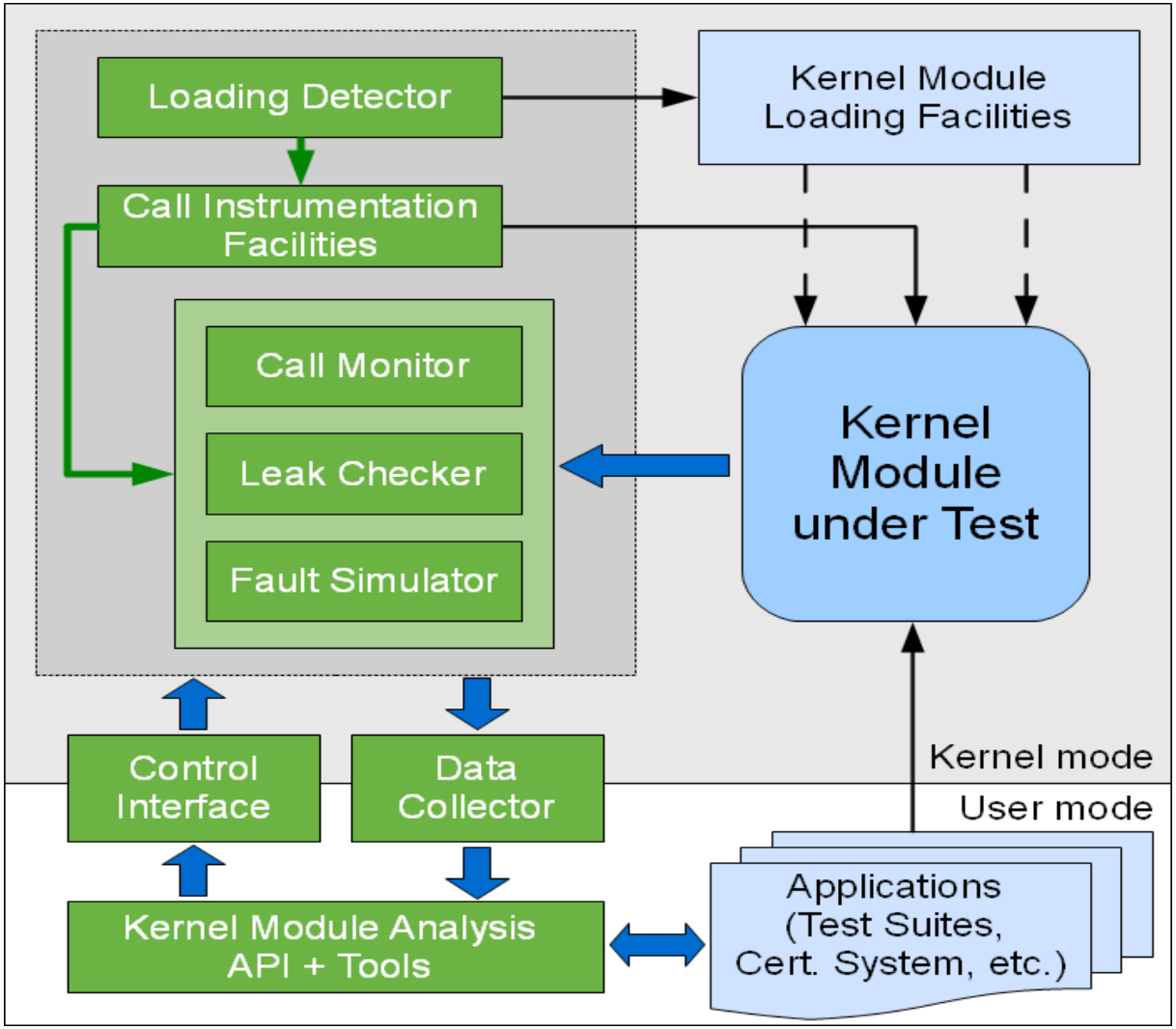
- Memory Allocation Failures
- Disk Space Allocation Failures
- Read/Write Operation Failures

Fault Injection - Implementation

- Based on KEDR framework*
  - intercept requests for memory allocation/bio requests
    - to collect information about potential fault points
    - to inject faults
  - also used to detect memory/resources leaks

(*) http://linuxtesting.org/project/kedr

KEDR Workflow



http://linuxtesting.org/project/kedr

Experiments – Oracle Problem

- Assertions in tests are disabled
- Kernel oops/bugs detection
- Kernel assertions, lockdep, memcheck, etc.
- Kernel sanitizers
- KEDR Leak Checker

Methodology – The Problem

- Source code coverage is used to measure results on fault injection
- If kernel crashes code, coverage results are unreliable

Methodology – The Problem

- Source code coverage is used to measure results on fault injection
- If kernel crashes code, coverage results are unreliable
- As a result
  - Ext4 was analyzed only
  - XFS, BTRF, JFS, F2FS, UbiFS, JFFS2 crashes and it is too labor and time consuming to collect reliable data

Experiment Results

Systematic vs. Random

	Increment *new lines*	**Time** *min*	**Cost** *second/line*
Xfstests without fault simulation	-	2	-
Xfstests+random(p=0.005,repeat=200)	411	182	27
Xfstests+random(p=0.01,repeat=200)	380	152	24
Xfstests+random(p=0.02,repeat=200)	373	116	19
Xfstests+random(p=0.05,repeat=200)	312	82	16
Xfstests+random(p=0.01,repeat=400)	451	350	47
Xfstests+stack filter	423	90	13
Xfstests+stackset filter	451	237	31

Systematic vs. Random

+ 2 times more cost effective
+ Repeatable results
– Requires more complex engine

+ Cover double faults
– Unpredictable
– Nondeterministic

Test Aspects (1)

ISP RAS

Active Aspects	T2C	OLVER	Autotest	Cfg	FI	KEDR-LC	S2E	RH	KStrider
				+-	+	-	+	+	-
Target Test Situations Set				cfgs				Specific	
requirements coverage	+	+							
class equivalence coverage		+							
model coverage (SUT/reqs)		+							
source code coverage					almost		+		
Test Situations Setup/Set Gen									
passive								+-	
fixed scenario	+		+						
manual	+								
pre-generated									
coverage driven				+-					
random			+-						
adapting scenario		+							
coverage driven		+							
source code coverage					almost		+		
model/... coverage		+							
random					as option				
Test Actions									
application interface	+	+	+						
HW interface									
internal actions					+		+	+	
inside					+			+	
outside							+		

Test Aspects (2)

Monitoring Aspects	T2C	OLVER	Autotest	Cfg	FI	KEDR-LC	S2E	RH	KStrider
				-	-	+	+-	+	+-
Kinds of Observable Events									
interface events	+	+	+						
internal events						+	+	+	+
Events Collection									
internal	+	+	+			+			+
external							+		
embedded									
Requirements Specification						Specific	Plugin	Specific	Specific
in-place (local, tabular)	+		+	If	Dis		Dis		
formal model (pre/post+invariants,...)		+		If	Co		Co		
assertions/prohibited events	External	External	External	Co	Co		Co		
Events Analysis									
online	+	+	+						
in-place	+		+			+		+	
outside		+							
offline									+

Experience (RTOS)

RTOS	Company	Application Domain
OC2000/OC3000	NIISI RAS	submarines, Su-35, ...
BagrOS	OKB Sukhoi	Su-57
RelMK-653	RPKB	
MOS-OP	Aviaavtomatika	
EOS	Elektroavtomatika	Tu-160M2
***	NTC Module	Luna-Glob
JetOS	ISPRAS	Civil aviation

Experience (Linux)

- LSB (Linux Foundation) – LSB Compliance Test Suite and Infrastructure
  - > 100 bugs in libraries, > 150 bugs in specifications
  - http://linuxtesting.org/lsb_infrastructure
- Linux Driver Verification (MinObrNauki, OSADL)
  - http://linuxtesting.org/ldv
  - > 300 bugs in Linux kernel fixed
- AstraLinux (RusBITech) – Custom Linux Security Module
  - http://linuxtesting.org/astraver
  - Security Policy Model verification
  - Deductive verification of LSM
- Alt Linux (BaseAlt) – SELinux
  - Security Policy Model development and verification

Thank you!

Alexey Khoroshilov
khoroshilov@ispras.ru
http://linuxtesting.org/

ISP RAS
Ivannikov Institute for System Programming
of the Russian Academy of Sciences

Math

ISP RAS
Ivannikov Institute for System Programming
of the Russian Academy of Sciences

Test Results: Details