Techniques for the Abstraction of System Call Traces

Waseem Fadel

Abdelwahab Hamou-Lhadj

Software Behaviour Analysis Group

Concordia University

Montréal, QC, Canada

{w_fadel, abdelw}@ece.concordia.ca

Tracing and Monitoring Distributed Multi-Core Systems

Mid-Project Meeting, December 2010

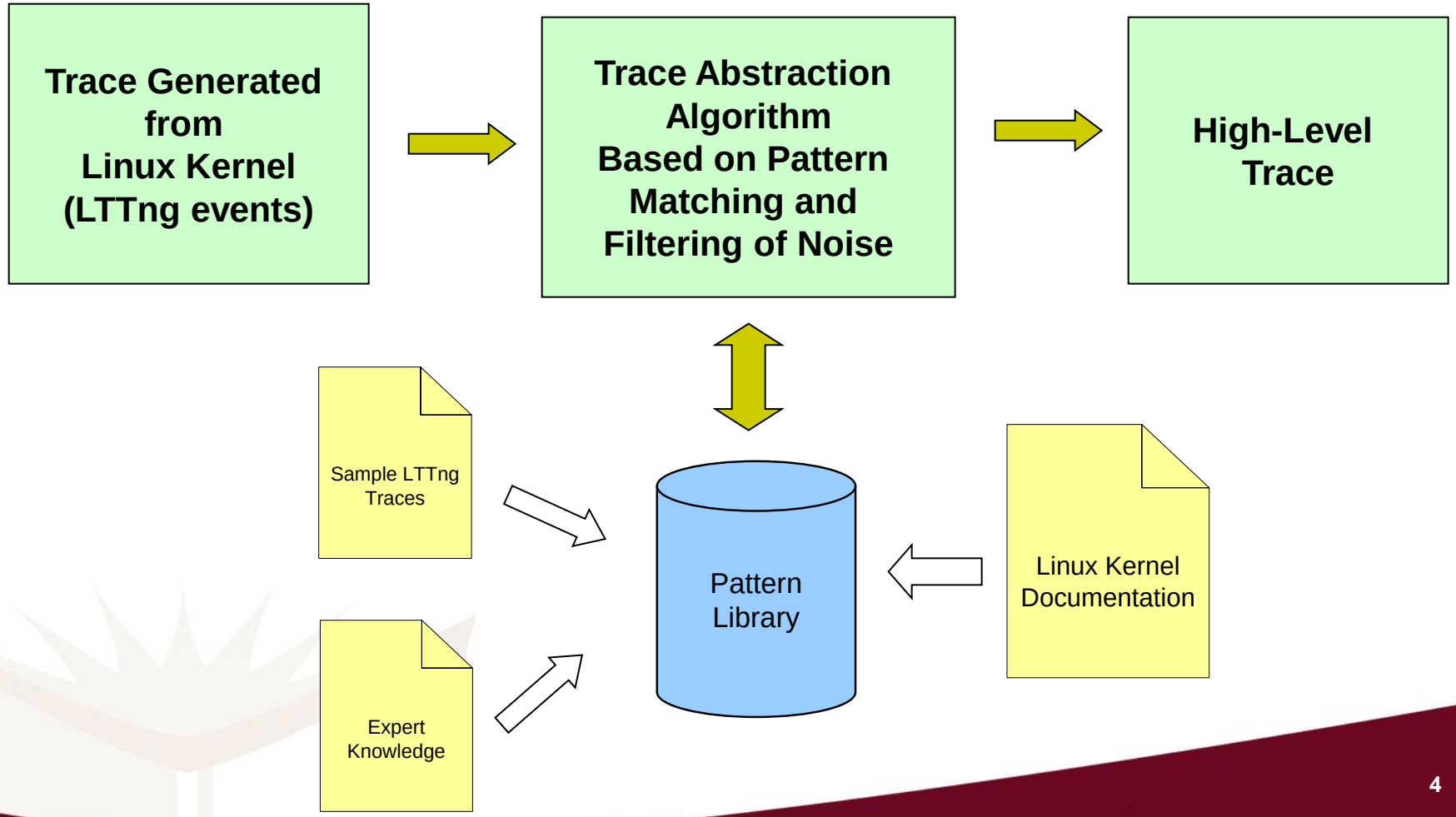
Agenda

- Objective
- Approach
- Progress since the last meeting
- Remaining challenges
- Conclusion

Objective

- Build abstractions from low-level system call traces generated using LTTng
  - E.g. mulitple disk blocks read requests, disk controller interrupts can be replaced by a simple ‘read file’
- Applications
  - Help users understand the behavioural aspects of a system to facilitate debugging, adding new features, etc.
  - Ensuring that subsequent versions of the same system evolve without new errors being introduced
  - Comparing instances of the same system in a redundant and diverse architecture for fault detection and isolation

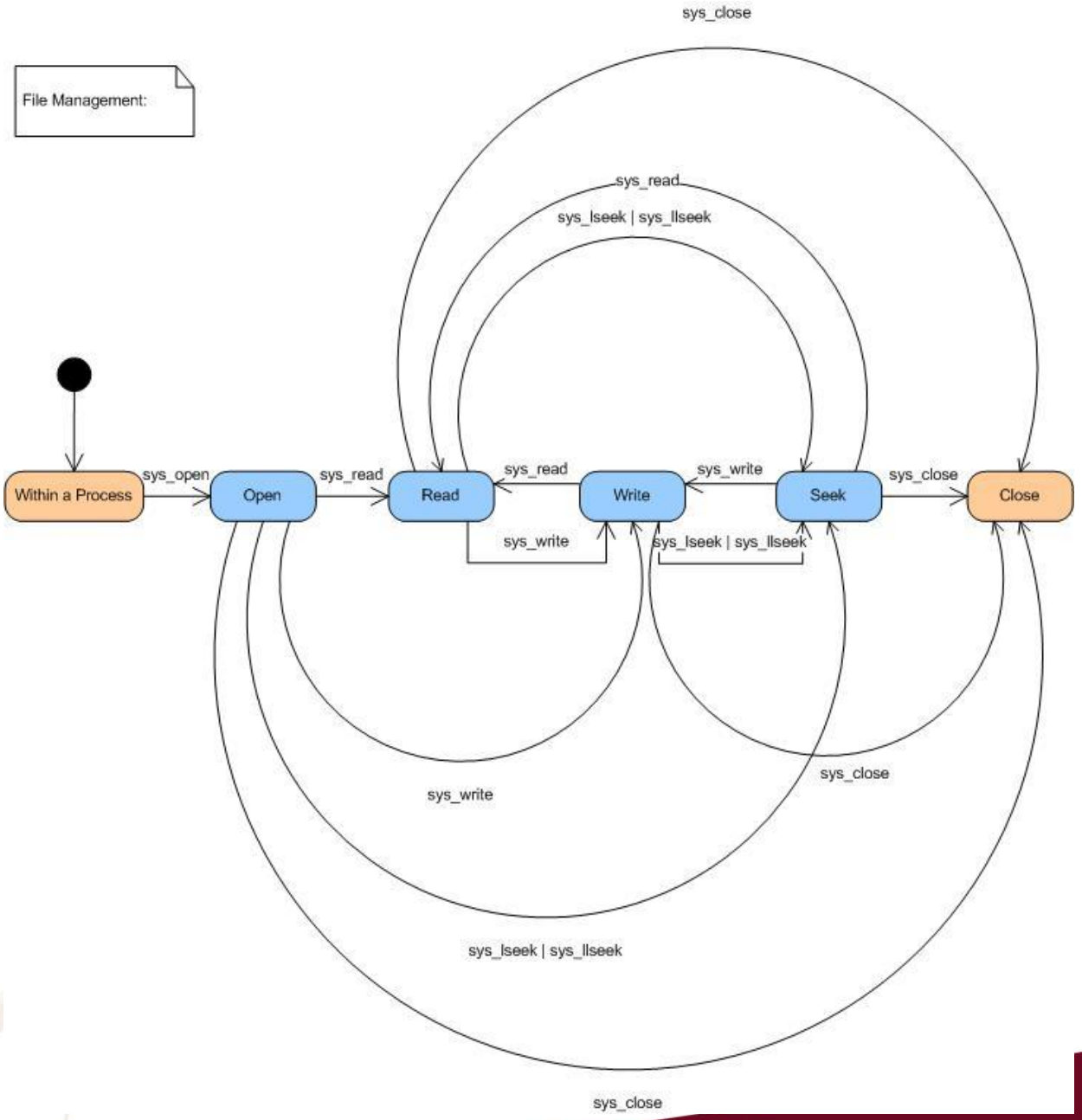
Approach


Trace Generated from Linux Kernel (LTTng events)
Trace Abstraction Algorithm Based on Pattern Matching and Filtering of Noise
High-Level Trace
Sample LTTng Traces
Expert Knowledge
Pattern Library
Linux Kernel Documentation

Pattern Library

- We built a pattern library that contains several patterns that represent key Linux kernel operations
  - File, socket and process management operations
- The patterns are modeled as state machines
  - States represent system modes (user_mode, syscall_mode, etc.)
  - Events consist of LTTng events

Example of a pattern



Filtering of Trace Noise

- We define noise in an LTTng trace as any event associated with memory management, page faults, and interrupts
  - Are dependent on a specific system architecture
  - Can occur anywhere in the trace and in any order
  - Are treated similarly to the way utilities have been treated in related work
- Associated events are treated as a set
  - i.e. order of occurrence of detailed events is ignored

Progress since the last meeting

- 30 more patterns have been defined (not all of them are implemented)
  - In total around 70 patterns have been formally defined
- Improvements have been made to the Linux Kernel Trace Abstraction Tool
- The development of a schema for defining patterns
- Additional case studies on large traces
- We started exploring VM and user space traces
- Thesis writing and defence

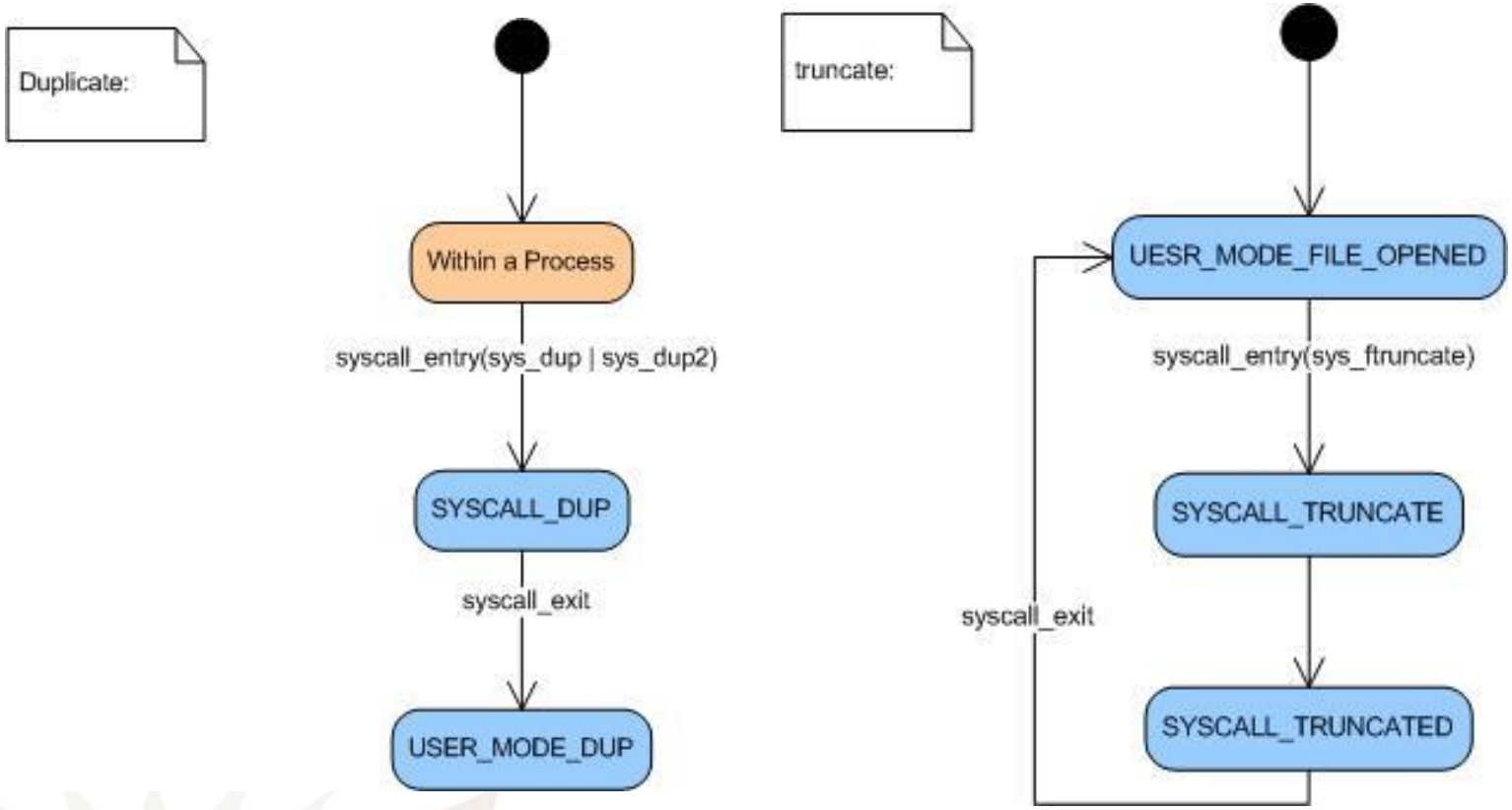
Catalogue of Patterns (updated)

- File Management Operations
  - Open, Read, Write, Seek, Close, Access, File Control, Stat, Read Link, File Duplicate, File Truncate, Device Control, and Poll
- TCP/UDP Socket Management
  - Create, Connect, Bind, Listen, Accept, Send, Receive, Close

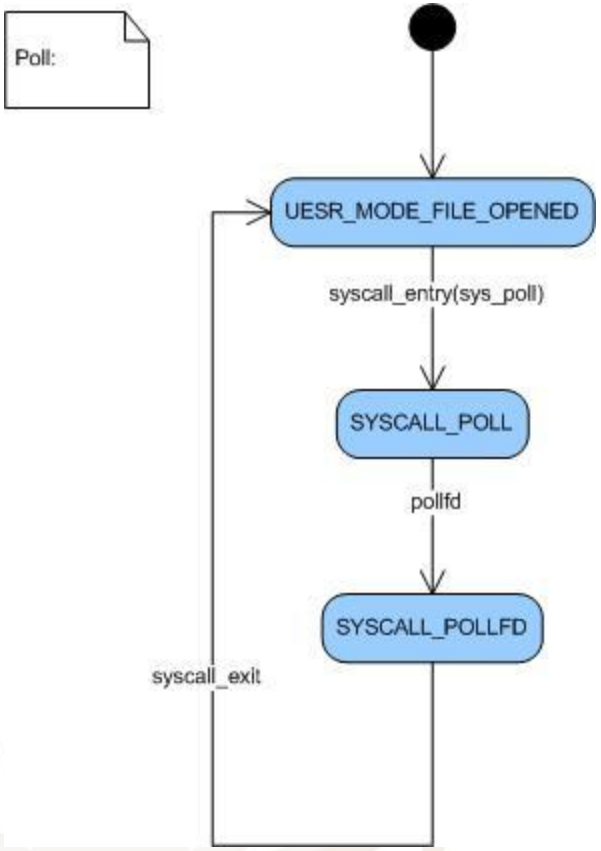
Catalogue of Patterns (cont.)

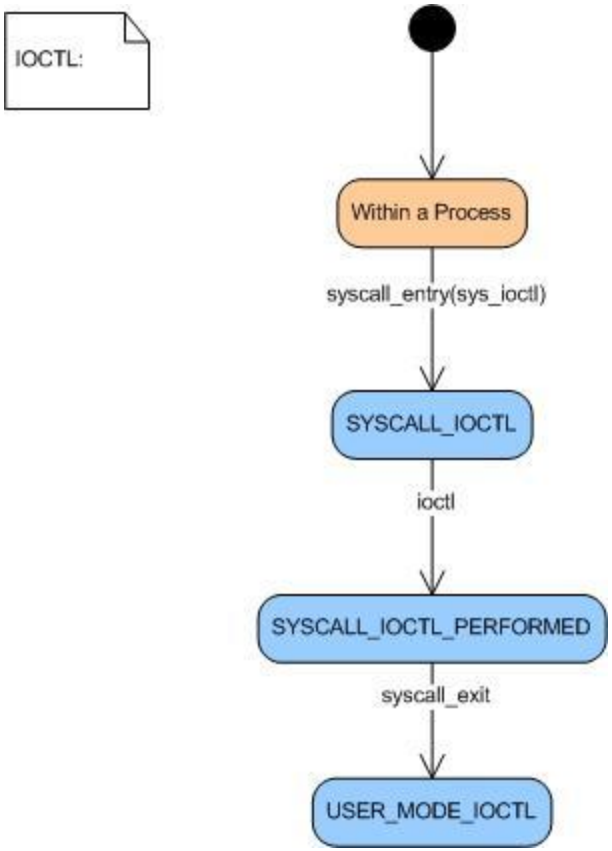
- Process Management:
  - Clone, Execute, Get Resource Limit, Get Time of Day, Exit, Unlink, Get User ID, Get Group ID, Get Process ID, Get Parent Process ID, Set Scheduling Parameters, Get Scheduling Parameters, Get Maximum Scheduling Algorithm Priority, Get Minimum Scheduling Algorithm Priority, Set Scheduling Policy and Parameters, Change Dir, Signal Return, Clock Get Time, Futex, Get Directory Entries, IPC, Get Memory Advice, Pipe, and Change Mode

Example of New Patterns



New Patterns (cont.)

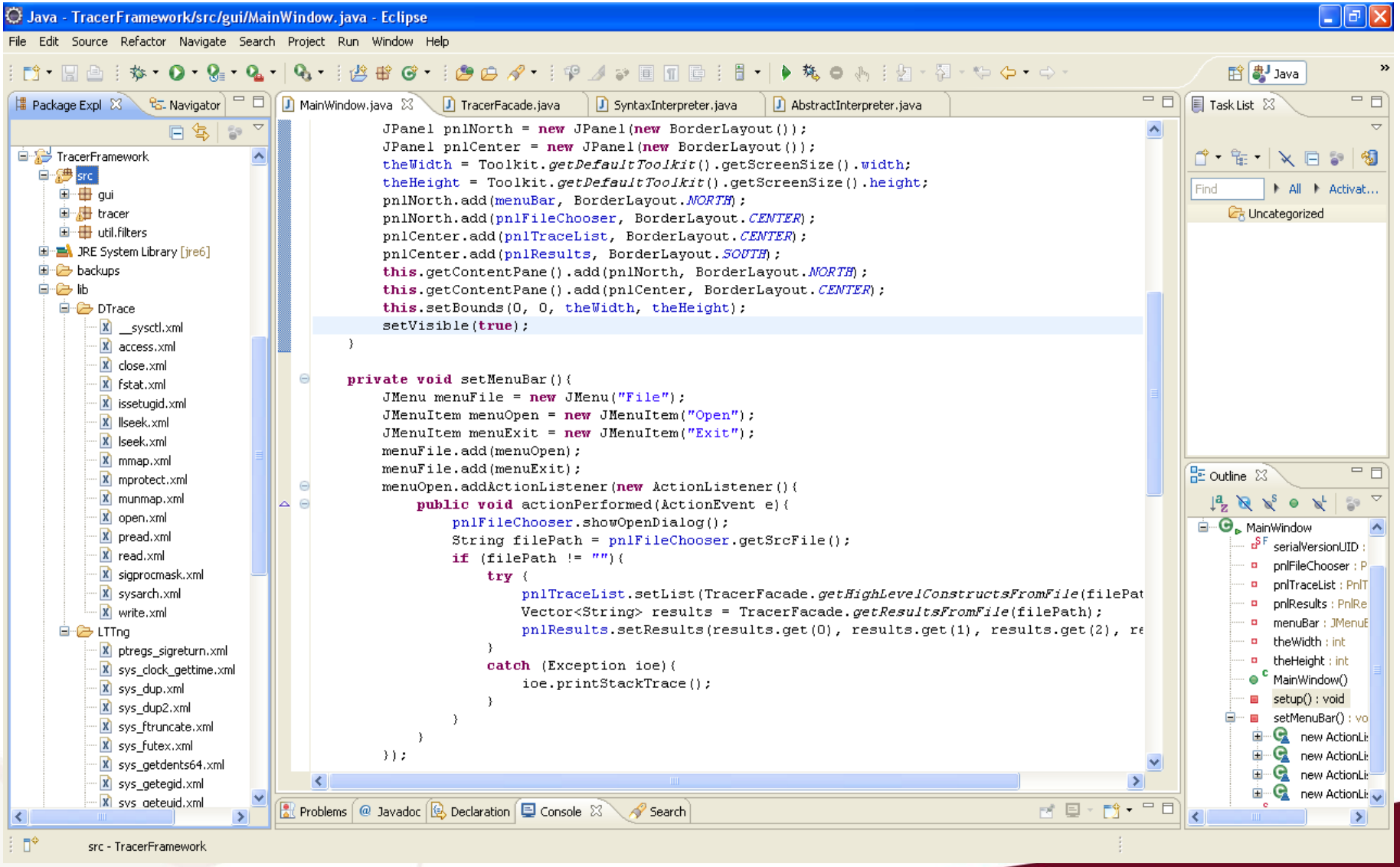




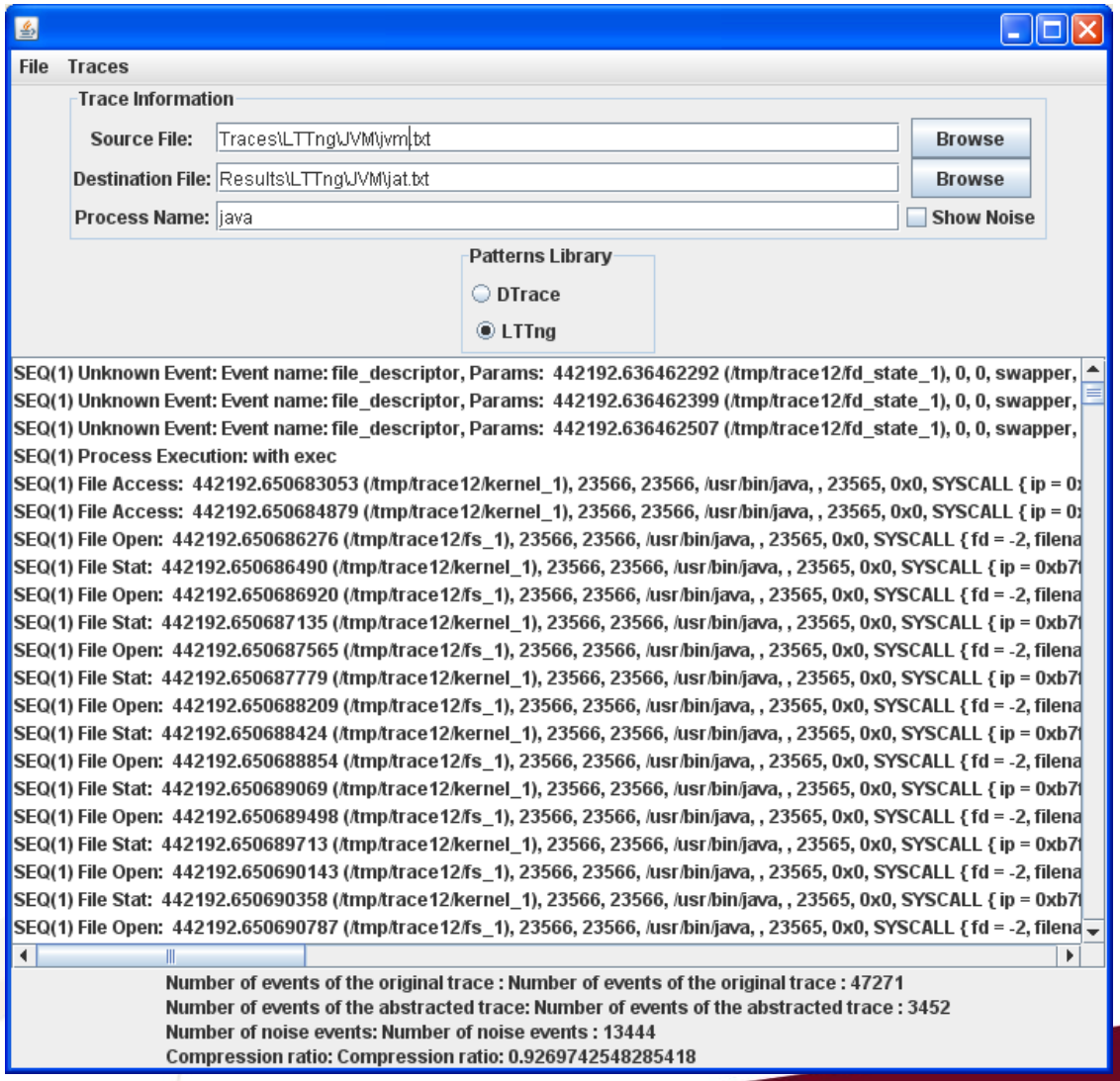
The Linux Kernel Trace Abstraction Tool

- The tool takes as input a trace generated from LTTng tracer
- It applies the abstraction process to the trace
- It outputs the trace in its abstracted format
- It is developed in Java and targeted to be integrated with the TMF Eclipse plugin

Snapshot


Java - TracerFramework/src/gui/MainWindow.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Expl
Navigator
TracerFramework
src
gui
tracer
util.filters
JRE System Library [jre6]
backups
lib
DTrace
__sysctl.xml
access.xml
close.xml
fstat.xml
issetugid.xml
llseek.xml
lseek.xml
mmap.xml
mprotect.xml
munmap.xml
open.xml
pread.xml
read.xml
sigprocmask.xml
sysarch.xml
write.xml
LTTng
ptregs_sigreturn.xml
sys_clock_gettime.xml
sys_dup.xml
sys_dup2.xml
sys_ftruncate.xml
sys_futex.xml
sys_getdents64.xml
sys_getegid.xml
sys_geteuid.xml
MainWindow.java
TracerFacade.java
SyntaxInterpreter.java
AbstractInterpreter.java
JPanel pnlNorth = new JPanel(new BorderLayout());
JPanel pnlCenter = new JPanel(new BorderLayout());
theWidth = Toolkit.getDefaultToolkit().getScreenSize().width;
theHeight = Toolkit.getDefaultToolkit().getScreenSize().height;
pnlNorth.add(menuBar, BorderLayout.NORTH);
pnlNorth.add(pnlFileChooser, BorderLayout.CENTER);
pnlCenter.add(pnlTraceList, BorderLayout.CENTER);
pnlCenter.add(pnlResults, BorderLayout.SOUTH);
this.getContentPane().add(pnlNorth, BorderLayout.NORTH);
this.getContentPane().add(pnlCenter, BorderLayout.CENTER);
this.setBounds(0, 0, theWidth, theHeight);
setVisible(true);
}
private void setMenuBar(){
JMenu menuFile = new JMenu("File");
JMenuItem menuOpen = new JMenuItem("Open");
JMenuItem menuExit = new JMenuItem("Exit");
menuFile.add(menuOpen);
menuFile.add(menuExit);
menuOpen.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
pnlFileChooser.showOpenDialog();
String filePath = pnlFileChooser.getSrcFile();
if (filePath != ""){
try {
pnlTraceList.setList(TracerFacade.getHighLevelConstructsFromFile(filePat
Vector<String> results = TracerFacade.getResultsFromFile(filePath);
pnlResults.setResults(results.get(0), results.get(1), results.get(2), re
}
catch (Exception ioe){
ioe.printStackTrace();
}
}
}
});
Problems
Javadoc
Declaration
Console
Search
Task List
Find
All
Activat...
Uncategorized
Outline
MainWindow
serialVersionUID
pnlFileChooser : P
pnlTraceList : PnlT
pnlResults : PnlRe
menuBar : JMenuB
theWidth : int
theHeight : int
MainWindow()
setup() : void
setMenuBar() : vo
new ActionLi
new ActionLi
new ActionLi
new ActionLi
src - TracerFramework

A Simple GUI


File
Traces
Trace Information
Source File:
Traces\LTTng\JVM\jvm.txt
Browse
Destination File:
Results\LTTng\JVM\jat.txt
Browse
Process Name:
java
Show Noise
Patterns Library
DTrace
LTTng
SEQ(1) Unknown Event: Event name: file_descriptor, Params: 442192.636462292 (/tmp/trace12/fd_state_1), 0, 0, swapper,
SEQ(1) Unknown Event: Event name: file_descriptor, Params: 442192.636462399 (/tmp/trace12/fd_state_1), 0, 0, swapper,
SEQ(1) Unknown Event: Event name: file_descriptor, Params: 442192.636462507 (/tmp/trace12/fd_state_1), 0, 0, swapper,
SEQ(1) Process Execution: with exec
SEQ(1) File Access: 442192.650683053 (/tmp/trace12/kernel_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { ip = 0x
SEQ(1) File Access: 442192.650684879 (/tmp/trace12/kernel_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { ip = 0x
SEQ(1) File Open: 442192.650686276 (/tmp/trace12/fs_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { fd = -2, filena
SEQ(1) File Stat: 442192.650686490 (/tmp/trace12/kernel_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { ip = 0xb7
SEQ(1) File Open: 442192.650686920 (/tmp/trace12/fs_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { fd = -2, filena
SEQ(1) File Stat: 442192.650687135 (/tmp/trace12/kernel_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { ip = 0xb7
SEQ(1) File Open: 442192.650687565 (/tmp/trace12/fs_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { fd = -2, filena
SEQ(1) File Stat: 442192.650687779 (/tmp/trace12/kernel_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { ip = 0xb7
SEQ(1) File Open: 442192.650688209 (/tmp/trace12/fs_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { fd = -2, filena
SEQ(1) File Stat: 442192.650688424 (/tmp/trace12/kernel_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { ip = 0xb7
SEQ(1) File Open: 442192.650688854 (/tmp/trace12/fs_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { fd = -2, filena
SEQ(1) File Stat: 442192.650689069 (/tmp/trace12/kernel_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { ip = 0xb7
SEQ(1) File Open: 442192.650689498 (/tmp/trace12/fs_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { fd = -2, filena
SEQ(1) File Stat: 442192.650689713 (/tmp/trace12/kernel_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { ip = 0xb7
SEQ(1) File Open: 442192.650690143 (/tmp/trace12/fs_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { fd = -2, filena
SEQ(1) File Stat: 442192.650690358 (/tmp/trace12/kernel_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { ip = 0xb7
SEQ(1) File Open: 442192.650690787 (/tmp/trace12/fs_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { fd = -2, filena
Number of events of the original trace : Number of events of the original trace : 47271
Number of events of the abstracted trace: Number of events of the abstracted trace : 3452
Number of noise events: Number of noise events : 13444
Compression ratio: Compression ratio: 0.9269742548285418

New Features

- The abstraction process operates on the whole trace instead of one process
- Link between the abstracted events and the corresponding lines of the original trace have been added
- Patterns are modeled as XML files which can be fed to the tool

New Features (cont.)

- Easy to add new patterns
- Easy to build higher level abstractions based on the current level
- Pattern library and the programming language are totally separated
- Patterns can be exchanged between different tools
- TMF integration – still in progress

XML Representation of Patterns

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pattern name="Sample Pattern" type="HighLevelSampleConstrcut"
    noise="false">
<event name="syscall_entry" syscall_name="sys_sample" order="1"
    prev_state="IGNORE">
      <current_state>SYSCALL_SAMPLE</current_state>  </event>
<event name="sample" order="2"
    prev_state="SYSCALL_SAMPLE">
      <current_state>SYSCALL_SAMPLED</current_state>
    </event>
<event name="syscall_exit" order="LAST"
    prev_state="SYSCALL_SAMPLED">
      <current_state>USER_MODE_SAMPLED</current_state>
    </event>
</pattern>
```

XML Representation of Patterns (cont.)

```
<?xml version="1.0" encoding="UTF-8"?>
<pattern name="Duplicate File Descriptor" type="HighLevelDupConstrcut"
    noise="false">
  <event name="syscall_entry" syscall_name="sys_dup" order="1"
    prev_state="IGNORE" current_state="SYSCALL_DUP">
  </event>
  <event name="syscall_exit" order="LAST" prev_state="SYSCALL_DUP"
    current_state="USER_MODE_DUP">
  </event>
</pattern>
```

Case Studies

- We applied our approach to large traces generated from the following systems
  - Java Virtual Machine
  - The Eclipse framework
  - Gedit
  - GIMP image editor
  - Firefox

Quantitative Analysis

Process	Initial Size	Size after Abstraction	Number of Noise Events	Compression Ratio
Eclipse	1226985	465886	94362	62%
GIMP	847575	243871	132343	71%
Firefox	646710	309926	41631	52%
Gedit	186167	100523	10830	46%
JVM	47271	3452	13444	93%

Qualitative Analysis

A snapshot of a C application that was traced by LTTng

```c
#include <stdio.h>

int main(void){
    FILE *fp;
    fp = fopen("output.txt", "w");
    fprintf(fp, "This is a test line\n");
    fprintf(fp, "This is another test line\n");
    fprintf(fp, "This is the last test line");
    fclose(fp);
    fp = fopen("output.txt", "r");
    int c = 0;
    while (c!=EOF) {
        c=fgetc(fp);
            printf("%c", c);
    }
    fclose(fp);
    return 0;
}
```

Qualitative Analysis

```
Abstracted trace after removing noise events:
=============================================
SEQ(1) Process Execution: with exec
SEQ(1) File Access:  442192.435311130 (/tmp/trace10/kernel_1), 22438, 22438, ./Files, , 29184, 0x0, S
SEQ(1) File Access:  442192.435313279 (/tmp/trace10/kernel_1), 22438, 22438, ./Files, , 29184, 0x0, S
SEQ(1) File Open:  442192.435315212 (/tmp/trace10/fs_1), 22438, 22438, ./Files, , 29184, 0x0, SYSCALL
SEQ(1) File Stat:  442192.435315427 (/tmp/trace10/kernel_1), 22438, 22438, ./Files, , 29184, 0x0, SYS
SEQ(1) File or Socket Close:  442192.435316609 (/tmp/trace10/fs_1), 22438, 22438, ./Files, , 29184, 0
SEQ(1) File Access:  442192.435317791 (/tmp/trace10/kernel_1), 22438, 22438, ./Files, , 29184, 0x0, S
SEQ(1) File Open:  442192.435321551 (/tmp/trace10/fs_1), 22438, 22438, ./Files, , 29184, 0x0, SYSCALL
SEQ(1) File Read:  fd = 3 }
SEQ(1) File Stat:  442192.435323162 (/tmp/trace10/kernel_1), 22438, 22438, ./Files, , 29184, 0x0, SYS
SEQ(1) File or Socket Close:  442192.435328963 (/tmp/trace10/fs_1), 22438, 22438, ./Files, , 29184, 0x0, SYSCALL { fd = 3 }
SEQ(1) File Open:  442192.435348299 (/tmp/trace10/fs_1), 22438, 22438, ./Files, , 29184, 0x0, SYSCALL
SEQ(1) File Stat:  442192.435349373 (/tmp/trace10/kernel_1), 22438, 22438, ./Files, , 29184, 0x0, SYS
SEQ(1) Process Schedule:  442192.435348192 (/tmp/trace10/kernel_1), 22438, 22438, ./Files, , 29184, 0
SEQ(1) Unknown Event: Event name: add_to_page_cache, Params:  442192.435351200 (/tmp/trace10/mm_1), 2
SEQ(1) File Write:  fd = 3 }
SEQ(1) File or Socket Close:  442192.435351629 (/tmp/trace10/fs_1), 22438, 22438, ./Files, , 29184, 0
SEQ(1) File Open:  442192.435352596 (/tmp/trace10/fs_1), 22438, 22438, ./Files, , 29184, 0x0, SYSCALL
SEQ(1) File Stat:  442192.435353026 (/tmp/trace10/kernel_1), 22438, 22438, ./Files, , 29184, 0x0, SYS
SEQ(1) File Read:  fd = 3 }
SEQ(1) File Stat:  442192.435354315 (/tmp/trace10/kernel_1), 22438, 22438, ./Files, , 29184, 0x0, SYS
SEQ(2) File Write:  fd = 1 }
SEQ(1) File Read:  fd = 3 }
SEQ(1) File or Socket Close:  442192.435357860 (/tmp/trace10/fs_1), 22438, 22438, ./Files, , 29184, 0
SEQ(1) File Write:  fd = 1 }
SEQ(1) Process Exit: Process Exit:  442192.435366024 (/tmp/trace10/kernel_1), 22438, 22438, ./Files, , 29184, 0x0, SYSCALL { pid = 22438 }, Send Signal:  442192.435366669
Results:
========
Number of events of the original trace : 365
Number of events of the abstracted trace : 26
Number of noise events : 123
Compression ratio: 0.9287671232876712
```

Corresponds to process execution

```
fp = fopen("output.txt", "w");
fprintf(fp, "This is a test line\n");
fprintf(fp, "This is another test line\n");
fprintf(fp, "This is the last test line");
fclose(fp);
```

```
fp = fopen("output.txt", "r");
while (c!=EOF) {
  c=fgetc(fp);
  printf("%c", c);
}
fclose(fp);
```

Exploring VM and Userspace traces

- Experimenting different kinds of traces generated from LTTng
  - Virtual Machine Traces
  - Userspace Traces (UST)
- Investigating the abstraction of traces into higher-levels

Virtual Machine Traces

- Studied several sample traces generated while running KVM (provided by Julien Desfossez)
- KVM events have similar patterns as system call events
- KVM patterns need to be defined and added to the Pattern Library through the defined XML schema and fed to the abstraction tool

Improved Abstraction Based on Recurrent Patterns

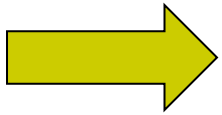
- High-level events could appear in the form of patterns that occur in a non-contiguous way
- As an example, the following events:

  File Open, File Read, File Close, Socket Create, Socket Bind, Socket Listen, Socket Accept, Socket Send, Socket Close
- Could correspond to a pattern that appears in multiple places in the trace
- Such patterns can be detected and replaced with a higher-level representation

Improved Abstraction Based on Recurrent Patterns (cont.)

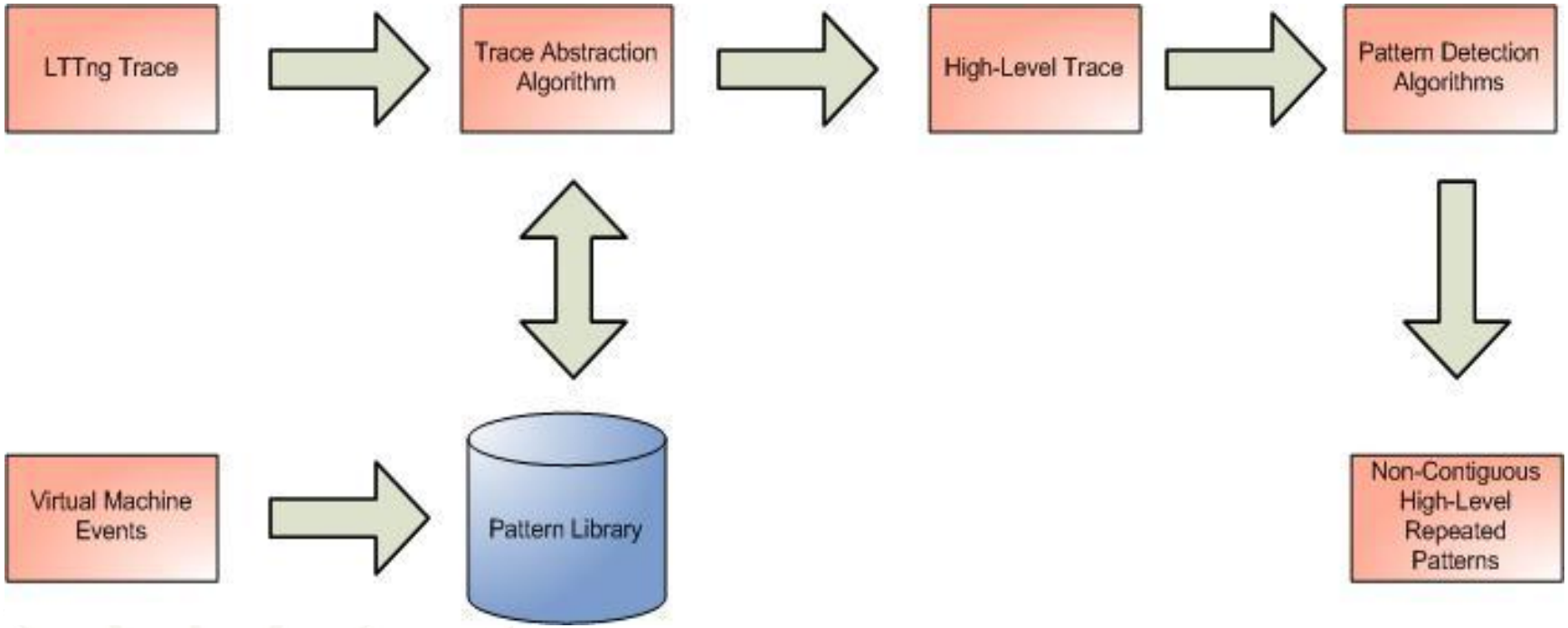
Example:

1- Process Execute
2- File Open
3- File Read
4- File Close
5- Get Time of Day
6- Read Link
7- Unlink
8- File Open
9- File Read
10- File Close
11- Process Exit



1- Process Execute
2- FM[2,8] = {File Open, File Read, File Close}
5- Get Time of Day
6- Read Link
7- Unlink
11- Process Exit

Possible techniques for detecting Recurrent Patterns



Possible techniques for detecting Recurrent Patterns (cont.)

- Various techniques could be explored for the detection of recurrent patterns:
  - String matching techniques (maximal pairs)
  - N-gram extraction algorithms
  - Suffix trees
  - Etc.

User Space Traces

- We experimented with sample user space traces provided by David Goulet
- Anything could be traced in a user-space application
- The flow of an application is monitored by tracing entry-exit points of that application's routines (methods, functions, or procedures)

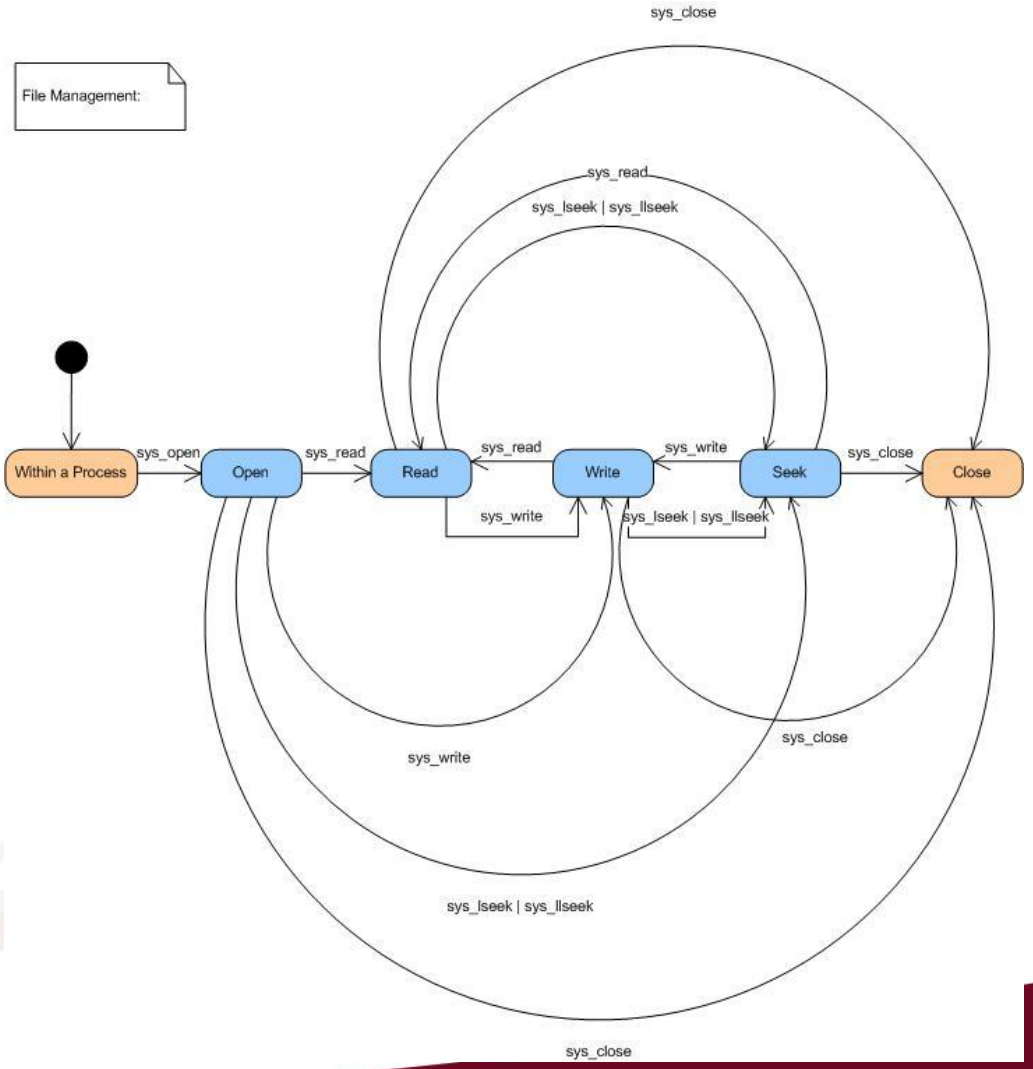
Abstraction of User Space Traces

- Many techniques have been developed to abstract routine call traces
  - Detecting and removing low-level implementation details
  - Detecting sequences of events
  - Transforming the trace into a Directed Acyclic Graph
  - Grouping of events based on the nesting level
  - Sampling
- These techniques need to be experimented with in the context of this research

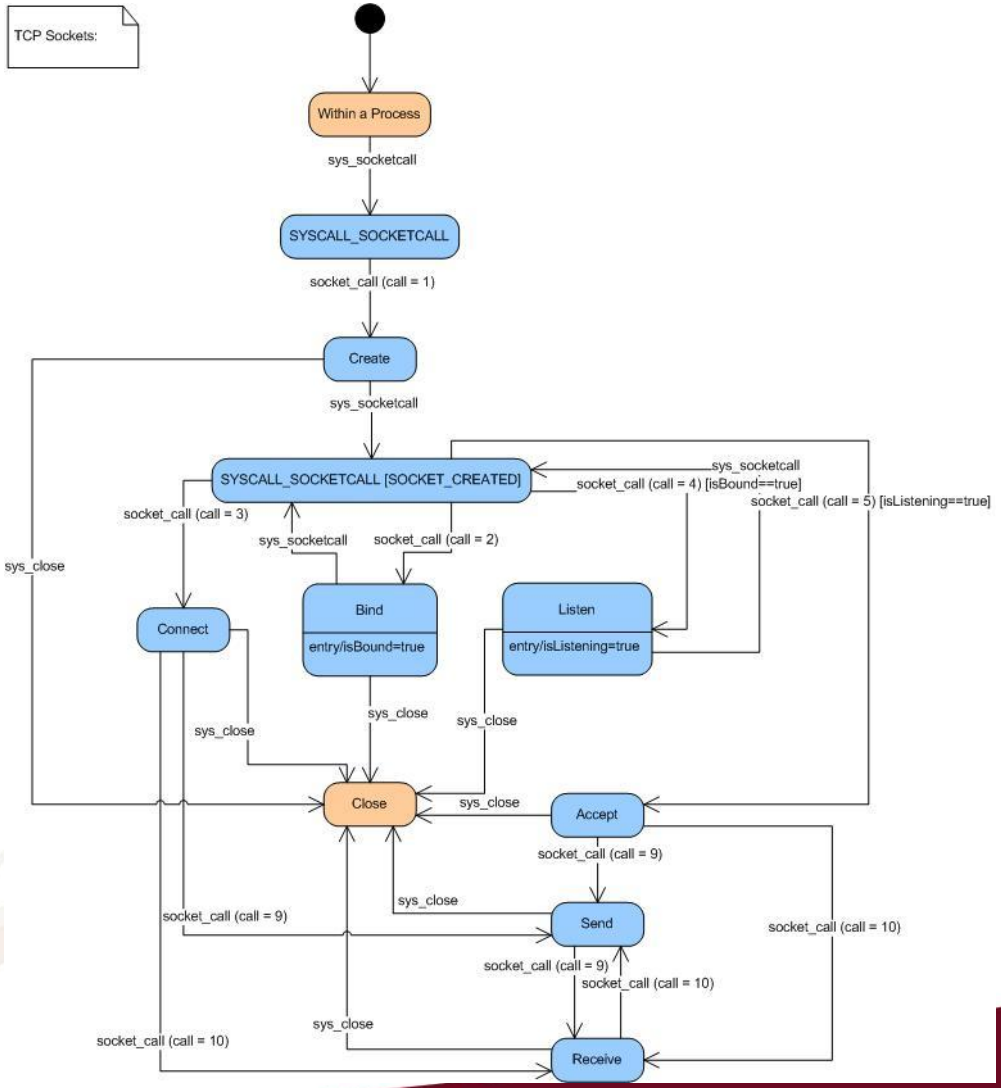
Higher-Level Abstraction

- Defining how high in the abstraction layers we should go
- Investigating the benefits of building higher-level patterns based on the patterns defined in the Pattern Library
- Solving the problem of interleaved events belonging to different higher-level patterns

Higher-Level Abstraction (cont.)



Higher-Level Abstraction (cont.)



Higher-Level Abstraction (cont.)

- The following abstracted trace lines:

  File Open: 442192.435321551 (/tmp/trace10/fs_1), 22438, 22438, ./Files, , 29184, 0x0, SYSCALL { fd = 3, filename = "/lib/tls/i686/cmov/libc.so.6" }

  File Read: fd = 3 }

  File Stat: 442192.435323162 (/tmp/trace10/kernel_1), 22438, 22438, ./Files, , 29184, 0x0, SYSCALL { ip = 0xb7fc1a6e, syscall_id = 197 [sys_fstat64+0x0/0x30] }

  File or Socket Close: 442192.435328963 (/tmp/trace10/fs_1), 22438, 22438, ./Files, , 29184, 0x0, SYSCALL { fd = 3 }

  Socket Create: 442192.652063137 (/tmp/trace12/net_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { family = 1, type = 1, protocol = 0, sock = 0xd563d340, ret = 3 }

  Socket Connect: 442192.652064103 (/tmp/trace12/net_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { fd = 3, uservaddr = 0xbf8dbb0a, addrlen = 110, ret = -2 }

  File or Socket Close: 442192.652064426 (/tmp/trace12/fs_1), 23566, 23566, /usr/bin/java, , 23565, 0x0, SYSCALL { fd = 3 }

- Could be replaced by:

  File Management (File Open, File Read, File Stat, File Close)

  Socket Management (Socket Create, Socket Connect, Socket Close)

Remaining Challenges

- Continuous improvement of the pattern library
  - Defining additional patterns
  - Dealing with new LTTng events
- Improving the algorithm in terms of performance
- Implementing different pattern detection algorithms over the abstracted traces
- Integration with the TMF plugin

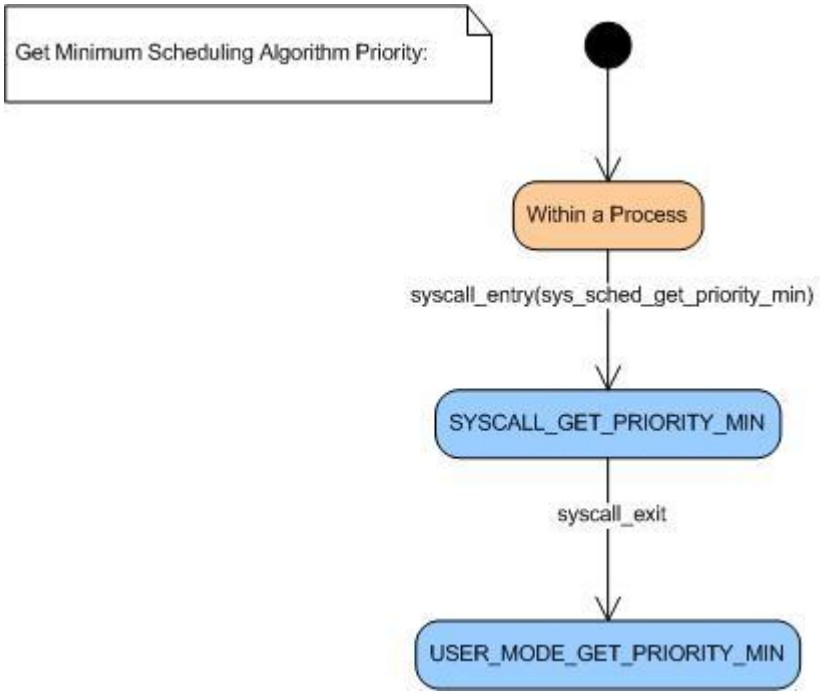
Conclusion

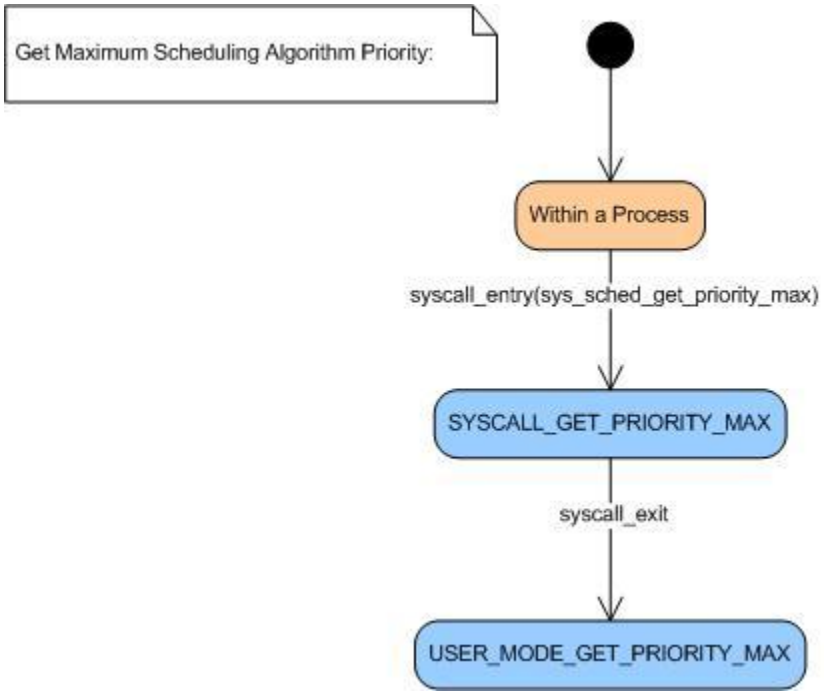
- We introduced techniques to abstract execution traces resulting from the Linux kernel
- Our approach is based on building a pattern library that consists of patterns of the most common operations in Linux
- We also defined noise patterns that result from memory management operations and page faults
- We introduced an algorithm to abstract the system call traces by using the pattern library
- We applied our techniques to traces generated from several processes

Thank You!
Questions?

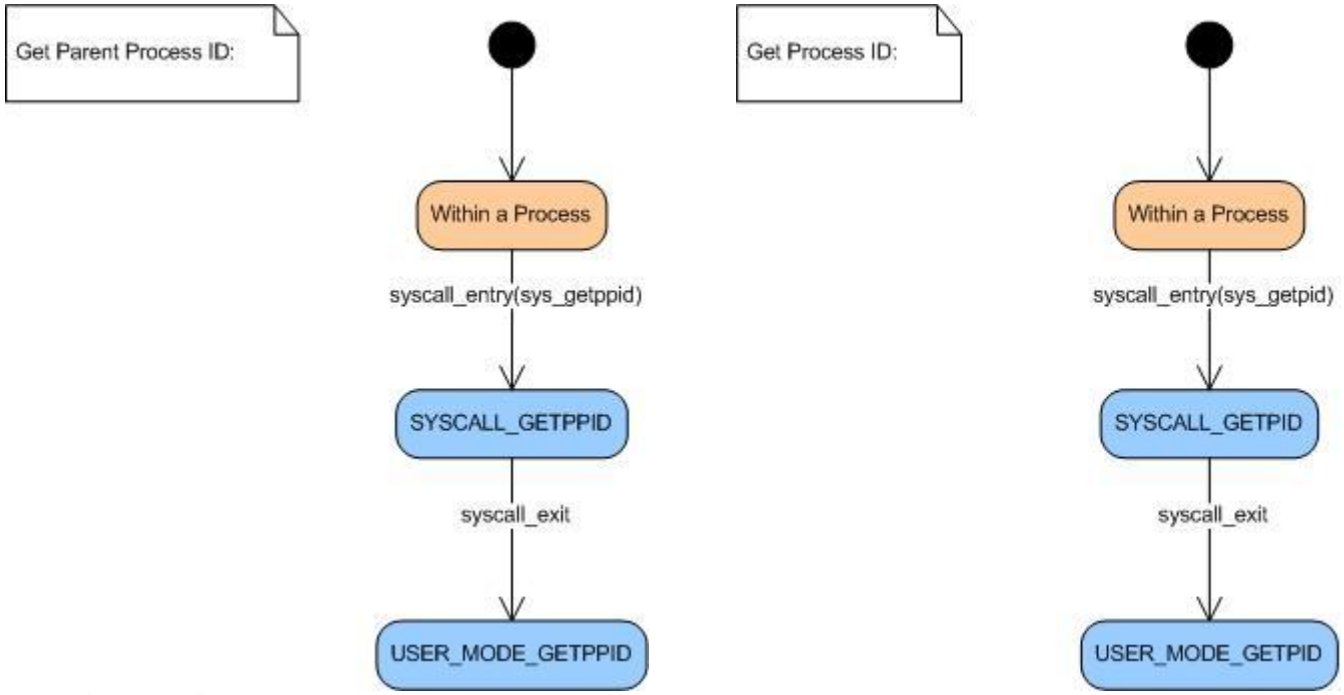
New Patterns

New Patterns (cont.)

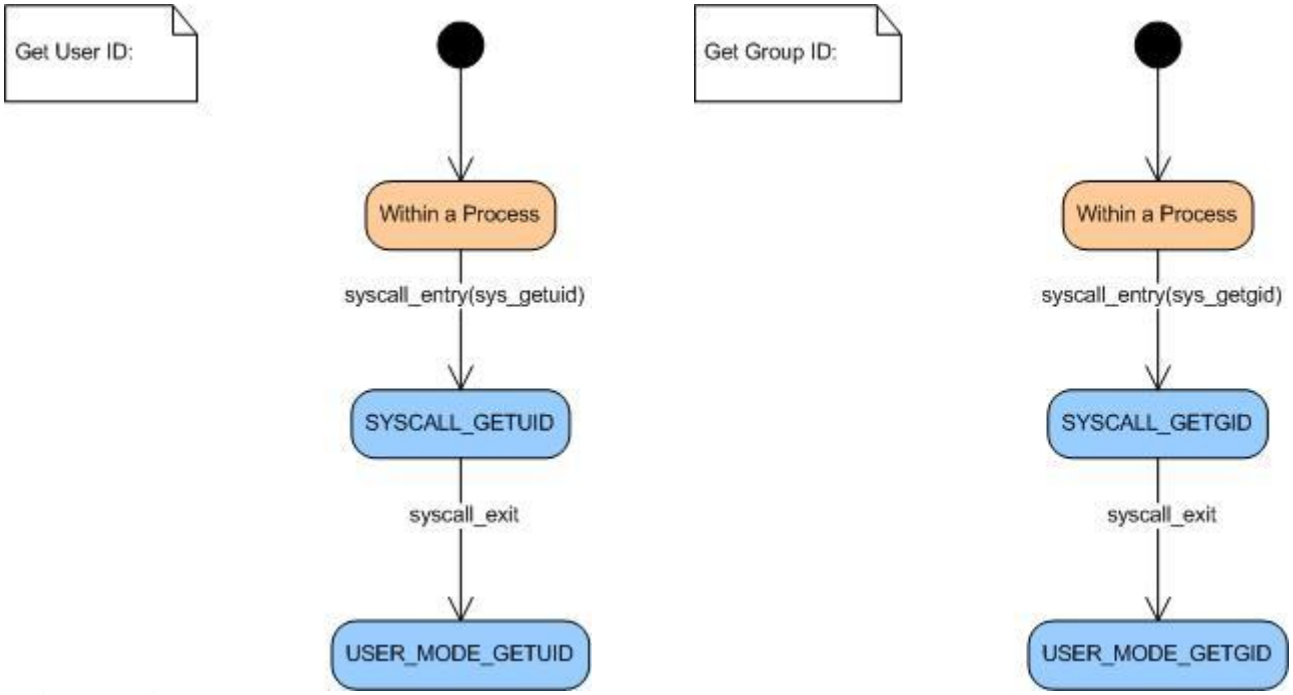




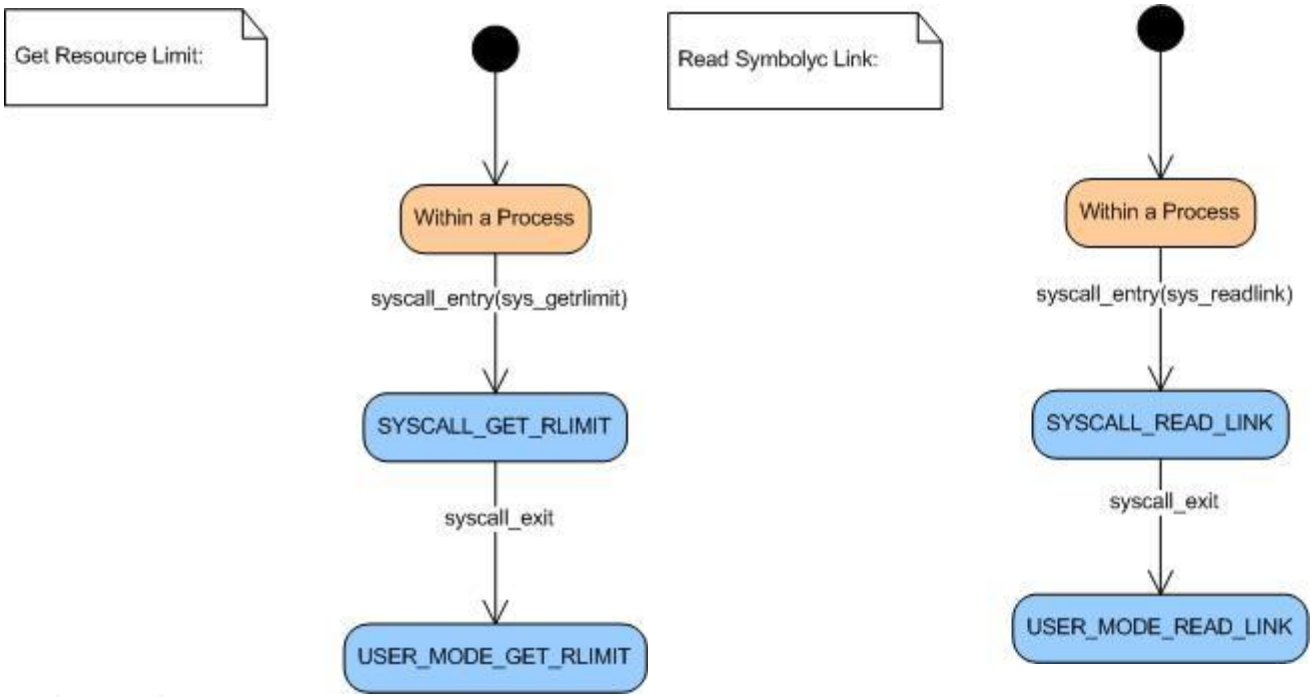
New Patterns (cont.)



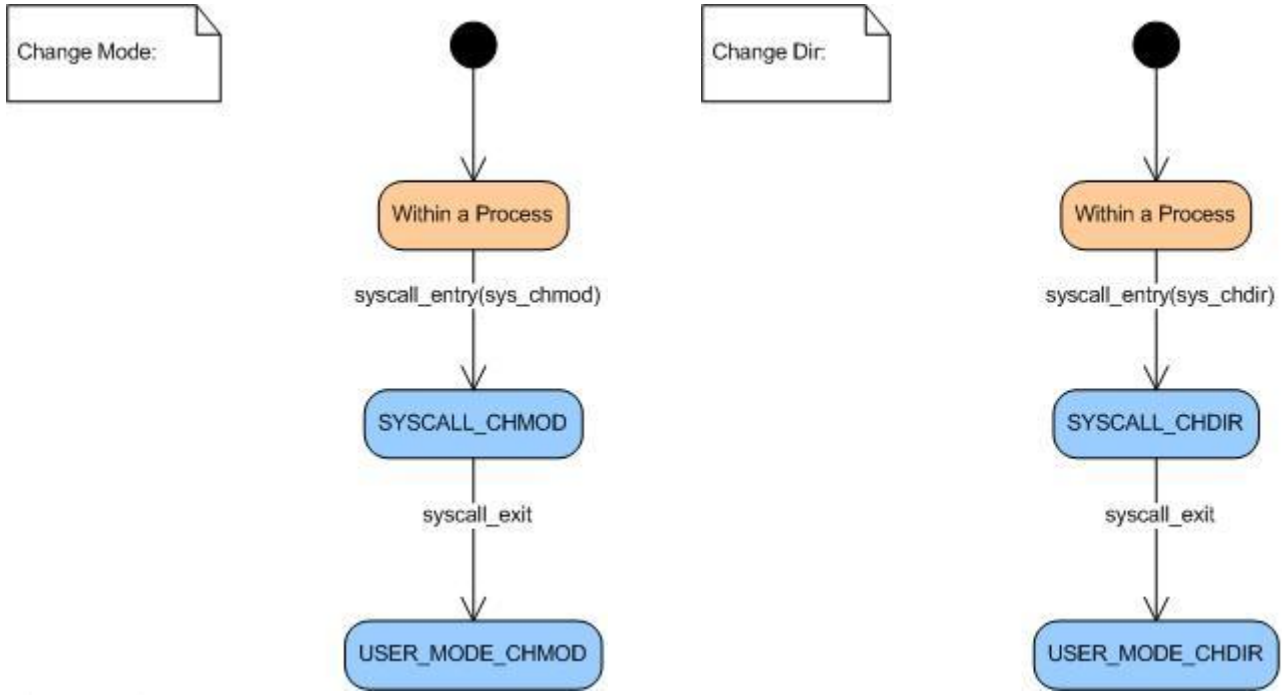
New Patterns (cont.)



New Patterns (cont.)



New Patterns (cont.)



New Patterns (cont.)