Characterization of COTS Microkernel-based Systems using MAFALDA

Jean-Charles Fabre

LAAS-CNRS
Toulouse, France

IFIP Working Group 10.4 Meeting — Paraty, Brazil — March 1-3, 2001
Problem statement

• **Building executive supports for dependable systems, two options:**
  - Development from scratch is complex & expensive
  - Use of commercial components is questionable

• **Main tendency for embedded systems**
  - Use of COTS componentized microkernels
  - Define a specific instance for the application
  - System development: two options
Outline

• The objectives of MAFALDA

• MAFALDA in action

• Experimental results

• Lessons learnt
Objectives of MAFALDA

• **Characterization by SWIFI**
  
  (S/W Implemented Fault Injection)
  
  – Identification of failure modes
  – Evaluation of error detection coverage
  – Identification of propagation channels
  – Assessment of interface robustness

• **Wrapping framework**
  
  – Definition of formal wrappers
  – Definition of a reflective implementation framework
  – Application to both white-box & black-box candidates

• **Evaluation of the wrapped microkernel instance**
Failure modes

Application Corruption

Internal Corruption

Error

Microkernel

Application / middleware

Propagation

API

Error

System hang

Internal detection

Error Status

Exceptions

Application failure

Erroneous results

Application hang

Application Oracle

Wp_i

Wp_j

μkCi

μkCj

severity
Fault injection experiment

Console
[c042d32e:c042d32e:506:1979._oSemUnblock:TEXT]\ 3e XOR 1 \(\rightarrow\) 3f

Time trigger has occurred

The fault has been activated (breakpoint0)

Fault removed [Single Step Called]

Outputs
th2_exit(0) SENDER_STEP3_OK RECEIVER_STEP3_OK th1_create(th2) th2_threadScheduler(prio--
MEM_BLOCK_1__
SENDRER_STEP1_OK MEM_BLOCK_2_OK th1_doWork(A) th1_delay(20) th3_doWork(B) th3_semV(s1)
th3_semP(s2) th4_doWork(A) th4_semP(1) th2_doWork(B) RECEIVER_STEP1_OK SENDER_STEP2_OK
RECEIVER_STEP2_OK th1_threadScheduler(prio--
MEM_BLOCK_2_OK
th2_delay(30) SENDER_STEP3_OK RECEIVER_STEP3_OK th4_doV
th4_semV(s2) th4_semV(s1) th4_threadDelay(0) th1_doWork(C) th2_doWork(D) th2_exit(0)
MEM_BLOCK_3_OK

SCH
th2_doWork(B)
th1_threadScheduler(prio--
th2_delay(30)
th1_doWork(C)
th2_exit(0)

SVN
th4_doWork(A)

MEM
MEM_BLOCK_2_OK
MEM_BLOCK_3_OK
MEM_BLOCK_1_OK
MEM_BLOCK_2_OK
MEM_BLOCK_3_OK

COM
SENDERR_STEP3_OK RECEIVER_STEP3_OK
SENDERR_STEP3_OK RECEIVER_STEP3_OK
SENDERR_STEP1_OK RECEIVER_STEP1_OK
SENDERR_STEP2_OK
Sample of measures
Campaigns

• **Microkernels**
  - Candidates:
    - Chorus Classix r3.1 (Kernel API),
    - Lynx OS v 3.0.1 (Kernel/Posix API)
  - Components:
    - Synchronisation (*semaphores*)
    - Memory (*protected regions*)
    - Communication (*message passing*)
    - Scheduling (*preemptive FIFO*)

• **Campaign parameters**
  - Same workload mapped on two different APIs
  - Running on the same Pentium-based platform
  - Between 1000 to 3000 experiments for each component
  - All components targeted
  - Both microkernel and parameters fault injection experiments
Chorus vs. LynxOS

Code segment fault injection

Synchronisation

Kernel

Scheduling
Chorus vs. LynxOS

Code segment fault injection

Communication

Kernel

Memory

Chorus Classix r3.1
LynxOS r 3.0.1
Chorus vs. LynxOS

Parameter fault injection

Synchronisation

API

Kernel

Memory

Chorus Classix r3.1
LynxOS r 3.0.1
Chorus vs. LynxOS

Parameter fault injection

Similar behavior, except that a system call with given parameters can hang the application or even the kernel.
Chorus vs. LynxOS

Parameter fault injection

Communication

API

Kernel

- int portMigrate (options, srcactorcap, portli, dstactorcap, seqnum)
- int portDelete (actorcap, portli)

KnCap

Chorus Classix r3.1

LynxOS r 3.0.1
Running mode impact

Downloading application code into kernel space
(Synchronisation workload)

User mode

Kernel mode

Code segment fault injection
experiments carried out on Chorus Classix r 3.1
Detailed system call analysis

<table>
<thead>
<tr>
<th>Kernel call</th>
<th>GetPriority</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Parameter number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>int which</td>
<td>int pid</td>
</tr>
<tr>
<td>Activated faults</td>
<td>214</td>
<td>213</td>
</tr>
<tr>
<td>Application failure</td>
<td>0</td>
<td>0.0%</td>
</tr>
<tr>
<td>Application hang</td>
<td>0</td>
<td>0.0%</td>
</tr>
<tr>
<td>Exception</td>
<td>0</td>
<td>0.0%</td>
</tr>
<tr>
<td>Error status</td>
<td>214</td>
<td>100%</td>
</tr>
<tr>
<td>No observation</td>
<td>0</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel call</th>
<th>SetPriority</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>int which</td>
<td>int pid</td>
</tr>
<tr>
<td>Activated faults</td>
<td>138</td>
<td>180</td>
</tr>
<tr>
<td>Application failure</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Application hang</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Exception</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Error status</td>
<td>138</td>
<td>144</td>
</tr>
<tr>
<td>No observation</td>
<td>0</td>
<td>29</td>
</tr>
</tbody>
</table>

- Most of individual cases can be analysed – Examples:
  - Priority out-of-bounds (*Error status*)
  - Invalid priority (*Application failure*)

- Possible conclusions:
  - The corrupted input value can be detected (assertion missing)
  - The corrupted input is valid for the kernel and cannot be checked (to be checked at the application/middleware level)

- The input space is randomly corrupted (sometimes all bits)
Some Lessons learnt

• **Interpretation of results**
  - One campaign: a microkernel instance + an activation profile
  - Variability of results: stand-alone vs. Posix-based version
    reactive vs. static application

• **Raw data analysis**
  - Analysis of logged data ➔ precise analysis of faulty situations
  - User-defined semantics of the failure modes

• **Integrator’s vs. supplier’s viewpoint**
  - Integrator: weaknesses revealed ➔ ED mechanisms (wrappers)
  - Supplier: bugs not yet revealed ➔ product improvement

• **Target system evolution**
  - A slightly new instance ➔ new campaign needed
  - Is the new release/version acceptable?