Can Minix be "tuned" in order to satisfy hard real-time constraint without loosing its soul

Emery Kouassi Assogba & Marc Lobelle

emery.assogba@{uclouvain.be, uac.bj}
marc.lobelle@{uclouvain.be, uac.bj}

February the 1\textsuperscript{st}, 2016
Plan

Part 1

● About the title
● Test context
● Dhrystone test
● How the System Benchmarks Index Score evolves with the tick/second

Part 2

● Our real-time problem: the blended hardening technique (BHT)
About the title

Real-Time systems need (among other things)

- Predictability of behavior
- A real-time scheduler
- A short time quantum so that the scheduler is called often enough
- A fine grain clock

Need not

- Being fast, just fast enough for the job

Traditionally, MINIX has a long time quantum and a coarse clock.

How does it behave if we need to reduce both?
Test context

• Computer
  ➢ CPU Intel core 2 Quad 2.6 GHz
  ➢ RAM : DDR3 4 GB
Test context (2)

- BYTE UNIX Benchmarks (Version 5.1.2)
  - 1 CPU in system; running 1 parallel copy of tests
  - OS: Minix : 3.2.1
- Test result
  - One run for each result
  - System Benchmarks Index Score (Partial Only)
Dhrystone test

Dhrystone is a synthetic computing benchmark program developed in 1984 by Reinhold P. Weicker

- intended to be representative of system (integer) programming [wikipedia].
Evolution of the Index Score with the tick/second

- Moderate penalty with short ticks
- Little change with quantum values
Evolution of the Index Score with the tick/second

**Zoom on variations with quantum size**

Note that each figure is to be compared to the right part of the next
Evolution of the Index Score with the tick/second: with instrumentation (verifying dirty bit)

- **Aim of instrumentation:** measurements of the working set size (number of pages modified during a quantum) of each process

- **At end of each quantum of a process:**
  - find pages with DIRTY_BIT set in process page table
  - modified pages numbers are stored in a list (*current_ws_list*).
  - Two others lists are maintained:
    - the “previous working set” list (*prev_ws_list*) is the current working set at end of the previous quantum
    - the “previous previous working set list” (*prev_prev_ws_list*) is the working set at end of the ante-previous quantum.
    - quantum 0 is the first quantum of the process.
Evolution of the Index Score with the tick/second: with instrumentation (verifying dirty bit) (2)

- The following numbers of pages are measured
  - **#modified**: #pages present in current_ws_list, i.e. the working set size
  - **#new_in**: #pages present in current_ws_list but are not present in the prev_ws_list
  - **#new_out**: #pages present in the prev_ws_list but not present in the current_ws_list
  - **#def_out**: #pages present in the prev_prev_ws_list but neither present in current_ws_list, nor in the prev_ws_list: definitively out
  - **#in_out**: #pages present in the current_ws_list but not present in the prev_ws_list and are present in the prev_prev_ws_list: returning pages
Evolution of the Index Score with the tick/second: with instrumentation (verifying dirty bit)(3)

Long traditional tick

Instrumented Minix

- 60ticks/second: tick≈17ms
- 60ticks/second: tick=17ms(instrumented)
Evolution of the Index Score with the tick/second: with instrumentation (verifying dirty bit)(4)

Zoom on variations with quantum size

Instrumentation cost is
- 7% for smallest quantum size
- 1.56 % for highest quantum size
Evolution of the Index Score with the tick/second: with instrumentation (verifying dirty bit) (5)

**Short tick**

Drystone Minix with instrumented Minix

<table>
<thead>
<tr>
<th>Quantum size in µs</th>
<th>Number of loop for 2min 20 second run</th>
</tr>
</thead>
<tbody>
<tr>
<td>390.63</td>
<td>800</td>
</tr>
<tr>
<td>781.25</td>
<td>800</td>
</tr>
<tr>
<td>1562.5</td>
<td>800</td>
</tr>
<tr>
<td>3125</td>
<td>800</td>
</tr>
<tr>
<td>6250</td>
<td>800</td>
</tr>
<tr>
<td>12500</td>
<td>800</td>
</tr>
<tr>
<td>25000</td>
<td>800</td>
</tr>
<tr>
<td>50000</td>
<td>800</td>
</tr>
<tr>
<td>100000</td>
<td>800</td>
</tr>
<tr>
<td>200000</td>
<td>800</td>
</tr>
</tbody>
</table>

- 12800 ticks/second: tick≈78µs
- 12800 ticks/second: tick≈78µs (instrumented)
Evolution of the Index Score with the tick/second: with instrumentation (verifying dirty bit)(6)

Zoom on variation with quantum size

- Why does the system crash for smaller quantum size?
  - The presence of an ATI graphic card caused a lot of heat
  - The heat caused address translation errors in the MMU
- Removing the ATI graphic card and ventilating the mother board solved the problem
Evolution of the Index Score with the tick/second: with instrumentation (verifying dirty bit)

**Short tick when the problem was solved**
Evolution of the Index Score with the tick/second: with instrumentation (verifying dirty bit) (8)

Zoom on variation with quantum size

Instrumentation cost is
- 6.7% for smallest quantum size
- 0.8 % for highest quantum size
Evolution of the Index Score with the tick/second: with instrumentation (pseudo copy-on-write)

• At end of process quantum 0 (the first quantum):
  • all the process memory space is set to read-only.
    – so that the kernel is warned when the process is going to modify any of its pages
  • current_ws_list is empty

• During each quantum:
  • the current working set (current_ws_list) of the process is increased when a page fault occurs; pages are then set to R/W.

• At the end of each quantum:
  • the modified pages are set back to read only.
  • current_ws_list is reset to empty
Evolution of the Index Score with the tick/second: with instrumentation (pseudo copy-on-write)(2)

Long traditional tick

- 60ticks/second: tick=17ms
- 60ticks/second: tick=17ms(instrumented)
- 60ticks/second: tick=17ms(instrumented with pseudo cow)
The frequent copy-on-write cost is:

- 43.28% for smallest quantum size
- 21.42% for highest quantum size

Not so good!!! Let's improve the algorithm.
Evolution of the Index Score with the tick/second: with instrumentation (pseudo copy-on-write) (4)

**Short tick**

![Graph showing evolution of index score with tick/second]

- 12800 ticks/second: tick≈78µs
- 12800 ticks/second: tick≈78µs (instrumented)
- 12800 ticks/second: tick≈78µs (instrumented with pseudo cow)
Evolution of the Index Score by the tick/second: with instrumentation (pseudo copy-on-write)(5)

The frequent copy-on-write cost is:

- 8.56% for smallest quantum size
- 2.3% for highest quantum size

Not so good!!! let's improve the algorithm.
Evolution of the Index Score with the tick/second: with instrumentation (pseudo copy-on-write without reset to Read Only)

- At the end of process quantum 0 (the first quantum):
  - All the process memory space is set to read-only,
  - current_ws_list is empty

- During each quantum:
  - The current working set (current_ws_list) of the process is increased when a page fault occurs; pages are then set to R/W.

- At the end of each quantum:
  - Only unmodified R/W pages are set back to read only and removed from current_ws_list.
  - Dirty bit of modified RW pages is reset
Evolution of the Index Score with the tick/second: with instrumentation (pseudo copy-on-write without reset to Read only)(2)
Evolution of the Index Score with the tick/second: with instrumentation (pseudo copy-on-write without reset to Read only)(3)

Zoom on variations with quantum size

The improve copy-on-write cost is:
- 41.16% for smallest quantum size
- 19.59 % for highest quantum size

Compare to the pseudo copy-on-write algorithm there is some improvement. But the cost of frequent page fault remains high.
Evolution of the Index Score with the tick/second: with instrumentation (pseudo copy-on-write without reset to Read only)(4)

Short tick

12800 ticks/second: tick≈78μs (instrumented with improved pseudo cow)
Evolution of the Index Score with the tick/second: with instrumentation (pseudo copy-on-write without reset to Read only)(5)

Zoom on variations with quantum size

The improve copy-on-write cost is:
- 7.57% for smallest quantum size
- 1.11 % for highest quantum size

The penalty was improved
Average parameters of working set evolution of a few processes

- The average number of pages in the working set is relatively low, around 100 pages for server processes and around 4 for dhrystone processes. Except the file system server, which has 300 pages as average.
- The working set size is relatively stable, considering the number of pages coming in the working set and the number of pages going out from the working set.
- The number of pages going out and coming back again is also relatively low. So the optimization algorithm to reduce the number of copy-on-write makes sense.
Evaluation with time of the working set of process (73142)

Long traditional tick

Evolution with time of the working set size (long traditional tick)
Evaluation with time of the working set of process (73142)
Our real-time problem:
the blended hardening technique (BHT)
Protecting computers running in space environments (erg. satellites) against cosmic radiation effects

- SEU = bit flips caused by cosmic radiations = transient errors
- Hardening central memories or caches against direct effects of SEU is common (ECC/scrubbing)
- Hardening processors in hardware is more difficult and very expensive;
- pure software techniques can only reduce, not eliminate SEU effects
- Blended hardening is a mostly software solution using limited hardware features, either simple hardened hardware or side effects of classical hardware components
Principle of Blended Hardening

- **Hypotheses**
  - SEU are “infrequent” events. For a short time interval (say 1ms) the probability of suffering from more than 1 SEU can be neglected.
  - There is a “protected memory”, i.e. a memory area that is both hardened (immune to direct SEU effects) and immune to indirect ones (changes caused by a program made faulty because of a SEU).

- **Principle:**
  - divide the program in short “processing elements”; run each of them twice and compare the results: same: OK, proceed; different: restart the processing element.
  - This is OK because only one of the executions or the comparison can be made faulty by a SEU. If the single error occurs in the comparison it will just cause redoing a correct computation.
Using BHT to protect user mode processes in the operating system

• Additional hypotheses:
  – The central memory is hardened against direct SEU effects
  – The MMU and the OS itself are assumed to be protected independently.
  – User processes only interact with the outside world through the OS.
  – External interrupts or exceptions do not change the memory state of hardened processes.

• Problems to solve:
  – Implementing “protected memory”
  – Dividing the process in atomic processing elements without any knowledge of the program
Processing elements in user processes

- **UPE = code executed (in user mode) between 2 system calls or limits of a time quantum:**
  - starts after system call; ends at system call or timer trap (execution of system calls is not included in UPE)
  - has thus no direct interactions with the outside world; its execution is atomic and idempotent
  - is run twice in BHT
  - results of the two runs are compared
  - Results will be the same if no SEU

- **Problems to solve:**
  - Implementing “protected memory”
  - Replaying exactly the same processing element
How to implement the protected memory concept using the MMU?

- MMU can protect memory by restricting access.

- Problem is to identify the “results” of the user space processing element (kernel PE are assumed to be protected otherwise).

- Solution: when starting a PE, set the whole process memory in RO mode and use copy on write: the result is the copied pages at the end of UPE execution.
Replaying exactly the same processing element?

- If UPE ends at system call: easy
- If UPE ends at time quantum:
  - hard: one must count the instructions and let the UPE run again for exactly the same number of instruction;
  - but feasible with modern processors

Can it be done with MINIX without an unacceptable performance penalty?
Conclusions

• Minix was a good choice
  – Code is clean and well documented
  – Micro-kernel architecture allows to handle some system functions as user processes.
  – It resists to changes to the clock rate and the time quantum.

• So far performance penalty looks acceptable but more tests will be necessary.
Questions ?
Comments ?