Toward (SOS) Self-stabilizing Operating System

Shlomi Dolev and Reuven Yagel, Ben-Gurion University, Israel

Sep. 1st SAACS’04 Workshop, Zaragoza
Outline

- Motivation: current operating systems do not stabilize!
- Background & related work
- Construction & proving method
- Work done: a self-stabilizing tiny SOS
- Tiny Demo
Linux and Windows do not stabilize.
…The Spirit rover has a radiation-hardened R6000 CPU from Lockheed-Martin Federal Systems…The operating system is Wind River Systems' Vx-Works..

…attempted to allocate more files than the RAM-based directory structure could accommodate. That caused an exception, which caused the task that had attempted the allocation to be suspended…

…Spirit fell silent, alone on the emptiness of Mars, trying and trying to reboot.

http://www.eetimes.com/sys/news/OEG20040220S0046
Why is it so hard?

- Corbató 91, "It almost goes without saying that ambitious systems never quite work as expected"

- "You must pay extreme attention to detail here. One wrong bit will make things fail… "
  http://my.execpc.com/~geezer/os/pm.htm

- From Pentium’s manual:
  “… if the ESP or SP register is 1 when the PUSH instruction is executed, the processor shuts down due to a lack of stack space. No exception is generated to indicate this condition"
Operating System Robustness

- Dijkstra, “THE” Multiprogramming System (1968) - Layered Design
- Denning, Fault tolerant operating systems (1976) - Protection
- Improving the reliability of commodity operating systems (SOSP 2003) - Isolating driver faults

- Checkpointing, Monitoring, Transactions, µ-kernels, etc.
Self-Stabilization

- The combination and type of faults cannot be totally anticipated in on-going systems
- Any on-going system must be self stabilizing (or manually monitored)
Self-Stabilization

- A self-stabilizing system is a system that can automatically recover following the occurrence of (transient) faults.

- Transient faults: soft-errors ("98% of RAM errors are soft errors"), wrong crc during communication etc.

- Dijkstra ’73. Self-Stabilizing in Spite of Distributed Control
- Dolev ‘2K. Self-Stabilization
Following any sequence of transient faults, the (operating) system converges

Self Stabilization:

- System can be started in an arbitrary state and converge to a desired behavior
- Traditionally used in distributed systems (where transient faults are frequent)
- Cannot run self-stabilizing algorithms unless hardware+OS are stabilizing (Fair composition [D2K])

Hence achieved: dependability, self-healing, automatic recovery, adaptive system, …
Related (Ongoing) Projects

- Autonomic Computing, IBM et. al.
- Recovery Oriented Computing, Berkeley-Stanford

BGU
- Self-stabilizing micro-processor [DH03]
- Self-stabilizing operating system
- Self-stabilizing preserving compiler [DH04]
- Self-stabilizing autonomic recoverer [BDK03]
- Self-stabilizing distributed file system [DK02]
- Self-stabilizing middleware [DS01]
Outline

- Motivation: current operating systems do not stabilize!
- Background & related work
- Construction & proving method
- Work done: a self-stabilizing tiny SOS
- Tiny Demo
Research Directions

- Black-box
  - Existing OS (Unix, Windows, RTOS)
  - Hybrid solutions e.g. Jaluna
- Carefully tailoring a micro kernel (targeting e.g. RFIDs)
  - Processor scheduling
  - Memory management
  - Device allocation
Working Method

• Define a goal for some sub-system and set its most essential requirements
• Gradually evolve proofed working self-stabilizing mechanisms
• Combine the solutions into a coherent system
• Examples follow …
Assumptions

- Every configuration is possible (Murphy’s Law)
- Program code is hardwired (in ROM) and is correct
- CPU instruction manual (e.g. x86\IA-32) defines a transition function.
- Self-stabilizing [DH03]
Solution 1 (Black Box)

✦ **Goal:** Adding stabilization to some operating system

✦ **Requirements:**
  - Defining a legal execution is usually impractical
  - Restore original state (variables + code), infinitely often

✦ **Solution:**
  - Periodic Reset Re-install and Execute
    - Watchdog timer (self-stabilizing)
    - Periodic processor reset
    - During bootstraps OS reinstall from ROM

✦ **Weak self-stabilization**
  - \( E = (c_i, a_i, c_{i+1}, \ldots, \text{RRE}, c_1, a_1, c_2, a_2, \ldots, c_i, a_i, c_{i+1}, \ldots, \text{RRE}, c_1, a_1, c_2, a_2, \ldots) \)
  - Is it always acceptable?
Solution 2 (Black Box)

- Periodic re-install code and continue execute
- Really non-maskable interrupt
  - Limit the option to mask NMI
  - NMI vector hardwired

- Consistency check & establishment
- (Hierarchical) boots if needed
Solution 3 (Tailored)

Tiny SOS Scheduler:

Goal: Processor management (in spite of process’ direct control)

Requirements:
- Self-stabilization
- Fairness
- Process stabilization preserving

Activation and full execution no matter what the system state is
Process counter increments modulo N
Monitor next process’ validity, e.g., PC in limits
Tiny SOS Scheduler

- ~70 lines of a real machine assembly code
- 16bit Real mode & 32bit Protected mode.
- Standard build and emulation tools (Nasm, ld, Bochs)
- Detailed proof of requirement preservation

; increase task
10 mov word ax, [currentProc]
11 and ax, PROC_MASK
...

; load task state
...
;restore ip
52 mov ax, [bx+4]
;validate ip
53 and ax, IP_MASK
54 mov word [ss:STACK TOP], ax
;restore general registers
55 mov cx, word [bx+12]
56 mov dx, word [bx+14]
57 mov si, word [bx+16]
58 mov di, word [bx+18]
Tiny SOS Scheduler

Any State
Tiny SOS Scheduler

Clock tick / execute next

Any State
Tiny SOS Scheduler

Clock tick / execute next

Any State

NMI / load PC with scheduler handler
Tiny SOS Scheduler

Any State

Clock tick / execute next

NMI / load PC with scheduler handler

Establish Scheduler Consistency
Tiny SOS Scheduler

Clock tick / execute next

Any State

NMI / load PC with scheduler handler

Establish Scheduler Consistency

Next Process Validated & Ready
Tiny SOS Scheduler

Clock tick / execute next

Any State

NMI / load PC with scheduler handler

Establish Scheduler Consistency

Doing Process

IRET / Jump to next task

Next Process Validated & Ready
Tiny SOS Scheduler

Clock tick / execute next

Any State

NMI / load PC with scheduler handler

Establish Scheduler Consistency

Clock tick / execute next

Doing Process

IRET / Jump to next task

Next Process Validated & Ready
Tiny SOS Scheduler

- Any State
  - Clock tick / execute next
  - NMI / load PC with scheduler handler

- Process(ing)
  - Clock tick / execute next
  - NMI / load PC with scheduler handler

- Establish Scheduler Consistency
  - IRET / Jump to next task

- Next Process Validated & Ready
  - IRET / Jump to next task
Tiny SOS Scheduler

Any State

Clock tick / execute next

Establish Scheduler Consistency

Next Process Validated & Ready

Process(ing)

Some Error

Some Error

Some Error
Tiny SOS Scheduler

Clock tick / execute next

Any State

NMI / load PC with scheduler handler

Establish Scheduler Consistency

Next Process Validated & Ready

Process(ing)
Proof (a few lemmas)

- In every execution $E$, the code of the scheduler is started to be executed and is executed from the first instruction to the last instruction infinitely often.
- In every execution $E$ of the scheduler, each process is executed infinitely often.
- The self-stabilizing scheduler preserves stabilization of processes.
VGA BIOS - Version 2.40
Copyright (C) 1990-2000 Elpin Systems, Inc.
All rights reserved.

Licensed for use with bochs, courtesy of MandrakeSoft.

For information on this or other VGA development products, contact Elpin Systems at: (800) 723-9038 or www.elpin.com
Bochs BIOS, 1 cpu, $Revision: 1.85.2.1 $ $Date: 2003/01/16 21:58:42 $

Booting from Floppy...

Loading Boot Image
.............
.............
....
Starting to load Self-Stabilizing OS.

Welcome to SIMPLIX.
Booting from Floppy...

Loading Boot Image

 ................
 ................
 ....
 Welcome to SIMPLIX.

Loading OS interrupt handlers at address 20000 (ROM!)
Loading task code at address 10000
Installing interrupt handlers at addresses 0x20 (=0x84), 0x70 (=0x1c4) (ROM!)
Jumping to first task code at address 10000

"Everything should be made as simple as possible, but no simpler." (Einstein)
Booting from Floppy...

Loading Boot Image
.............
.............
....
Welcome to SIMPLIX.
Loading OS interrupt handlers at address 20000 (ROM!)
Loading task code at address 10000
Installing interrupt handlers at addresses 0x20 (=0xB4), 0x70 (=0x1C4) (ROM!)
Jumping to first task code at address 10000

Scheduler switches to next task, counter= 19

0) Everything should be made as simple as possible, but no simpler. (Einstein)
1) Keep It Simple, Stupid. (KISS engineers’ principle)
2) The unavoidable price of reliability is simplicity. (Hoare)
3) What can be done with fewer ... is done in vain with more. (Ockham)
Conclusion

- **Self-Stabilizing operating-system is a must**
- **This work shows theoretical and practical ways to achieve this goal**
- **We believe autonomic computing and self-* should benefit from the foundation of self-stabilization**
- [http://www.cs.bgu.ac.il/~yagel/sos](http://www.cs.bgu.ac.il/~yagel/sos)

Thank you