



Self-Stabilizing Operating Systems

Shlomi Dolev and Reuven Yagel
Computer Science Department
Ben-Gurion University of the Negev,
Beer-Sheva, Israel



SOSP'05 \ SIGOPS
Doctoral Workshop, Oct 23,
2005

SOS - Motivation

- Growing use of autonomous and remote systems (e.g. RFID)
- Human management is too expensive, risky or just unavailable
- The combination and type of faults cannot be totally anticipated in on-going systems (due to e.g. SEU\soft errors)
- Pentium HALTING problem: "... if the ESP or SP register is 1 when the PUSH instruction is executed, the processor shuts down..."

2

Proposed solution

- To build according to the well defined and understood paradigm of self-stabilization (traditionally used in distributed systems)
- Thereby achieving: trustworthiness, dependability, self-healing, automatic recovery, adaptive systems, ...



3

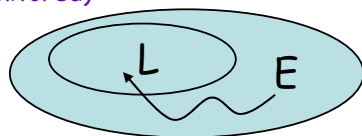
Self-stabilization

- Elegant fault tolerant approach
 - Started at any state, the system converges to a desired behavior
- Generally used in distributed systems
 - Routing, clock synchronization, leader election, etc.
- Overcome transient faults in the system.
 - Transient faults: soft-errors ("98% of RAM errors are soft errors"), wrong CRC during communication etc.

4

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 in Spite of Distributed Control" [Dijkstra '74]
- Self-Stabilization [Dolev '00]

5

OS Reliability

- Past examples:
 - Dijkstra, "THE" Multiprogramming System '68 (Layered Approach)
 - Denning, Fault tolerant operating systems '76 (Protection)
 - KeyKOS '85, EROS '92 (Capabilities, Checkpoints)
 - Micro-kernel ~'90, Exo-kernel '94 (Minimal TCB)
- Current
 - JHU: The Coyotes Secure Operating System
 - IBM: K42, Autonomic Computing
 - SUN: Solaris 10, Predictive Self-Healing
 - MSR: Singularity, managed code OS

6

Goal: Autonomic Computer

- Following any sequence of transient faults (e.g. soft-errors), the (operating) system converges
- Using self stabilization:
 - A system can be started in an arbitrary state and converge to a desired behavior
 - Using fair composition to run hardware+OS
- BGU: Self-stabilizing systems, tools & paradigms
 - Microprocessor [DH'04]
 - Operating System [DY'04]
 - Compiler [DH'05]
 - Framework: autonomic recoverer [BDK'03]
 - Middleware: File System [DK'02], Group Comm. [DS'01]

7

SOS - Directions

- Black-box
 - Take existing (Desktop/Real-time) OS
 - Add stabilization layer
 - Detailed formal specification needed
- Carefully tailoring a tiny kernel
 - Processor scheduling [SAACS04]
 - Memory management [SSS05]
 - Device drivers



8

Method

- Additional requirements for each OS function
- Evolve self-stabilizing solutions that follow computer-architecture/OS progress
- Detailed proof for self-stabilization of algorithms AND implementation
- Processor (e.g. Pentium) instruction manual defines a transition function
 - Don't rely on existing compilers

9

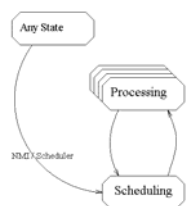
Assumptions

- Whole soft-state can be corrupted (e.g. Program Counter)
- Stabilization of other layers

10

Solution Foundations

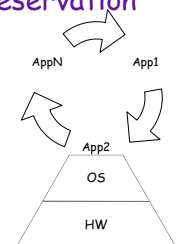
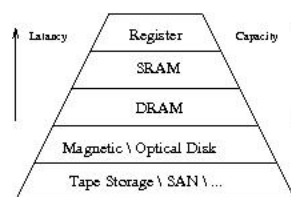
- Program loading & process scheduling
- Code portions in ROM
- Truly non-maskable interrupt and watchdog architecture
- Periodic reset reinstall & execute (weak)
- continuous monitoring and consistency enforcement



11

Example: Memory Management-Requirements

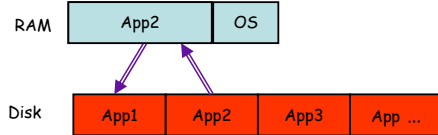
- Consistency of memory hierarchy
- Self-stabilization preservation



12

Solution 1: Full Swapping

- Allocate whole available memory to the running application

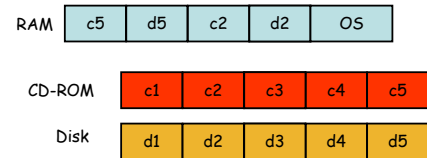


- Consistency: kept all the time
- Stabil. Preserving: no mutual sharing

13

Solution 2: Fixed Partitioning

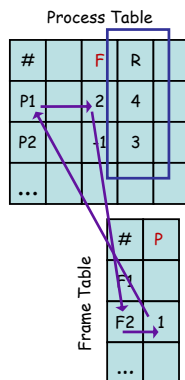
- Fixed slots in main memory for several programs



14

Solution 2: Fixed Partitioning

- Consistency: through continuous checks and consistency establishment of OS data structures
- Stabil. Preserving: via segmentation + code refreshing



15

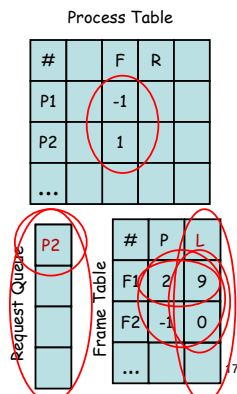
Solution 3: Dynamic allocations

- We want to allow applications to dynamically allocate memory
- How can we avoid a process that (faultily) allocates the whole available memory?
- What happens if a process "forgets" about its ownership?
- Leasing

16

Solution 3: Dynamic allocations

- Consistency: dynamic memory is temporarily leased & garbage collected, verification of PCB & queue
- Stabil. Preserving: access through special segment selector



17

Implementation

- Pentium in real-mode, single address space
 - Simple
 - common for sensors/microcontrollers
 - Protected mode & VM mechanisms can be handled accordingly
- Code size: ~1-2K
 - TinyOS ~1K
 - VxWorks ~10²K
 - Linux kernel ~4M
- Fault injection with the Bochs simulator

18

Implementation

```
1) MM_FindFrame:  ;(PT, FT, i)
; al contains current frame suggestion
; nf <- (frame[PT[i]] + 1) modulo M
2)    and         byte [bx+FRAME_COL], FRAME_MASK
3)    inc         al
4)    and         al, FRAME_MASK

;Check all slots for an empty one.
;while nf != frame[PT[i]] and FT[nf] != nil
5) while1:
6)    cmp         al, [bx+FRAME_COL]
7)    jz          endwhile1
8)    lea        si, [frames]
9)    add         si, ax
10)   mov        dl, [si]
11)   cmp        dl, NULL_PROCESS
12)   jz         endwhile1
; do nf <- (nf + 1) modulo M
13)   inc         al
14)   and         al, FRAME_MASK
15)   jmp        while1
16) endwhile1:
; return found frame number in register 'al'
17)   ret
```

19

```
UGA 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 UGA development products, contact
Elpin Systems at: (800) 723-9038 or www.elpin.com

bochs BIOS, 1 cpu, $Revision: 1.103.2.2 $ $Date: 2004/02/02 22:39:22 $

Booting from Floppy...
Welcome to SOS
Loading 2nd stage to address 1000h
Loading interrupt handlers at address 80000h (UROM!)
Installing interrupt table at addresses 0 (UROM!)
-
```

Future Work

- I/O device drivers
 - Major cause of operating systems failures
 - Co-operation of more than one microprocessor
 - Detailed driver / General monitoring layer
- Gather the different parts
- Micro-kernel / VMM

21

Conclusion

- The work shows theoretical and practical ways to achieve the goal of a self-stabilizing OS
- The (system) research community & industry can benefit from the foundation of self-stabilization
- <http://www.cs.bgu.ac.il/~yagel/sos>

22