University of Glasgow | School of Computing Science

Dependable Kernels and Device Drivers

Advanced Operating Systems (M)
Tutorial 5

Tutorial Outline

- Review of Problem Set 2
- Review of lectures
- Key learning outcomes
- Discussion

Review of Problem Set 2

- Question 1: Consider a system comprising three independent periodic tasks T_1 = (4, 1), T_2 = (5, 1) and T_3 = (10, 2). Demonstrate that this system can be scheduled in a preemptive manner on a single processor using a) the rate monotonic algorithm; and b) the earliest deadline first (EDF) algorithm.

Review of Problem Set 2

- Question 2: The system from question 1 must also support the execution of three aperiodic jobs: A_1 which is released at time 1, A_2 which is released at time 8, and A_3 which is released at time 12. Each aperiodic job executes for 1 unit of time. The system is scheduled using a preemptive rate monotonic scheduler, on a single processor, with a server task, T_s = (3, 0.5) to schedule the aperiodic jobs.
- Draw a diagram to illustrate the scheduler if T_s is implemented as a polling server. What are the response times of the aperiodic jobs?

Review of Problem Set 2

- Question 3: Is the system from question 2 schedulable if T_s is implemented as a deferrable server? What are the response times of the aperiodic jobs?

Review of Problem Set 2

- Question 4: Is the system from question 2 schedulable if T_s is implemented as a fixed-priority sporadic server? Explain your answer.

Review of Problem Set 2

- Question 5: The fixed-priority sporadic server has a complex set of budget consumption and replenishment rules. Discuss whether you believe the benefits of this server outweigh its complexity.

Review of Lectures (1)

- Dependable device drivers
  - Sources of bugs in device drivers
    - Based on a survey of Linux device driver bugs: 23% general, 38% hardware device handling, *39% concurrency and interactions with other kernel subsystems*
  - Improving device drivers: engineering approaches
    - Applying object oriented languages and techniques
    - Example: Apple MacOS X I/O Kit
  - Advancing the driver model
    - Making device driver state machines explicit in the source code
    - Automatic verification of device driver code to ensure correct implementation of protocols for interaction with other kernel subsystems, and to ensure correct locking
    - Example: device drivers in Microsoft's Singularity operating system

Review of Lectures (2)

- Dependable Kernel Architectures
  - Moving on from the monolithic, C-based, kernel architecture
  - Memory models
    - Example: Java memory model
  - Safe languages
    - Well-typed languages don't have undefined behaviour
    - Banishing the null pointer
    - Pattern matching and messages
    - Immutable data
    - Linear types
  - Microkernels and strongly isolated systems
    - Traditional microkernel model with hardware memory protection
    - Microkernel with software isolated processes: use the type system to enforce isolation between software processes, with no need for hardware protection
    - Example: Singularity from Microsoft Research

Key Learning Outcomes

- Understand how managed code and advanced type systems might be used in the design and implementation of future operating systems
  - To help the implementation of subsystems, such as device drivers
  - To help structure the system architecture, in the form of software isolation and robust message passing and concurrency primitives

Discussion

- Does the model of a microkernel with software isolated processes look plausible to you?
  - Advantages and disadvantages?
  - Benefits and limitations?

Discussion

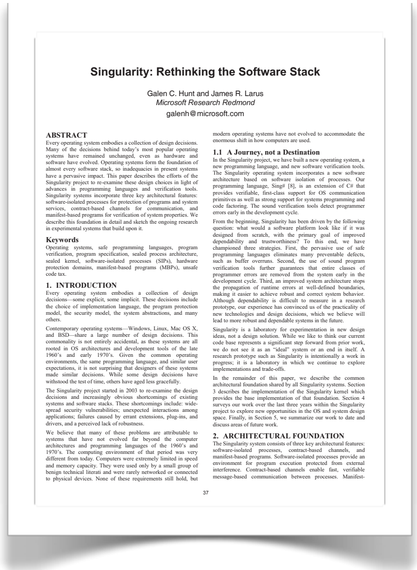
- The architectures we have considered rely on a core unsafe microkernel, around which the safe systems architecture is constructed
  - To what extent can the microkernel be written in a safe language?
  - Can some operations only be implemented in an unsafe manner?

Discussion

- Safe languages require more extensive runtime support and error checking → more overhead
  - E.g., checking array bounds, support for exception handling, garbage collection overheads
  - To what extent do reducing context switch times and faster message passing compensate for this overhead in the software isolated process model?

Discussion

- Questions, comments, and discussion on the papers?



Summary

- Widely deployed operating systems are based on the heritage of monolithic, C-based, kernels
  - Linux is the newest system to obtain wide deployment, but has little innovation in terms of fundamental systems architecture
  - Research prototypes are beginning to explore radically different system architectures