‘QoS Safe’ Kernel Extensions for Real-Time Resource Management

Richard West and Jason Gloudon

Boston University

Computer Science

Introduction

- General purpose systems have limitations:
  - Ill-equipped to meet service requirements of complex real-time applications

- Aim to **extend** COTS systems to:
  - better meet the service needs of individual applications
  - provide finer-grained service management than at user-level
  - adapt system behavior to compensate for changes in resource needs and availability

Bridging the `Semantic Gap'

- There is a `semantic gap' between the needs of applications and services provided by the system

  - Prior solutions to bridge this gap include:
    - Middleware (e.g., RT CORBA, QuO)
      - Heavyweight
    - Implementing functionality directly in applications
      - Inflexible
      - Must leverage system abstractions in complex ways
    - Special systems designed for extensibility
      - e.g., SPIN, VINO
      - Not COTS-based or `QoS Safe'

Extending COTS Systems

- Desktop systems now support QoS-constrained applications
  - e.g., Windows Media Player, RealNetworks Real Player

  - Therefore desirable to extend COTS systems but…
    - Many such systems are monolithic and not easily extended
- Some systems provide limited extensibility
  - e.g., kernel modules for device drivers in Linux
  - However, no support for extensions to override system-wide service policies

Extensibility and Safety

- Kernel-level extensions must be `QoS safe’
  - Traditional safety concerns must be maintained
    - Address space / memory protection & type-safety
    - Need to maintain integrity of system
  - Resource management decisions for one application must not adversely affect another application
  - Extension code must have bounded execution time
    - Execution time must be small enough not to impact behavior of system

Contributions

- SafeX – Safe Kernel Extensions
  - Extension architecture for general purpose systems
  - Allow applications to customize system behavior
    - Extensions affect service management decisions
    - Can lead to fewer service violations for RT tasks, compared to user-level management methods
- Mechanisms to provide `QoS Safety'
  - Provide mechanisms for meeting application-specific QoS constraints while maintaining system integrity

SafeX Goals

Computer Science

- To allow untrusted apps to dynamically-link `QoS safe' code into the kernel
  - Can deploy code on remote hosts
- To allow app-specific service monitoring and adaptation
- To improve QoS for real-time applications
  - even when there are changing resource demands
  - compared to user-level solutions

SafeX Approach

- Supports compile- and run-time safety checks to:
  - Guarantee QoS
    - The QoS contract requirement
  - Enforce timely & bounded execution of extensions
    - The predictability requirement
  - Guarantee an extension does not improve QoS for one application at the cost of another
    - The isolation requirement
  - Guarantee internal state of the system is not jeopardized
    - The integrity requirement

SafeX Features

Computer Science

- Extensions written in Popcorn & compiled into Typed Assembly Language (TAL)
  - TAL adds typing annotations / rules to assembly code
- Memory protection:
  - Prevents forging (casting) pointers to arbitrary addresses
  - Prevents de-allocation of memory until safe
- CPU protection:
  - Requires resource reservation for extensions
  - Aborts extensions exceeding reservations
  - SafeX decrements a counter at each timer interrupt to enforce extension time limits

Exception Handling

- The compiler inserts runtime safety checks into extensions
- Exceptions e.g., div-zero, null-pointer derefencing are caught by specified extensions handlers or default SafeX handlers

Synchronization

- Extensions cannot mask interrupts
  - Could violate CPU protection since expiration counter cannot decrement
- Problems aborting an extension holding locks
  - e.g., extension runs too long
  - May leave resources inaccessible or in wrong state

  - SafeX restricts synchronization primitives to core kernel code or SafeX code
    - Extensions access shared resources via SafeX interfaces

Additional Support

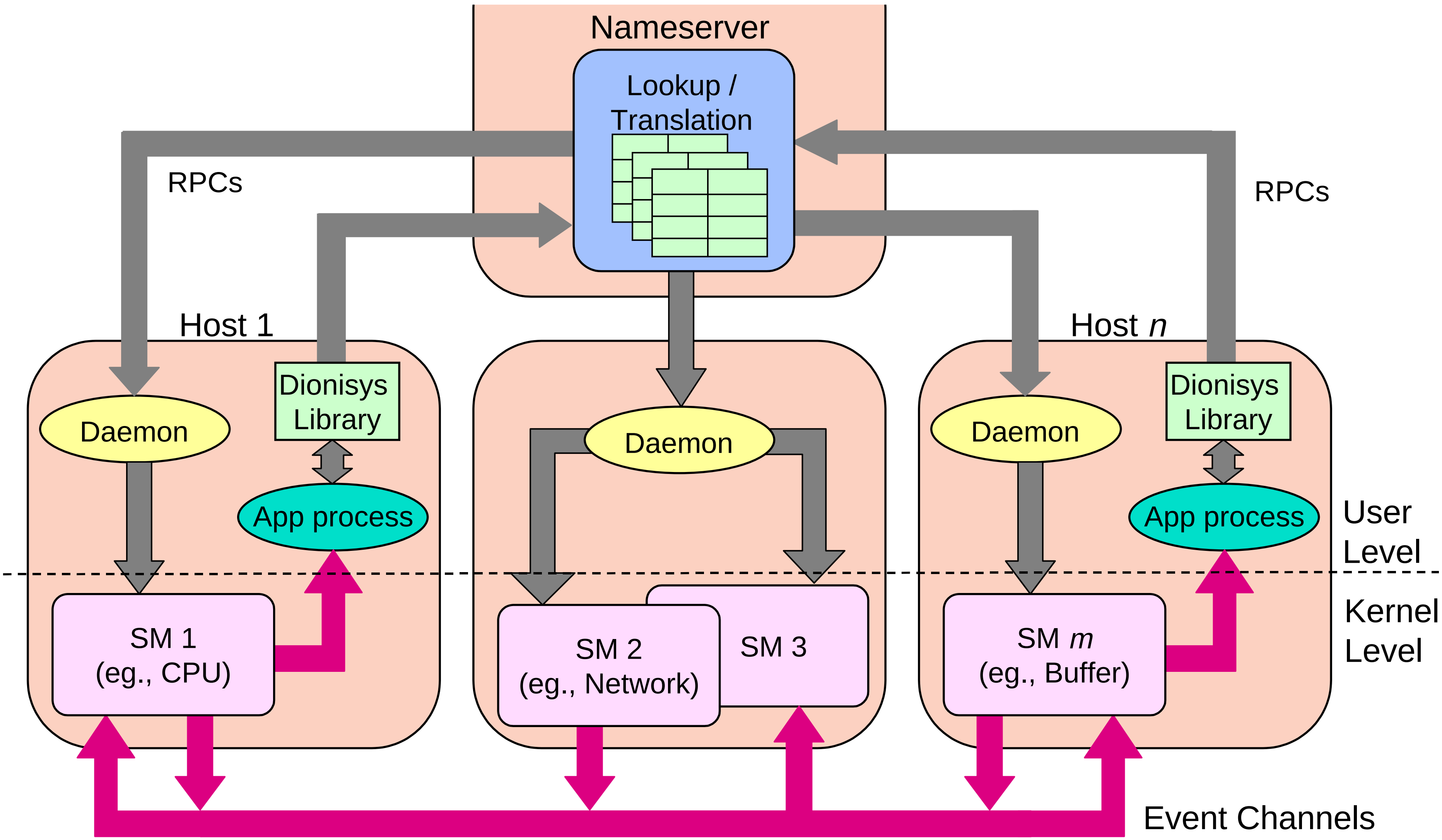
Computer Science

- SafeX runs in the daemon processes of the Linux Dionisys QoS system

- Applications link w/ the Dionisys library to create:
  - Service monitoring extensions (monitors)
  - Service adaptation extensions (handlers)
  - Application-specific service manager extensions

  - QoS attribute classes
  - Event channels between monitors and handlers

Linux Dionisys Overview



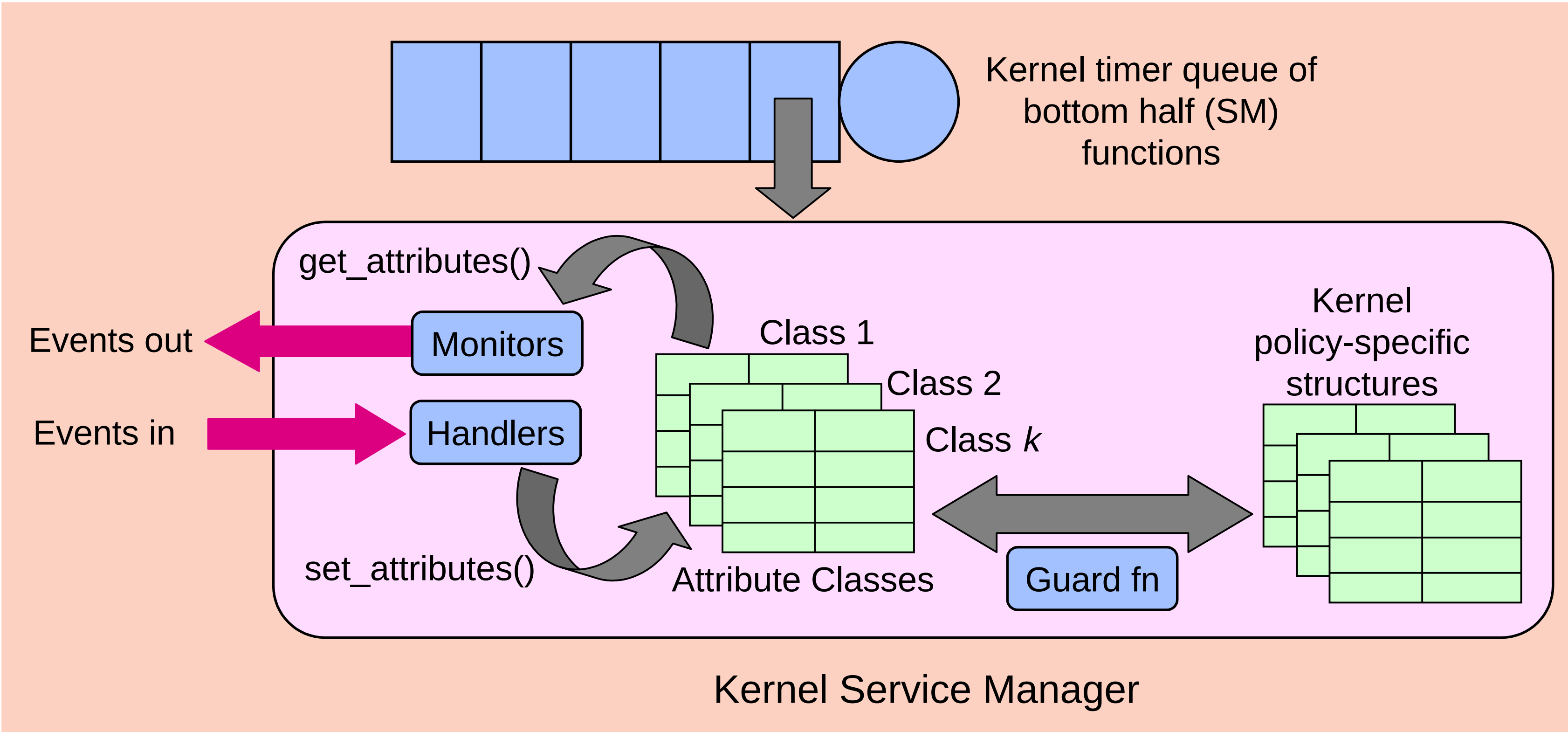
Service Managers

Boston University
Computer Science

- Encapsulations of resource management subsystems
- Have policies for providing service of a specific type
  - e.g., a CPU service manager has policies for CPU scheduling and synchronization
- Run as bottom-half handlers (in Linux)
  - Invoked periodically or in response to events within system
- Invoke monitor and handler extensions
  - Can execute asynchronously to application processes
  - Apps may influence resource allocations even when not running

Kernel Service Managers

Computer Science



- Monitors & handlers operate on attribute classes
  - name-value pairs (e.g. process priority – value)
- Service extensions with valid access rights can modify attributes

Attribute Classes & Guards

- Each host of a Dionisys system has an attribute class per application
  - Identified by class descriptors
- Attribute classes can be deployed on remote hosts
  - Access to these classes is granted to the extensions of processes that acquire permission from the class creators
- Guard functions are generated by SafeX
  - Responsible for mapping values in attribute classes to kernel data structures
  - Can enforce range and QoS guarantee checks

SafeX Interfaces

- SafeX provides get_/set_attribute () interfaces
  - Extensions use these interfaces to update service attributes
  - Extensions are not allowed to directly access kernel data structures
- Interfaces can only be used by extensions having necessary capabilities
  - Capabilities are type-safe (unforgeable) pointers
- Interfaces limit global affects of extensions
  - Balance application control over resources with system stability

Experimental Evaluation

Computer Science

- Experiments to compare adaptive CPU service management at kernel- and user-levels
- Aim: to meet the needs of CPU-bound RT tasks under changing resource demands from a `disturbance' process
- Platform:
  - 500MHz Pentium III, 128MB RAM
  - Patched 2.4.17 Linux kernel w/ SafeX & Dionisys features

Kernel Service Management

Computer Science

- A service manager monitors CPU utilization and adapts process timeslices
  - Timeslices adjusted by PID function of target & actual CPU usage
  - Monitoring performed every 10mS
- Kernel monitoring functions invoked via timer queue

User-Level Management

- A periodic RT process acts as a CPU service manager
  - Reads /proc/pid/stat
  - Adapts service via kill() syscalls
    - Using SIGSTOP & SIGCONT signals

Experimental Setup (1)

Computer Science

- (A) 3 MPEG encoding processes, P1, P2 & P3
    - P1 – target CPU = 20mS every period = 100mS
    - P2 – target CPU = 30mS every 100mS
    - P3 – target CPU = 80mS every 200mS
  - Repeatedly encode 56KB frames (160x120, 24bit)
- (B) 3 hardloop processes, P1, P2 & P3
    - P1 – target CPU = 40mS every period = 400mS
    - P2 – target CPU = 100mS every 500mS
    - P3 – target CPU = 60mS every 200mS
- An MMPP disturbance (CPU "hog")
  - 10 sec exponential inter-burst gap & 3 sec geometric burst lengths

Experimental Setup (2)

Computer Science

- Each app process has initial RT priority =

  80 x (target / period)

  - target & period denote target CPU time in a given period

- User-level service manager & disturbance start at RT priority = 96

- Kernel daemons run at RT priority = 97
- Utilization points recorded over 1 sec intervals

Monitors and Handlers


```
void monitor () {
  actual_cpu = get_attribute ("actual_cpu");
  target_cpu = get_attribute ("target_cpu");
  raise_event ("Error", target_cpu - actual_cpu);
}

void handler () {
  e[n] = ev.value; // nth sampled error

  /* Update timeslice adjustment by PID fn of error */
  u[n] = (Kp+Kd+Ki).e[n] - Kd.e[n-1] + u[n-1];

  set_attribute ("timeslice-adjustment", u[n]);
}
```

Guard Functions

Boston University Computer Science

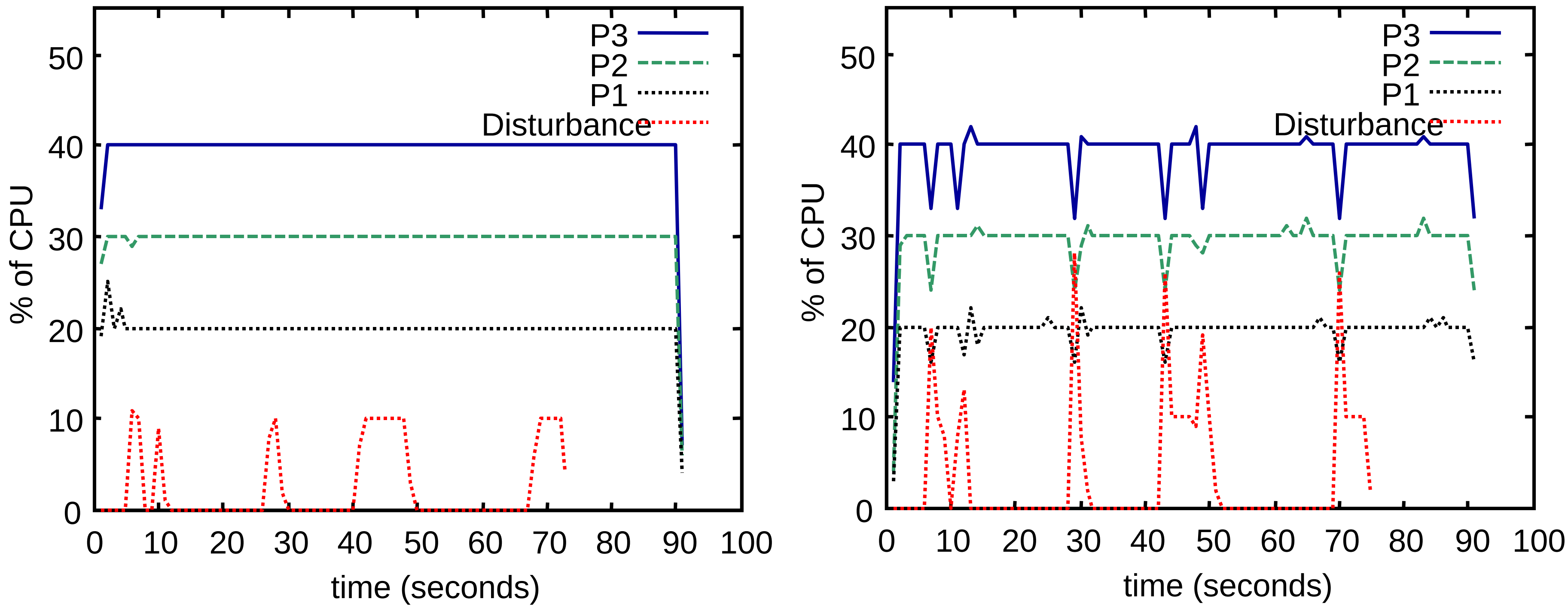
```
// Check the QoS safe updates to a process' timeslice

guard (attribute, value):
  if (attribute == "timeslice-adjustment")
    if (CPU utilization is QoS safe)
        timeslice = max (0, target_cpu + value);
    else block process;
```

- CPU utilization is deemed QoS safe if:
  Avg utilization over 2*period <= target utilization

(A) MPEG Encoding Results

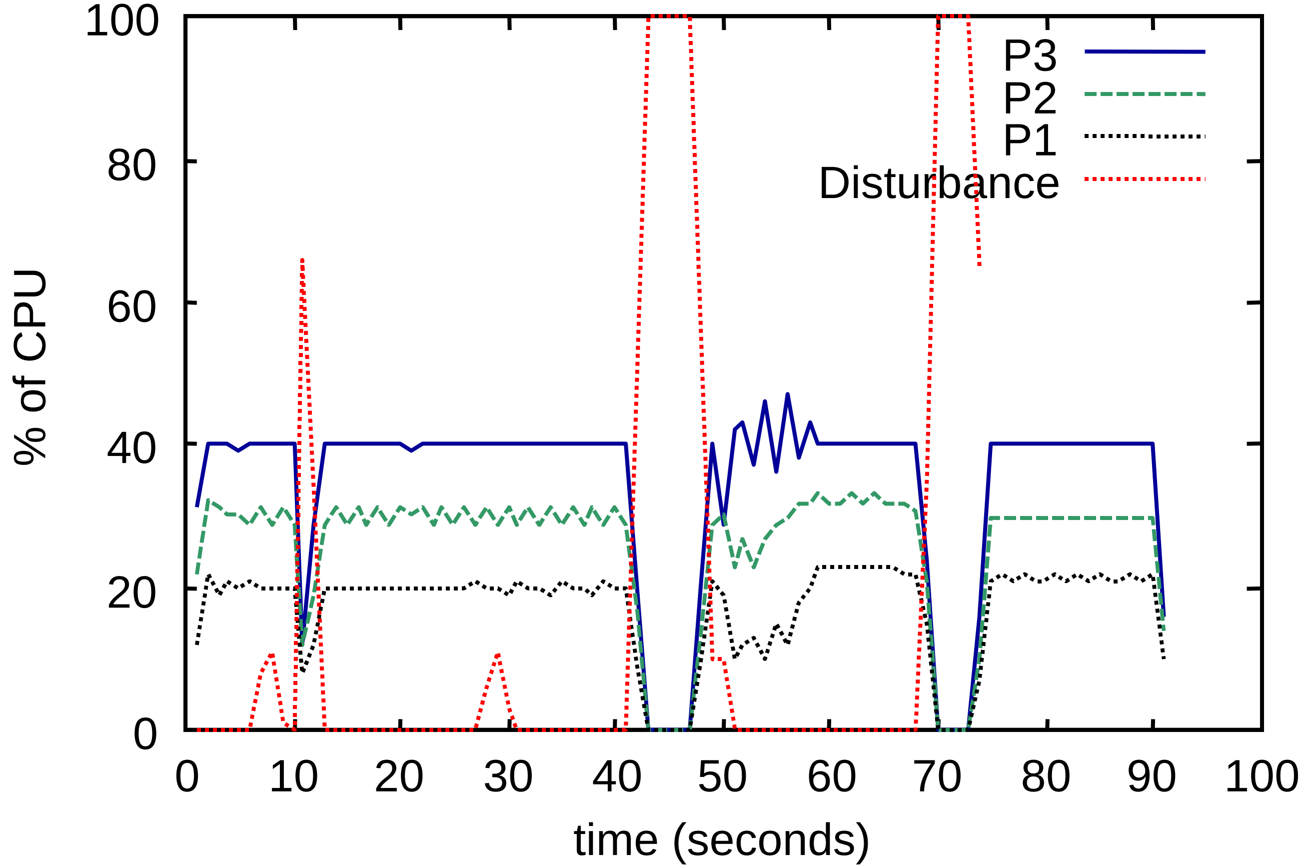
Computer Science



- Less service oscillation in left graph for kernel service management
  - Transient overloads do not affect service guarantees
- Right graph uses SCHED_RR scheduler for disturbance

(A) MPEG Encoding Results

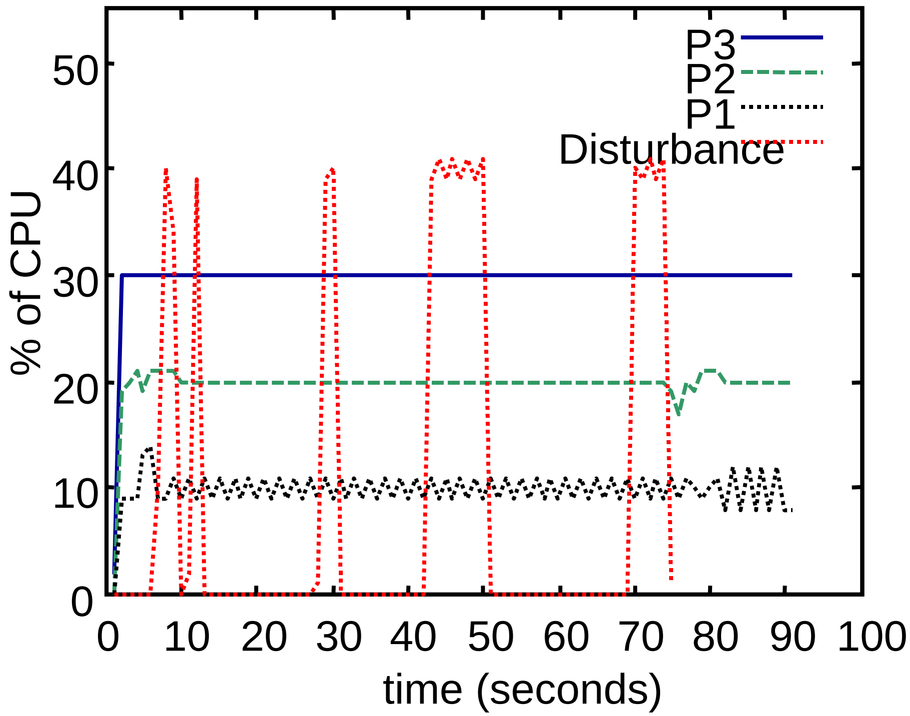
Computer Science

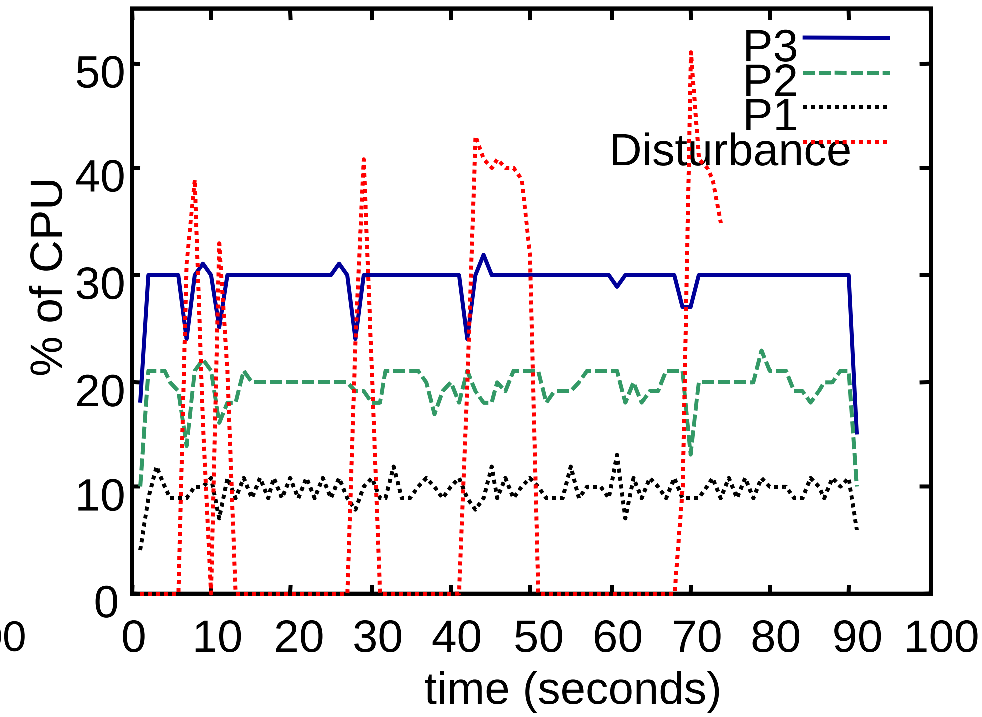


- Results for SCHED_FIFO scheduling
- User-level SM is blocked for duration of disturbance

(B) Hardloop Results

Computer Science

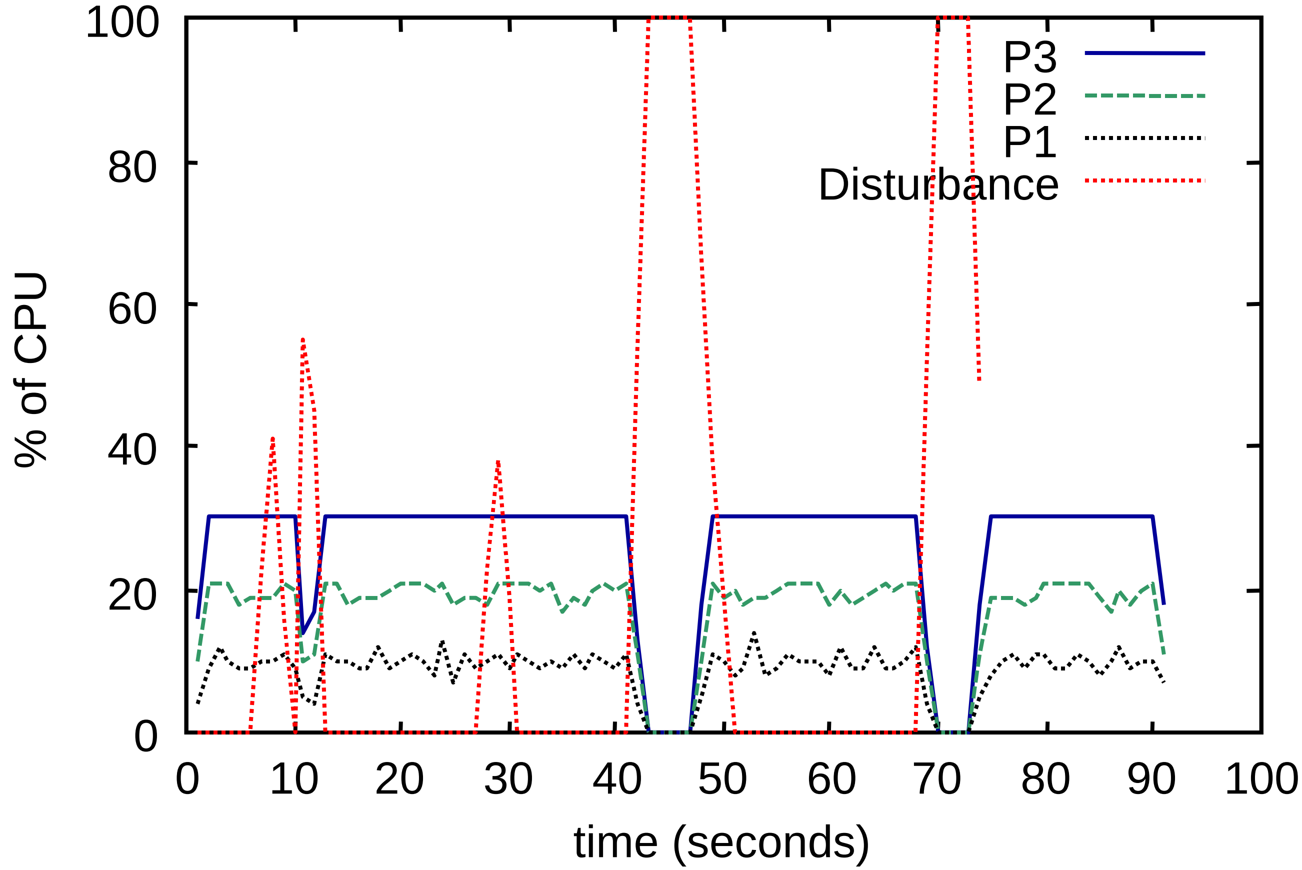




- Left = kernel service management
- Right = user-level management w/ SCHED_RR scheduling

(B) Hardloop Results

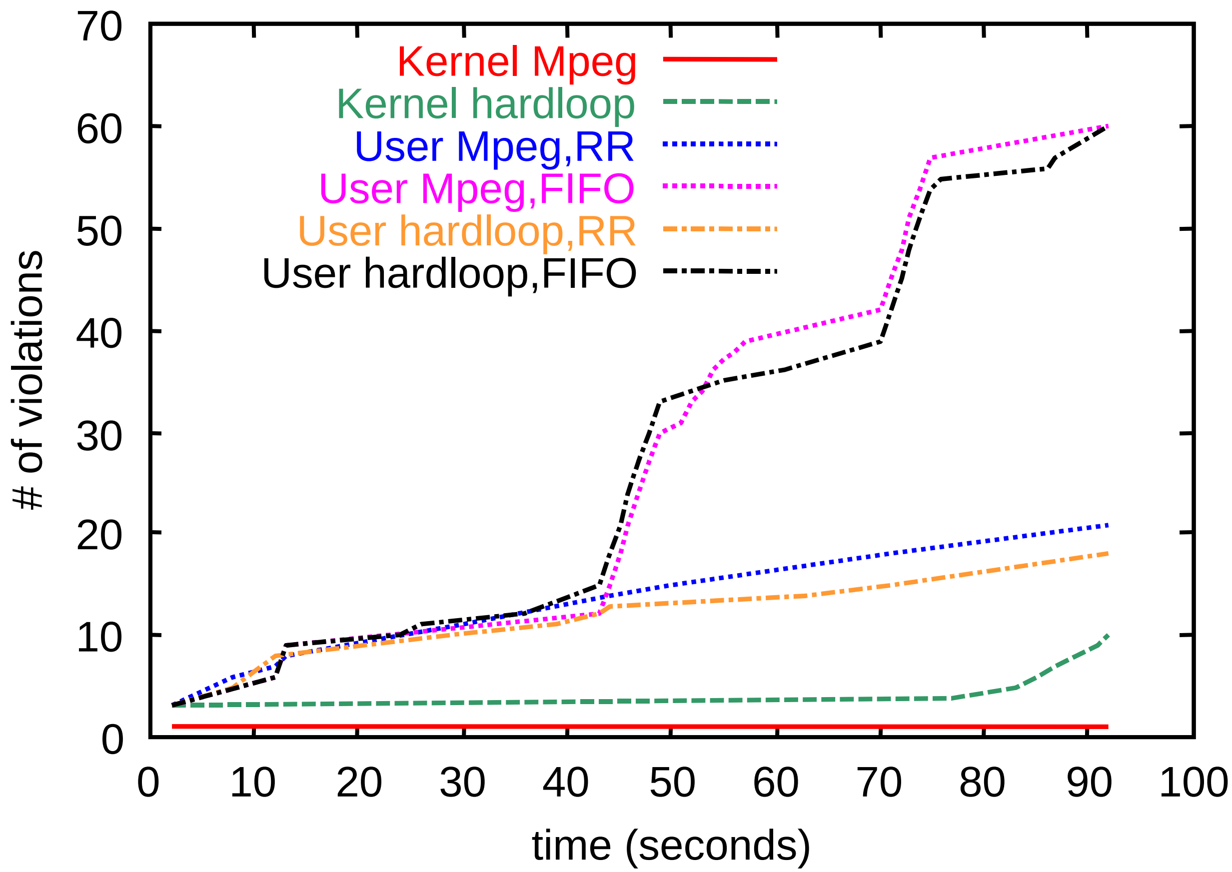
Boston University
Computer Science



- Results for SCHED_FIFO scheduling
- User-level SM is blocked for duration of disturbance

Service Violations

Computer Science



- Service violations occur when processes receive less than their target fraction of CPU time over their specified periods

Benchmarks

- User-level:
  - Signal dispatch = 1.5μS
  - Context-switch between SM and app process = 2.99μS
  - Reading /proc/pid/stat = 53.87μS
  - Monitors and handlers (for 3 processes) = 190μS
- Kernel-level:
  - Executing monitors and handlers (for 3 processes) = 20μS

Conclusions

Computer Science

- SafeX supports safe dynamic-linkage of code into the (Linux) kernel
- SafeX uses compile- & run-time support to create protection domains in the kernel
  - Provides memory and CPU protection for extensions
- Safe kernel extensions provide finer-grained service than user-level approaches
  - No scheduling of processes for service management
  - Not dependent on scheduling policies and timeslice granularities