RESEARCH STATEMENT

My research has focused predominantly on real-time systems, with particular attention given to the development of operating systems, kernels, scheduling algorithms, and resource management policies. The overarching objective of my research is to ensure the safe, predictable and efficient execution of tasks and services in safety-critical systems, even in the presence of faults, both in software and hardware, or through malicious attacks to breach system security. My work considers policies, mechanisms, system organization, and hardware features to enforce temporal and spatial isolation of tasks as they execute and compete for system resources. The aim is to support safety-critical applications including those in health-care, avionics, next-generation automotive systems, manufacturing, robotics and emerging areas such as the Internet-of-Things (IoT).

My scheduling and resource management work has led to the development of policies such as Dynamic Window-Constrained Scheduling (DWCS), Virtual Deadline Scheduling (VDS), and Cache-Aware Fair and Efficient Scheduling (CAFÉ). I have also developed techniques to integrate the scheduling of processes and interrupts, to avoid priority inversion problems with interrupts being serviced out-of-band and in precedence of higher-priority tasks. Similarly, I developed techniques in systems such as Linux and my own Quest operating system, to charge the overhead of handling interrupts to an accountable thread of control, rather than the current process suffering preemption.

Originally, DWCS was designed to be a network packet scheduler, that limited the number of late or lost packets over finite numbers of consecutive packets in loss-tolerant and/or delay-constrained, heterogeneous traffic streams. For example, many multimedia applications (including video-on-demand and streamed audio) can tolerate a certain fraction of late or lost information, without any noticeable degradation in the quality of playback at the receiver. However, care must be taken to prevent excessive numbers of consecutive packet losses. With DWCS, one can define a window-constraint, x/y, to limit the maximum tolerable number of lost or discarded packets to be no more than x for every y requiring service. This is similar to skip-over, or m-out-of-k, scheduling algorithms that were popular in the development of multimedia systems in the 1990s. DWCS could equally be applied to tasks, controlling the extent to which tasks are serviced late or simply discarded in preference to other time-critical tasks. VDS extended DWCS by combining task or packet service intervals with their window constraints, to produce a series of virtual deadlines. The task or packet with the earliest virtual deadline received precedence over others, similar to how proportional share schedulers operate. Work on DWCS, VDS and process-aware interrupt scheduling and accounting has been published in multiple IEEE RTSS and RTAS conferences, and journals such as the IEEE Transactions on Computers.

In CAFÉ, I worked with colleagues at VMware to develop techniques to schedule threads (and virtual machines) on multicore processors with shared last-level caches. Multicore processors provide combined power and performance benefits over higher clock frequency uniprocessors, and today they are ubiquitous across servers, desktops, and embedded devices including smart-phones and tablets. However, these processors tend to have shared last-level (L2 and L3) caches, leading to co-runner performance variations due to conflict misses. That is, one software thread can evict the cache contents of a thread running on another core. This interference makes it difficult to ensure performance isolation amongst competing threads on a multicore processor. Moreover, hardware management of caches is typically opaque to the software running on multicore processors. While at VMware, I developed a patented technique to use hardware performance counters found on modern processors, to accurately determine cache occupancy variations as one thread executes and triggers cache-line fills, while others suffer cache line evictions when not running. I developed techniques for modeling cache "pressure" amongst competing threads, and compensating proportional fair scheduling algorithms for cache conflict misses. This culminated in CAFÉ, a cache-aware fair and efficient scheduler for multicore processors, which was tested in VMware's ESX Server hypervisor, and also Intel's CMPSched$im simulator. Performance results showed significant reductions in co-runner performance variation amongst threads, which were able to make progress in accordance to their weighted CPU shares.

Over the past ten years I have focused attention on the development of my own real-time operating system, called Quest (see *http://www.questos.org* for more details). I wrote the original version of Quest in 2005, as a platform to investigate real-time scheduling policies. Writing real-time scheduling policies and resource management mechanisms for systems such as Linux was proving increasingly difficult, due to the complexity of dealing with control flow interactions in a system comprising many millions of lines of code. Quest provided the basis for a clean-slate design: a system that focuses on *time as a first-class resource*, and which is robust against software faults.

Towards this end, I developed a *Virtual CPU* (VCPU) scheduling framework for Quest. Quest's VCPU scheduler assigns a local scheduling queue to each physical CPU (core or hardware thread). Tasks and interrupt service routines run in the context of software threads, and are bound to specific VCPUs. Each VCPU is assigned a pair, C and T, to guarantee its runnable threads at least C CPU cycles every window of T cycles. The default Quest VCPU scheduler chooses to run a thread from the highest priority VCPU with non-zero budget according to rate-monotonic scheduling theory. The budget of each VCPU is based initially on its capacity, C, and it is consumed while the VCPU is active on a physical CPU. VCPUs are resource containers, accounting for CPU usage by threads and system events such as interrupts, and providing the basis for making scheduling decisions. Quest implements each VCPU for conventional tasks (i.e., processes) as a Sporadic Server, ensuring that tasks which block and wake up at arbitrary times can be analyzed as though they are periodic processes. This ensures that a system comprising events at arbitrary times can still be managed predictably, and admission control can ensure the temporal isolation amongst a set of VCPUs. For interrupts, it would be excessive to handle them using sporadic servers, since interrupt handlers tend to be short-lived entities and they would fragment the budgets of their sporadic servers into too many fine-grained pieces. Instead, I developed a technique to schedule and account for interrupt processing using a single indivisible budget for an (IO) VCPU, calculated dynamically during system execution. This has led to Quest being the first system built entirely on a collection of bandwidth-preserving servers for predictable and efficient management of tasks and interrupts. The work has been published in IEEE RTAS and RTSS conferences.

In recent years, my work has focused on the use of virtualization techniques to build the Quest-V separation kernel. John Rushby defined a separation kernel in his seminal SOSP'81 paper as a collection of distributed components assigned to regimes, or virtual machines, which appear indistinguishable from separate private machines for each component. In Quest-V, each regime takes the form of a *sandbox*, which encapsulates a subset of machine physical resources: memory, I/O devices, and CPU cores or hardware threads. Each sandbox also defines a software domain, containing a set of services and application threads granted access to the sandbox machine resources. A Quest-V multicore system can combine Quest real-time sandboxes with legacy non-real-time Linux sandboxes, to implement a mixed-criticality system. For example, automotive systems feature mixed-criticality services, to manage low-criticality infotainment services (such as for navigation and audio) and high-criticality vehicle control (such as for stability control and anti-lock braking). While some high-end vehicle management systems feature more than a hundred embedded processors connected over a CAN bus network, multicore processors offer the opportunity to consolidate mixed-criticality tasks. The challenge is how to ensure low-criticality tasks do not affect those that must execute to ensure system correctness and safety.

Quest-V uses hardware virtualization techniques found on modern x86 multicore processors to partition sandboxes into secure domains. It enforces both *temporal* and *spatial* isolation of tasks executing within each sandbox. Quest-V is similar to other systems such as Muen, PikeOS, Mentor Graphics Hypervisor, NoHype, and Siemens Jailhouse, to name a few. However, it is the first separation kernel using hardware virtualization to realize a chip-level mixed-criticality system, which removes the hypervisor from normal execution. Quest-V does not rely on Xen or Linux, instead it is bootstrapped using monitor code that fits within a 4KB page of memory, making it suitable for embedded systems with limited memory. Additionally, all scheduling, interrupt and I/O management (including data transfers) are handled by sandbox code rather than a hypervisor. This takes the hypervisor out of the normal execution path, heightening security and increasing efficiency. Comparisons in our VEE 2014 paper show almost negligible virtualization costs compared to a standalone Linux system, and much better performance than Xen for process management. More recently, our RTSS 2014 paper shows how Quest-V supports predictable and efficient communication between sandboxes, enabling real-time migration of VCPUs and tasks for load balancing, fault recovery and resource affinity.

Quest-V additionally uses our hardware performance counter techniques developed in CAFÉ to partition shared caches on multicore processes. We have built on this to develop what we believe to be the first efficient dynamic page coloring technique for cache isolation on multicore processors. Our work on COLORIS, a color isolation framework has been published in PACT 2014.

For the future, my work will continue to build on Quest-V. My plans are to apply it to the management of smart devices for use in mixed-criticality applications, and also the Internet-of-Things. We are currently building a driverless vehicle platform to test the ability of Quest-V to support mission-critical tasks (e.g., motion control) while sensing, path planning and telemetry all operate in separate, potentially lower-criticality sandboxes. Real-time fault detection and recovery is one key topic under investigation with Quest-V. Similarly, the extension of Quest-V across networks encompassing Wi-Fi, Ethernet, USB, shared memory and other communication media will be investigated. One plan is to apply Quest-V to IoT applications such as for home automation, where the separation of data flows for different users (e.g., emergency operators versus family members) will need to be both secure and predictable.