Building Mixed Criticality Real-Time systems on seL4

...

Curtis Millar - Thesis A

Modern real-time systems

- Real-time cyber-physical systems are becoming ever more prevalent
  - Medical, aerospace, automotive
- Easier dedicate processing hardware for each real-time component for certification
- Consolidate real-time components onto fewer physical processors
- Use common off-the-shelf hardware components for real-time systems
- Minimise verification cost with verified isolation of components
  - Isolate high-criticality components from low-criticality
  - Only need to verify high-criticality and shared components

With seL4, we can lower the cost of building trustworthy mixed-criticality real-time systems

Real-time Systems

What is a real-time system?

- Functional requirements that have strict dependencies on timing of external events
  - Control the state of a system continuously / with high temporal resolution
  - Respond to events within time bound
- Hard real-time components must *always* satisfy their timing requirements
  - May cause a system to become inconsistent or unrecoverable
- Soft real-time components may be able to recover or may only have reduced value when the fail to satisfy timing requirements

How are real-time systems modelled?

- A set of *tasks* which are an (often infinite) set of *jobs*
- A *job* is a single instance of work in response to an event or as the result of a periodic *release*
  - The *release time* is the instant after which a job may be selected for execution
  - The *deadline* is the instant before which a job must complete execution
  - The *execution time* is the amount of processor time a job will consume for complete execution
  - These properties are often not know in advance
- A task often encapsulates necessary bounds of all jobs
  - The *period* or *minimum inter-release time* is the smallest time between consecutive jobs in a task
  - The *relative deadline* is the duration after their *release time* all jobs in a task must complete
  - The *worst case execution time* (WCET) is the largest execution time of any job in a task

How are real-time tasks implemented?

- Jobs within a task do not generally execute concurrently
- Jobs within a task generally perform similar work and require the same resources & state
- Task can be implemented as an OS thread with jobs being activations by a scheduler
- Many tasks may be able to coexist in a protection domain (implemented as virtual address space and process context)

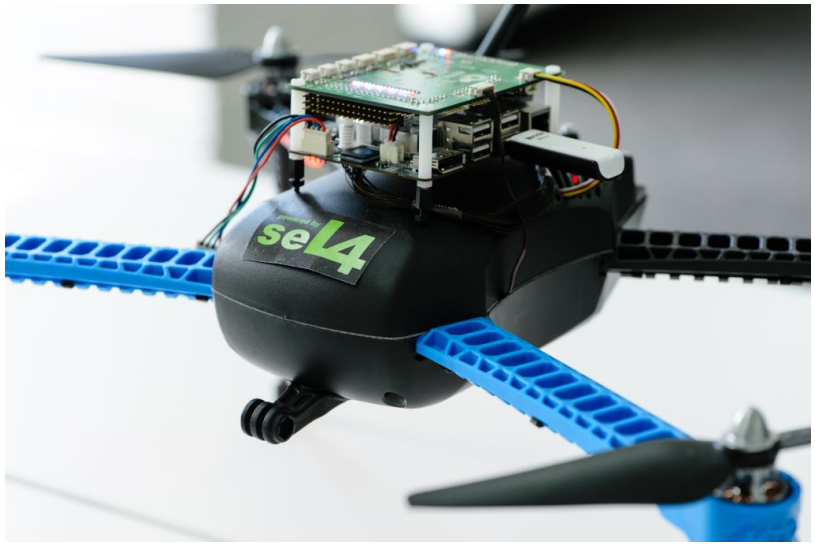
How do we know if a real-time system will work?

- Assume the worst:
  - A job's *release time* is always its task's *period* after the previous job
  - A job's *execution time* is its task's *worst case execution time*
- Choose a scheduling algorithm
- Perform an *admission test*
  - Will the chosen scheduling algorithm always select jobs such that every job will complete execution before its deadline?

Mixed criticality systems

- Not all tasks are equally important or *critical*
  - Failure of certain tasks may lead to loss of life or important assets (*safety critical*)
  - Failure of some tasks may lead to the system being unable to fulfil their purpose (*mission critical*)
  - Some tasks may be able to fail without affecting overall system outcomes (*non-critical*)
  - Most certification standards define a set of criticality levels with each task in a system being assigned to a particular one
- Criticality does not imply *priority*
  - Just because a task cannot be allowed to fail does not mean it must execute before less critical tasks
- Different criticalities must be *isolated* from each other
  - A task cannot be allowed to *interfere* with a task of higher criticality
    - Pre-empt; prevent or delay scheduling; invalidate shared resources

seL4



Strong isolation & trust boundaries

- seL4 is a microkernel the is highly effective for constructing systems of well-isolated components
- Enforces strong isolation between components
- Provides integrity for highly critical components
- Kernel itself is a small trusted component of the overall system
- Can construct systems with well contained *trusted computing base* (TCB)
    - Greatly reduced burden to guarantee correctness or demonstrate trustworthiness

seL4 for mixed criticality systems

- Extends seL4 to explicitly model allocation of time
  - Scheduling Contexts confer allocation of a proportion of processing time
- Guarantees maximum bandwidth on time allocations
- Modifies scheduling to enforce bandwidth
- Modifies IPC to preference high-priority threads
- Allows transfer of time allocation between threads with IPC
  - Client can donate SC to server, server execution is charged to client's SC
- Allows for scheduler configuration at user-level

How do we build mixed criticality systems with seL4?

Outline

- Analysis and benchmarking tools
- Root-level admissions control
- User-level schedulers
  - Analysis of user-level scheduling overheads
  - Analysis of changes to support different user-level scheduling techniques
- Soft real-time recovery
- Shared resource servers
  - Recovery of high-availability resource servers
- Real-world real time system

Analysis and benchmarking tools

- Trace all scheduling behavior
  - Thread activations
  - Interrupts & Preemption
  - Kernel utilisation
  - Temporal faults
  - Utilisation

- Test workloads
  - Schedulability / utilisation
  - Latency
  - Jitter
- Repeatable, random workload generation

Root-level admissions control

- Construct a reusable user-level component to perform global admissions test
- Ensure all SCs get a lower bound on execution time within bandwidth
- Rate-limit interrupts to bound preemption

User-level schedulers

- Build component-local real-time scheduler
- Internal task selection policy
- Internal admissions test (in addition to root test)
- Benchmark overhead of user-level task control and selection
  - Well behaved jobs
  - Misbehaving jobs
- Benchmark impact of any changes to kernel or Scheduling Contexts

Soft real-time recovery

- Tasks with jobs that may fail to meet deadlines
- Tasks that may fault or may be generally unreliable
- Roll forward / abandon job
- Reset to known good state
- Completely reload

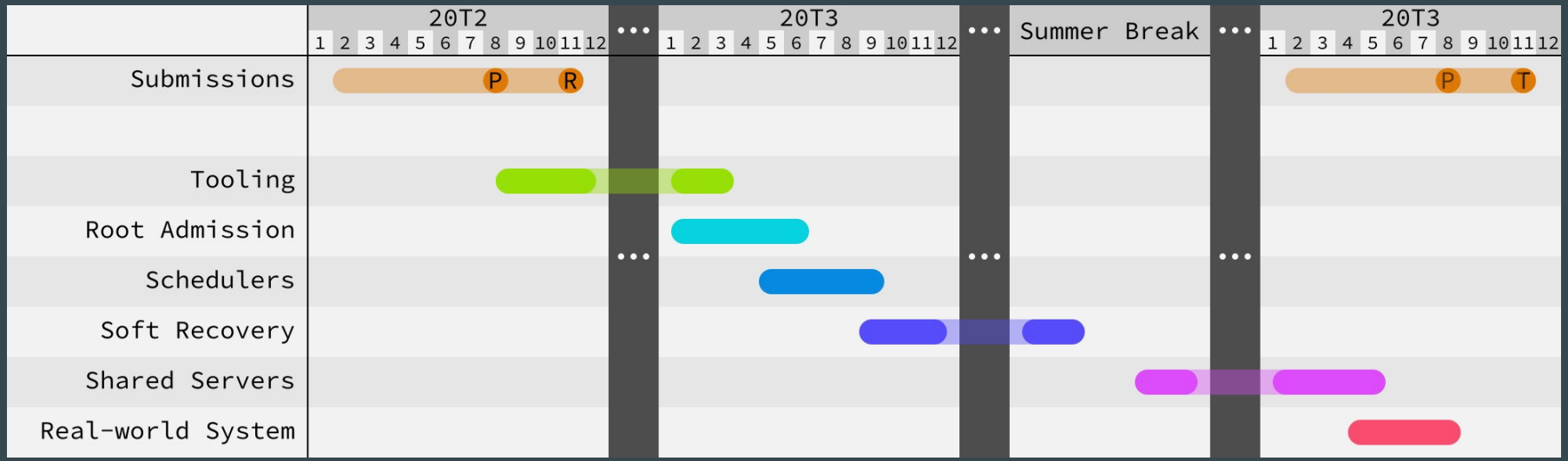
Shared resource servers

- Resource servers shared between tasks of differing criticalities
- May have high-priority low-criticality clients and low-priority high-criticality clients
- If a low-criticality task preempts access to a resource by a high-criticality task, we must still guarantee that the high-criticality task will be able to access the resource with sufficient time before its deadline
- How do we deal with a server responding to a client with insufficient budget?
    - Get the server to a consistent state
    - Guarantee time to recover the server
    - Ensure clients to wait until budget is sufficient or fault if budget is insufficient (need to know WCET of server anyway)

Real-world real time system

- Build a real cyber-physical system
- Mixed-criticality real-time components
- Resources shared between mixed-criticality clients
- Potential candidate: Quadcopter with networked media streaming
  - Flight control is high-criticality but low-priority
  - Media streaming is low-criticality but should be low-latency (high-priority)
  - Shared telemetry resources

Timeline


20T2
1 2 3 4 5 6 7 8 9 10 11 12
20T3
1 2 3 4 5 6 7 8 9 10 11 12
Summer Break
20T3
1 2 3 4 5 6 7 8 9 10 11 12
Submissions
P
R
P
T
Tooling
Root Admission
Schedulers
Soft Recovery
Shared Servers
Real-world System

Questions?

References

- Burns, Alan and Robert I. Davis (2019). Mixed Criticality Systems - A Review (Twelfth Edition) .
- Klein, Gerwin et al. (Oct. 2009). 'seL4: Formal Verification of an OS Kernel'. In: ACM Symposium on Operating Systems Principles . Big Sky, MT, USA: ACM, pp. 207–220.
- Liu, Jane W. S (2000). Real-Time systems . eng. Upper Saddle River, NJ.
- Lyons, Anna (2018). 'Mixed-Criticality Scheduling and Resource Sharing for High-Assurance Operating Systems'. Available from publications page at http://ts.data61.csiro.au/. PhD thesis. UNSW.