Motivation: Consolidation/Energy Efficiency

- Consider current approach with ECUs in cars:
  - Today: 50-100 individual “Engine Control Units”
  - Trend: Consolidate into smaller number of processors
  - How to provide guarantees?
- Better coordination for hard realtime/streaming media
  - Save energy rather than “throwing hardware at it”

Recall: Non-Real-Time Scheduling

- Primary Goal: maximize performance
- Secondary Goal: ensure fairness
- Typical metrics:
  - Minimize response time
  - Maximize throughput
  - E.g., FCFS (First-Come-First-Served), RR (Round-Robin)
Characteristics of a RTS

- Extreme reliability and safety
  - Embedded systems typically control the environment in which they operate
  - Failure to control can result in loss of life, damage to environment or economic loss

- Guaranteed response times
  - We need to be able to predict with confidence the worst case response times for systems
  - Efficiency is important but predictability is essential
    - In RTS, performance guarantees are:
      - Task- and/or class centric
      - Often ensured a priori
    - In conventional systems, performance is:
      - System oriented and often throughput oriented
      - Post-processing (… wait and see …)

- Soft Real-Time
  - Attempt to meet deadlines with high probability
  - Important for multimedia applications

Terminology

- Scheduling:
  - Define a policy of how to order tasks such that a metric is maximized/minimized
  - Real-time: guarantee hard deadlines, minimize the number of missed deadlines, minimize lateness

- Dispatching:
  - Carry out the execution according to the schedule
  - Preemption, context switching, monitoring, etc.

- Admission Control:
  - Filter tasks coming into the system and thereby make sure the admitted workload is manageable

- Allocation:
  - Designate tasks to CPUs and (possibly) nodes. Precedes scheduling

Typical Realtime Workload Characteristics

- Tasks are preemptable, independent with arbitrary arrival (=release) times
- Times have deadlines (D) and known computation times (C)
- Tasks execute on a uniprocessor system
- Example Setup:

Example:
Non-preemptive FCFS Scheduling
Example: Round-Robin Scheduling

Real-Time Scheduling

- Primary goal: ensure predictability
- Secondary goal: ensure predictability
- Typical metrics:
  - Guarantee miss ratio = 0 (hard real-time)
  - Guarantee Probability(missed deadline) < X% (firm real-time)
  - Minimize miss ratio / maximize completion ratio (firm real-time)
  - Minimize overall tardiness; maximize overall usefulness (soft real-time)
- E.g., EDF (Earliest Deadline First), LLF (Least Laxity First), RMS (Rate-Monotonic Scheduling), DM (Deadline Monotonic Scheduling)
- Real-time is about enforcing predictability, and does not equal to fast computing!!!

Scheduling: Problem Space

- Uni-processor / multiprocessor / distributed system
- Periodic / sporadic / aperiodic tasks
- Independent / interdependant tasks
- Preemptive / non-preemptive
- Tick scheduling / event-driven scheduling
- Static (at design time) / dynamic (at run-time)
- Off-line (pre-computed schedule), on-line (scheduling decision at runtime)
- Handle transient overloads
- Support Fault tolerance

Task Assignment and Scheduling

- Cyclic executive scheduling (⇒ later)
- Cooperative scheduling
  - scheduler relies on the current process to give up the CPU before it can start the execution of another process
- A static priority-driven scheduler can preempt the current process to start a new process. Priorities are set pre-execution
  - E.g., Rate-monotonic scheduling (RMS), Deadline Monotonic scheduling (DM)
- A dynamic priority-driven scheduler can assign, and possibly also redefine, process priorities at run-time.
  - Earliest Deadline First (EDF), Least Laxity First (LLF)
**Simple Process Model**

- Fixed set of processes (tasks)
- Processes are periodic, with known periods
- Processes are independent of each other
- System overheads, context switches etc, are ignored (zero cost)
- Processes have a deadline equal to their period
  - i.e., each process must complete before its next release
- Processes have fixed worst-case execution time (WCET)

**Performance Metrics**

- Completion ratio / miss ration
- Maximize total usefulness value (weighted sum)
- Maximize value of a task
- Minimize lateness
- Minimize error (imprecise tasks)
- Feasibility (all tasks meet their deadlines)

**Scheduling Approaches (Hard RTS)**

- Off-line scheduling / analysis (static analysis + static scheduling)
  - All tasks, times and priorities given a priori (before system startup)
  - Time-driven; schedule computed and hardcoded (before system startup)
  - E.g., Cyclic Executives
  - Inflexible
  - May be combined with static or dynamic scheduling approaches
- Fixed priority scheduling (static analysis + dynamic scheduling)
  - All tasks, times and priorities given a priori (before system startup)
  - Priority-driven, dynamic(!) scheduling
    - The schedule is constructed by the OS scheduler at run time
  - For hard / safety critical systems
  - E.g., RMA/RMS (Rate Monotonic Analysis / Rate Monotonic Scheduling)
- Dynamic priority scheduling
  - Tasks times may or may not be known
  - Assigns priorities based on the current state of the system
  - For hard / best effort systems
  - E.g., Least Completion Time (LCT), Earliest Deadline First (EDF), Least Slack Time (LST)

**Cyclic Executive Approach**

- Clock-driven (time-driven) scheduling algorithm
- Off-line algorithm
- Minor Cycle (e.g. 25ms) gcd of all periods
- Major Cycle (e.g. 100ms) lcm of all periods

Construction of a cyclic executive is equivalent to bin packing

<table>
<thead>
<tr>
<th>Process</th>
<th>Period</th>
<th>Comp. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>25</td>
<td>8</td>
</tr>
<tr>
<td>C</td>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>50</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>100</td>
<td>2</td>
</tr>
</tbody>
</table>
Cyclic Executive (cont.)

```
loop
  Wait_For_Interrupt;
  Procedure_For_A;
  Procedure_For_B;
  Procedure_For_C;
  Wait_For_Interrupt;
  Procedure_For_A;
  Procedure_For_B;
  Procedure_For_C;
  Wait_For_Interrupt;
  Procedure_For_A;
  Procedure_For_B;
  Procedure_For_D;
  Procedure_For_E;
end loop;
```

Cyclic Executive: Observations

- No actual processes exist at run-time
  - Each minor cycle is just a sequence of procedure calls
- The procedures share a common address space and can thus pass data between themselves.
  - This data does not need to be protected (via semaphores, mutexes, for example) because concurrent access is not possible
- All ‘task’ periods must be a multiple of the minor cycle time

Cyclic Executive: Disadvantages

- With the approach it is difficult to:
  - incorporate sporadic processes;
  - incorporate processes with long periods;
    - Major cycle time is the maximum period that can be accommodated without secondary schedules (=procedure in major cycle that will call a secondary procedure every N major cycles)
  - construct the cyclic executive, and
  - handle processes with sizeable computation times.
    - Any ‘task’ with a sizeable computation time will need to be split into a fixed number of fixed sized procedures.

Online Scheduling for Realtime
**Schedulability Test**

- Test to determine whether a feasible schedule exists
- **Sufficient Test**
  - If test is passed, then tasks are definitely schedulable
  - If test is not passed, tasks may be schedulable, but not necessarily
- **Necessary Test**
  - If test is passed, tasks may be schedulable, but not necessarily
  - If test is not passed, tasks are definitely not schedulable
- **Exact Test (= Necessary + Sufficient)**
  - The task set is schedulable if and only if it passes the test.

**Rate Monotonic Analysis: Assumptions**

- **A1**: Tasks are periodic (activated at a constant rate).
  - Period $P_i$ = Interval between two consecutive activations of task $T_i$
- **A2**: All instances of a periodic task $T_i$ have the same computation time $C_i$
- **A3**: All instances of a periodic task $T_i$ have the same relative deadline, which is equal to the period ($D_i = P_i$)
- **A4**: All tasks are independent (i.e., no precedence constraints and no resource constraints)

Implicit assumptions:
- **A5**: Tasks are preemptable
- **A6**: No task can suspend itself
- **A7**: All tasks are released as soon as they arrive
- **A8**: All overhead in the kernel is assumed to be zero (or part of $i + iT$)

**Rate Monotonic Scheduling: Principle**

- **Principle**: Each process is assigned a (unique) priority based on its period (rate); always execute active job with highest priority
- The shorter the period the higher the priority
  - $P_i < P_j \Rightarrow \pi_i > \pi_j$ (1 = low priority)
- W.l.o.g. number the tasks in reverse order of priority

**Example: Rate Monotonic Scheduling**

- **Example instance**

<table>
<thead>
<tr>
<th>Process</th>
<th>Period</th>
<th>Priority</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>25</td>
<td>5</td>
<td>T1</td>
</tr>
<tr>
<td>B</td>
<td>60</td>
<td>3</td>
<td>T3</td>
</tr>
<tr>
<td>C</td>
<td>42</td>
<td>4</td>
<td>T2</td>
</tr>
<tr>
<td>D</td>
<td>105</td>
<td>1</td>
<td>T5</td>
</tr>
<tr>
<td>E</td>
<td>75</td>
<td>2</td>
<td>T4</td>
</tr>
</tbody>
</table>

- **RMA - Gant chart**
Example: Rate Monotonic Scheduling

Deadline Miss

\[ T_i = (p_i, c_i) \]

\[ P_i = \text{period}, \quad C_i = \text{processing time} \]

Utilization

\[ U_i = \frac{C_i}{P_i} \]

Utilization of task \( T_i \)

Example: \( U_2 = \frac{2}{5} = 0.4 \)

RMS: Schedulability Test

Theorem (Utilization-based Schedulability Test):

A periodic task set \( T_1, T_2, \ldots, T_n \) with \( D_i = P_i, \quad 1 \leq i \leq n \), is schedulable by the rate monotonic scheduling algorithm if

\[
\sum_{i=1}^{n} \left( \frac{C_i}{P_i} \right) \leq n(2^{1/n} - 1), \quad n = 1, 2, \ldots
\]

This schedulability test is “sufficient”!

- For harmonic periods (evenly divides), the utilization bound is 100%
- \( n(2^{1/n} - 1) \to \ln 2 \) for \( n \to \infty \)

RMS Example

- \( T_1 = (4,1), \quad T_2 = (5,2), \quad T_3 = (7,2) \)
- \( \frac{C_1}{P_1} = 1/4 = 0.25, \quad \frac{C_2}{P_2} = 2/5 = 0.4, \quad \frac{C_3}{P_3} = 2/7 \approx 0.286 \)
- The schedulability test requires

\[
\sum_{i=1}^{n} \left( \frac{C_i}{P_i} \right) \leq n(2^{1/n} - 1), \quad n = 1, 2, \ldots
\]

- Hence, we get

\[
\sum_{i=1}^{3} \left( \frac{C_i}{P_i} \right) \approx 0.936 > 3(2^{1/3} - 1) \approx 0.780
\]
EDF: Assumptions

A1: Tasks are periodic or aperiodic.
   Period $P_i$ = Interval between two consecutive activations of task $T_i$
A2: All instances of a periodic task $T_i$ have the same computation time $C_i$
A3: All instances of a periodic task $T_i$ have the same relative deadline, which is equal to the period ($D_i = P_i$)
A4: All tasks are independent (i.e., no precedence constraints and no resource constraints)

Implicit assumptions:
A5: Tasks are preemptable
A6: No task can suspend itself
A7: All tasks are released as soon as they arrive
A8: All overhead in the kernel is assumed to be zero (or part of $C_i$)

EDF Scheduling: Principle

- Preemptive priority-based dynamic scheduling
- Each task is assigned a (current) priority based on how close the absolute deadline is.
- The scheduler always schedules the active task with the closest absolute deadline.

EDF Schedulability Test

Theorem (Utilization-based Schedulability Test):
A task set $T_1, T_2, ..., T_n$ with $D_i = P_i$ is schedulable by the earliest deadline first (EDF) scheduling algorithm if

$$\sum_{i=1}^{n} \left( \frac{C_i}{D_i} \right) < 1$$

Exact schedulability test (necessary + sufficient)
Proof: [Liu and Layland, 1973]

EDF Optimality

EDF Properties
- EDF is optimal with respect to feasibility (i.e., schedulability)
- EDF is optimal with respect to minimizing the maximum lateness
EDF Example: Domino Effect

EDF minimizes lateness of the “most tardy task” [Dertouzos, 1974]

Constant Bandwidth Server

- Intuition: give fixed share of CPU to certain of jobs
  - Good for tasks with probabilistic resource requirements
- Basic approach: Slots (called “servers”) scheduled with EDF, rather than jobs
  - CBS Server defined by two parameters: Qs and Ts
  - Mechanism for tracking processor usage so that no more than Qs CPU seconds used every Ts seconds (or whatever measurement you like) when there is demand. Otherwise get to use processor as you like
- Since using EDF, can mix hard-realtime and soft realtime:

Comparison of CBS with EDF in Overload

- If scheduled items do not meet EDF schedulability:
  - EDF yields unpredictable results when overload starts and stops
  - CBS servers provide better isolation
- Isolation particularly important when deadlines crucial
  - I.e. hard real-time tasks

CBS on multiprocessors

- Basic problem: EDF not all that efficient on multiprocessors.
  - Schedulability constraint considerably less good than for uniprocessors. Need:
    \[
    U(\tau_{k+1}) \leq \frac{U}{1 - U_k}
    \]
- Key idea of paper: send highest-utilization jobs to specific processors, use EDF for rest
  - Minimizes number of processors required
  - New acceptance test:
    \[
    m \geq \min_{k=1}^{n} \left\{ (k-1) + \frac{U(\tau_{k+1})}{1 - U_k} \right\}
    \]
Are these good papers?

- What were the authors’ goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the “Test of Time” challenge?
- How would you review this paper today?

Going Further: Guaranteeing Resources

- What might we want to guarantee?
  - Examples:
    » Guarantees of BW (say data committed to Cloud Storage)
    » Guarantees of Requests/Unit time (DB service)
    » Guarantees of Latency to Response (Deadline scheduling)
    » Guarantees of maximum time to Durability in cloud
    » Guarantees of total energy/battery power available to Cell
  - What level of guarantee?
    - With high confidence (specified), Minimum deviation, etc.
  - What does it mean to have guaranteed resources?
    - A Service Level Agreement (SLA)?
    - Something else?
  - “Impedance-mismatch” problem
    - The SLA guarantees properties that programmer/user wants
    - The resources required to satisfy SLA are not things that programmer/user really understands

Resource Allocation must be Adaptive

- Properties of a Cell
  - A user-level software component with guaranteed resources
  - Has full control over resources it owns (“Bare Metal”)
  - Contains at least one memory protection domain (possibly more)
  - Contains a set of secured channel endpoints to other Cells
  - Contains a security context which may protect and decrypt information
- When mapped to the hardware, a cell gets:
  - Gang-schedule hardware thread resources (“Harts”)
  - Guaranteed fractions of other physical resources
    » Physical Pages (DRAM), Cache partitions, memory bandwidth, power
  - Guaranteed fractions of system services
- Predictability of performance
  - Ability to model performance vs resources
  - Ability for user-level schedulers to better provide QoS

Resource-Level Abstraction: the Cell
Implementing Cells: Space-Time Partitioning

- Spatial Partition: Performance isolation
  - Each partition receives a vector of basic resources
    » A number HW threads
    » Chunk of physical memory
    » A portion of shared cache
    » A fraction of memory BW
    » Shared fractions of services
- Use of hardware partitioning mechanisms
- Partitioning varies over time
  - Fine-grained multiplexing and guarantee of resources
    » Resources are gang-scheduled
- Controlled multiplexing, not uncontrolled virtualization
- Partitioning adapted to the system’s needs
- Modified CBS—use of timing to provide gang scheduling

Two Level Scheduling: Control vs Data Plane

- Split monolithic scheduling into two pieces:
  - Course-Grained Resource Allocation and Distribution to Cells
    » Chunks of resources (CPUs, Memory Bandwidth, QoS to Services)
    » Ultimately a hierarchical process negotiated with service providers
  - Fine-Grained (User-Level) Application-Specific Scheduling
    » Applications allowed to utilize their resources in any way they see fit
    » Performance Isolation: Other components of the system cannot interfere with Cells use of resources

Example: Convex allocation (PACORA)

Convex Optimization with Online Application Performance Models

- Continuously minimize the penalty of the system
  (subject to restrictions on the total amount of resources)

Example: Feedback allocation

- Utilize dynamic control loops to fine-tune resources
- Example: Video Player interaction with Network
  - Server or GUI changes between high and low bit rate
  - Goal: set guaranteed network rate:

- Alternative: Application Driven Policy
  - Static models
  - Let network choose when to decrease allocation
  - Application-informed metrics such as needed BW
Architecture of Tessellation OS

Framework for User-Level Runtimes

PULSE: Preemptive User-Level Schedulers

Frameworks for User-Level Runtimes

PULSE: Preemptive User-Level Schedulers

- Framework Components
  - Hardware threads (harts)
  - Timer Interrupts for preemption

- Able to handle cell resizing
  - Automatic simulation of suspended scheduler segments

- Available schedulers
  - Round-robin (and pthreads)
  - EDF and Fixed Priority
  - Constant Bandwidth Server (M-CBS)
  - Juggle: load balancing SPMD apps

- Other framework: Lithe
  - Non-preemptive, Cooperative
  - Allows construction of schedulers that cooperate with libraries in handling processor resources.

Example: GUI Service

- Provides differentiated service to applications
  - Soft service-time guarantees
  - Performance isolation from other applications

- Operates on user-meaningful “actions”
  - E.g. “draw frame”, “move window”

- Exploits task parallelism for improved service times

Performance of GUI Service

Out of 4000 frames
Conclusion

• **Constant Bandwidth Server (CBS)**
  – Provide a compatible scheduling framework for both hard and soft real-time applications
  – Targeted at supporting Continuous Media applications

• **Multiprocessor CBS (M-CBS)**
  – Increase overall bandwidth of CBS by increasing processor resources

• **Resource-Centric Computing**
  – Centered around resources
  – Separate allocation of resources from use of resources
  – Next Time: Resource Allocation and Scheduling Frameworks