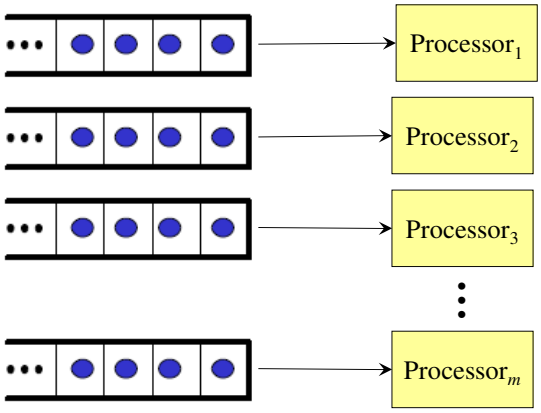
Task Splitting on Multiprocessors

Raj Rajkumar

Lecture #14

Partitioned Scheduling for Multiprocessors

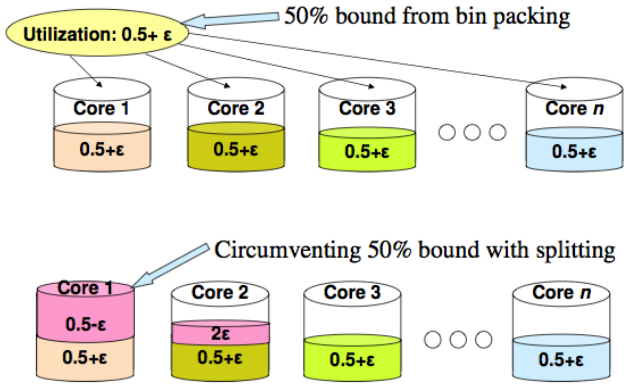
One Ready Queue Per Processor



Remember: The Bad Case of Bin Packing

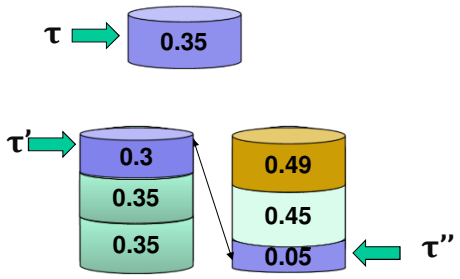
- Given a list of objects of size 0.51, 0.51, 0.51, 0.51, 0.51
- Pack these objects into bins of capacity 1.0
- How many bins are required?
- With objects of size 0.50+ε, 0.50+ε, 0.50+ε, 0.50+ε, 0.50+ε,
  - What is the utilization of the system?
- Large objects can be a problem!
  - Think *rocks*…

Task Splitting



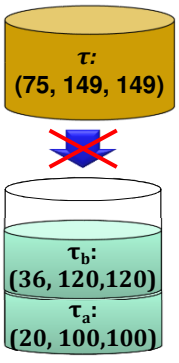
Splitting: Illustration

- Keep allocating tasks to a core (M_p) if they "fit"
- If a task does not fit
  - Split the task **τ** into **τ' and τ''**
  - Assign **τ' to** M_p and assign remainder **τ''** to next core (M_{p+1})

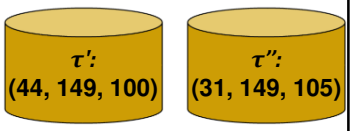


How to split a task?

- If task τ (C, T, D) does not fit into core M_p
  - And M_p is non-full
  - Split task τ into τ' and τ'' such that
    - τ' has C' units of execution on core M_p
    - τ'' has (C-C') units of execution on core M_{p+1}
    - Let R' be the response time of task τ'
- Deadline of τ' = deadline of the highest priority task
- τ' given the highest priority
- τ'' should start after R' but finish by original D
- *τ'' will be the highest priority task on M_{p+1}*



Will complete at 44



Design Considerations

- When to split tasks?
  - Scenarios with "chunky" tasks
    - Too big to fit in one core
    - Too much utilization wasted in core otherwise
  - Task splitting is done to further improve efficiency
  - Tasks with soft deadlines are ideal to split
- Task splitting reduces processor count
  - Leads to savings in cost
  - Reduction in baseline power, and
  - Savings in physical space

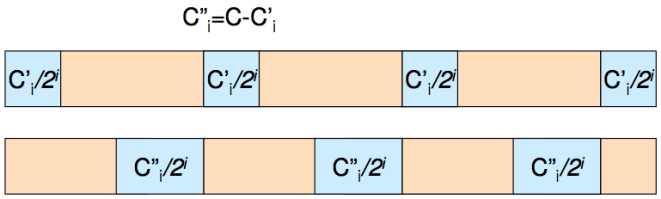
Period Transformation

- A technique to reconcile the scheduling priority and semantic importance of a task.
- **Recall**: In case of an overload, only the highest priority tasks can meet their deadlines under fixed-priority preemptive scheduling.
- A task (*C, T*) can be transformed into a task (C/2, T/2) i.e. with two virtual pieces, the first piece executing the first half of the task and the second piece executing the second piece of the task
  - With a shorter period, the task can be assigned a higher priority
- A task (*C, T*) can be transformed into *a task* (*C/k*, *T/k*) with *k* virtual pieces
  - The task has a much shorter period, and can be assigned a higher priority

Task Splitting for Harmonics

- Consider tasks,
- Assume **period transformation** of split tasks
- How to split $\frac{C_i}{2^i T}$ among processors P_1 and P_2
- Periodic servers $(C'_i/2^i, T)$ and $(C''_i/2^i, T)$:

$$\frac{C_0}{2^0 T}, \frac{C_1}{2^1 T}, \frac{C_2}{2^2 T}, \ldots, \frac{C_n}{2^n T}$$



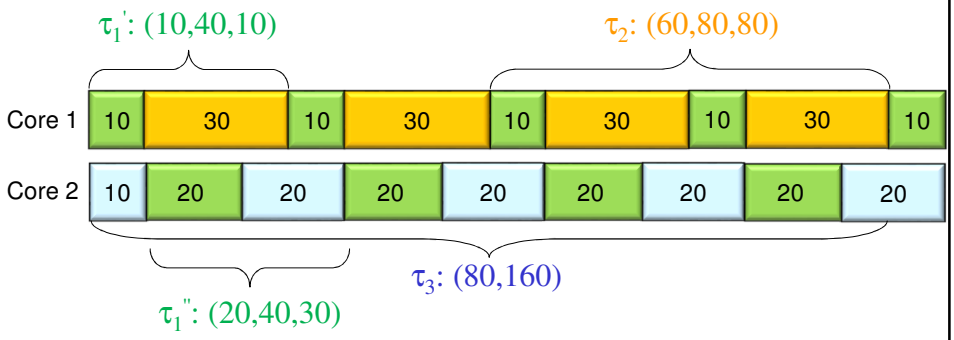
Example A (1 of 2)

- Consider the task set τ_1: (30, 40), τ_2: (60, 80), τ_3: (80, 160).
  - The utilization values are 0.75, 0.75 and 0.5 for a total utilization of 2.0.
  - But, no two tasks fit together → need 3 processors!
- Use BF**D**:
  - Per RMS, priority sequence: τ_1, τ_2 and τ_3
  - These are harmonic tasks: hence, schedulability bound = 100% for RMS.
  - Bin-packing sequence: τ_1, τ_2 and τ_3.
  1. τ_1 → core 1
  2. τ_2 → cannot fit into core 1. So, we decide to split.
  - Split which task? Task τ_1 has the highest priority on core 1. So, we split task τ_1.
    - $U(\tau_1') = 0.25$ (1.0 – 0.75 of τ_2) for schedulability
    - $C_1' = 0.25 * T_1 = 0.25 * 40 = 10$
    - $D_1' = C_1' = 10$
    - $C_1'' = C_1 - C_1' = 30 - 10 = 20$
    - $D_1'' = D_1 - D_1' = 40 - 10 = 30$.

  3. τ_1'' (20, 40, 30) → core 2.
  4. task τ_3 → core 2.

Example A (2 of 2)

- Core 1: τ_1': (10, 40, 10), τ_2: (60, 80, 80).
  - Is this taskset schedulable? **Yes.**
- Core 2: τ_1'': (20, 40, 30), τ_3: (80, 160, 160).
  - Is this taskset schedulable?
  - Yes!



Example B (1 of 2)

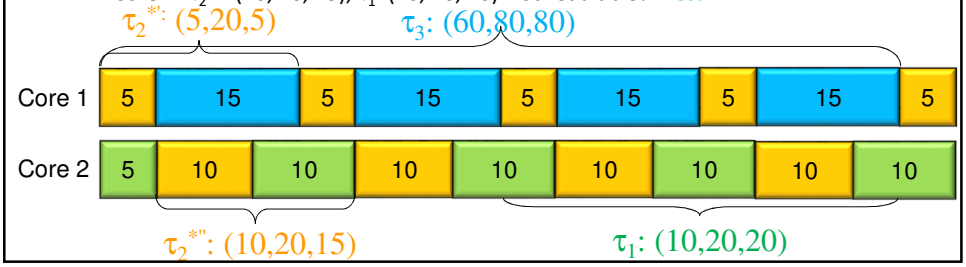
- Consider the task set τ_1: (10, 20), τ_2: (30, 40), τ_3: (60, 80).
  - The utilization values are 0.5, 0.75 and 0.75 for a total utilization of 2.0.
  - Again, no two tasks fit together.
- Use BF**D**:
  - Per RMS, priority sequence: τ_1, τ_2 and τ_3
  - These are harmonic tasks: hence, schedulability bound = 100% for RMS.
  - Bin-packing sequence: τ_2, τ_3 and τ_1.

1. τ_2 → core 1
2. τ_3 → cannot fit into core 1. So, split! Which task? Task τ_2 has the highest priority on core 1. So, we split task τ_2.
   - $U(\tau_2')$ = 0.25 (1.0 – 0.75 of τ_3) for schedulability
   - $C_2' = 0.25 * T_2 = 0.25 * 40 = 10$
   - $D_2' = C_2' = 10$
   - $C_2'' = C_2 - C_2' = 30 - 10 = 20$
   - $D_2'' = D_2 - D_2' = 40 - 10 = 30$.
3. τ_2'' (20, 40, 30) → core 2.
4. task τ_1 → core 2.

Example B (2 of 2)

- Core 1: τ_2': (10, 40, 10), τ_3: (60, 80, 80).
  - Is this taskset schedulable? **Yes.**
- Core 2: τ_1: (10, 20, 20), τ_2'': (20, 40, 30).
  - Is this taskset schedulable?
  - No! Oops – our allocation did not fit – we couldn't do that!
- Solution: Use period transformation to make τ_2'' the highest priority task on core 2?
  - Easier to transform task τ_2 so that it is the highest priority task in the taskset.
- Original taskset becomes τ_1: (10, 20), τ_2^*: (15, 20), τ_3: (60, 80).
- *Now*, use task splitting.
- Core 1: $\tau_2^{*'}$: (5, 20, 5), τ_3: (60, 80, 80). Schedulable? Yes.
- Core 2: $\tau_2^{*''}$: (10, 20, 15), τ_1: (10, 20, 20). Schedulable? **Yes!**

Break tie in favor of task τ_2^*



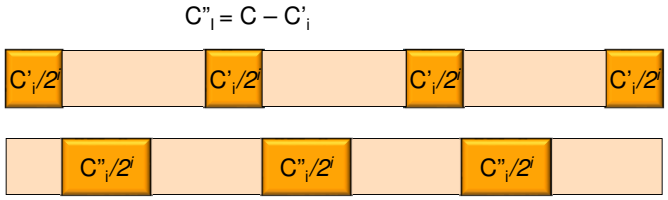
Takeaways from Examples A and B

- When we have to split a task, pick the highest priority task on that bin to make sure its $C' = D'$.
- The second split piece has a shorter deadline ($D'' = D - D'$). By using period transformation, we make this second piece be the highest priority task on the other bin as well.
- If you period-transform a split piece, might as well transform the *entire* task.

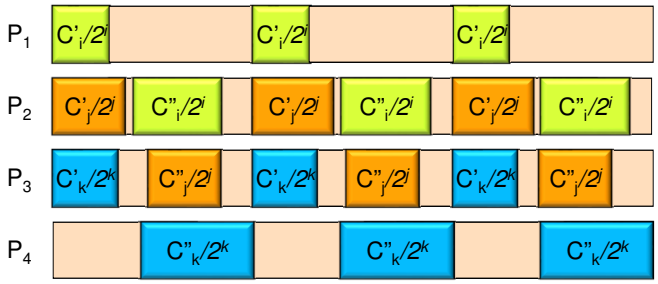
Task Splitting For Harmonics

- Consider harmonic tasks
- Assume **period transformation** of split tasks
- How to split $\frac{C_i}{2^i T}$ among processors P_1 and P_2
- Periodic servers ($C'_i/2^i$, T) and ($C''_i/2^i$, T) :

$$\frac{C_0}{2^0 T}, \frac{C_1}{2^1 T}, \frac{C_2}{2^2 T}, \ldots, \frac{C_n}{2^n T}$$



Analysis



- Achieves 100% utilization with harmonic tasksets
  - General algorithm also applies to other *harmonic* task sets
- Migration costs:
  - total migrations in the worst-case (per T_n)

$$O(m\frac{T_n}{T_0})$$

Conclusions

- With task splitting, the bad case of bin packing (even for the optimal bin-packing algorithm) is avoided.
- With period transformation, the scheduling priority of a task can be reconciled with the semantic importance of a task.
- By combining period transformation and task splitting, 100% schedulable utilization of each of *m* processors can be obtained for harmonic task sets.
  - Can be applied to non-harmonic tasksets too, but the schedulable utilization levels are lower (as in RMS for non-harmonic tasksets).

Priority Granularity

Number of Priority Levels Needed

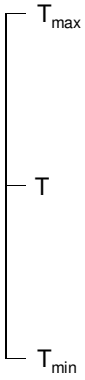
- Suppose we have a taskset with 1000 unique periods (or deadlines)
- Do we need 1000 priority levels to represent each of them uniquely?
- What about 10000 tasks?

Priority Spectrum Analysis

- Suppose we have one priority bit. All tasks must be either marked as high or low priority.
  - Consider period spectrum between T_{min} and T_{max}
  - Periods above T are marked low priority and periods below T are marked high priority
- Priority inversion is maximum when a task with period (T_{min} + ε) shares the same priority with a task of period ($T - \varepsilon$).
- We want $(T - \varepsilon) / (T_{min} + \varepsilon) = (T_{max} - \varepsilon) / (T + \varepsilon)$
- As $\varepsilon \rightarrow 0$, $T/T_{min} = T_{max}/T = 1/r$

$\rightarrow T^2 = T_{min} T_{max}$

i.e. T_{min}, T and T_{max} form a *geometric progression*.

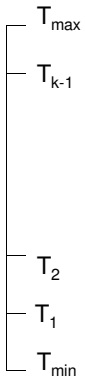


For *k* priority levels

- Generalizing this notion, for *k* priority levels, we have

$$T_1/T_{min} = T_2/T_1 = T_3/T_2 = \ldots = T_{max}/T_{k-1} = r$$

- That is, the period boundaries for priority assignment form a geometric progression.



Schedulability Bound with finite '*k*'

- The worst-case scheduling bound with insufficient priority levels is given (approximately) by*

$$U = \begin{cases} MIN_{1 \le i \le K}(1 - G_i + \ln(2G_i)) & 1/2 \le G_i \le 1 \\ G_i & 0 \le G_i \le 1/2 \end{cases}$$

where $G_i = (L_{i-1} + 1)/L_i$

Where, L_i corresponds to the discrete boundary of each priority "grid".

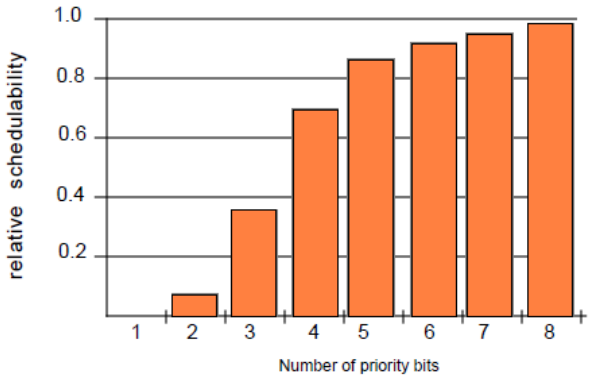
If a task's period falls between two grid bound boundaries such that $L_{j-1} < T \le L_j$, priority *j* will be assigned.

*John Lehoczky and Lui Sha, "Performance of real-time bus scheduling algorithms", Joint International Conference on Measurement and Modeling of Computer Systems, Proceedings of the *1986 ACM SIGMETRICS joint international conference on Computer Performance Modeling, Measurement and Evaluation*, 1986.

Revisiting *k* needed

- Suppose we pick T_{min} = 100µs and T_{max} = 100s, and we have *k* priority levels
- $T_{max} / T_{min} = 10^6 = r^k$

Relative Schedulability



Conclusions

- A small number of priority bits (8) is very close to having an infinite number of priority bits in practice.