Midterm Review #2
Consider a system of two tasks:

Task 1: \( P_1 = 1.7, \ D_1 = 0.5, \ C_1 = 0.5 \)

Task 2: \( P_2 = 8, \ D_2 = 3.2, \ C_2 = 2 \)

\[
I = \sum_j \left[ \frac{R_i}{P_j} \right] C_j
\]

\[
R_i = I + C_i
\]
Consider a system of two tasks:

Task 1: \( P_1 = 1.7, \ D_1 = 0.5, \ C_1 = 0.5 \)
Task 2: \( P_2 = 8, \ D_2 = 3.2, \ C_2 = 2 \)

\[
I = \sum_j \left[ \frac{R_i}{P_j} \right] C_j
\]

\[
R_i = I + C_i
\]

\[
I^{(0)} = C_1 = 0.5
\]

\[
R_2^{(0)} = I^{(0)} + C_2 = 2.5
\]

\[
I^{(1)} = \left[ \frac{R_2^{(0)}}{P_1} \right] C_1 = \left[ \frac{2.5}{1.7} \right] 0.5 = 1
\]

\[
R_2^{(1)} = I^{(1)} + C_2 = 3
\]

\[
I^{(2)} = \left[ \frac{R_2^{(1)}}{P_1} \right] C_1 = \left[ \frac{3}{1.7} \right] 0.5 = 1
\]

\[
R_2^{(2)} = I^{(2)} + C_2 = 3
\]

\[
3 < 3.2 \rightarrow \text{Ok!}
\]
Mixed Periodic and Aperiodic Task Systems

- Question: how to execute aperiodic tasks without violating schedulability guarantees given to periodic tasks?
Polling Server

- Runs as a periodic task (priority set according to RM)
- Aperiodic arrivals are queued until the server task is invoked
- When the server is invoked it serves the queue until it is empty or until the budget expires then suspends itself
  - If the queue is empty when the server is invoked it suspends itself immediately.
- Server is treated as a regular periodic task in schedulability analysis
Example of a Polling Server

- **Polling server:**
  - Period $P_s = 5$
  - Budget $B_s = 2$
- **Periodic task**
  - $P = 4$
  - $C = 1.5$
- All aperiodic arrivals have $C=1$

---

Diagram showing periodic and aperiodic arrivals.
Deferrable Server

- Keeps the balance of the budget until the end of the period
- Example (continued)
Worst-Case Scenario

Exercise: Derive the utilization bound for a deferrable server plus one periodic task

\[ U_p \leq \ln \left( \frac{U_s + 2}{2U_s + 1} \right) \]
Priority Exchange Server

- Like the deferrable server, it keeps the budget until the end of server period
- Unlike the deferrable server the priority slips over time: When not used the priority is exchanged for that of the executing periodic task
Priority Exchange Server

Example

Aperiodic tasks

Priority Exchange Server

Periodic Tasks

\[ U_p \leq \ln\left(\frac{2}{U_s + 1}\right) \]
Sporadic Server

- Server is said to be *active* if it is in the *running* or *ready* queue, otherwise it is *idle*.
- When an aperiodic task comes and the budget is not zero, the server becomes active.
- Every time the server becomes *active*, say at $t_A$, it sets replenishment time one period into the future, $t_A + P_s$ (but does not decide on replenishment amount).
- When the server becomes idle, say at $t_I$, set replenishment amount to capacity consumed in $[t_A, t_I]$

$$U_p \leq \ln\left(\frac{2}{U_s + 1}\right)$$
Slack Stealing Server

- Compute a slack function $A(t_s, t_f)$ that says how much total slack is available
- Admit aperiodic tasks while slack is not exceeded
Power of Computation

- **Terminology**
  - $R$ : Power spent on computation
  - $V$ : Processor voltage
  - $f$ : Processor clock frequency
  - $R_0$ : Leakage power

- **Power spent on computation is:**
  - $R = k_v V^2 f + R_0$
  - where $k_v$ is a constant
Energy of Computation

- Power spent on computation is:
  \[ R = k_v V^2 f + R_0 \]

- Consider a task of length \( C \) clock cycles and a processor operating at frequency \( f \)

- The execution time is \( t = C/f \)

- Energy spent is:
  \[ E = R \cdot t = (k_v V^2 f + R_0) \cdot (C/f) \]
Reducing Processor Frequency
Good or Bad?

- Does it make sense to operate the processor at a reduced speed to save energy? Why or why not?

  Possible Answer:

  \[ E = R \cdot t = (k_v \cdot V^2 \cdot f + R_0) \cdot (C/f) = k_v \cdot V^2 \cdot C + R_0 \cdot C/f \]

  - Conclusion: \( E \) is minimum when \( f \) is maximum.

    \[ \rightarrow \text{Operate at top speed} \]

- Is this really true? What are the underlying assumptions?
Dynamic Voltage Scaling (DVS): Reducing Voltage and Frequency

- Processor voltage can be decreased if clock frequency is decreased
  - Voltage and frequency can be decreased roughly proportionally.
  - In this case (where $V \sim f$):
    
    $$R = k_f f^3 + R_0$$
    $$E = (k_f f^3 + R_0)(C/ f) = k_f f^2 C + R_0 C/ f$$
Dynamic Voltage Scaling (DVS):
Reducing Voltage and Frequency

- Processor voltage can be decreased if clock frequency is decreased
  - Voltage and frequency can be decreased roughly proportionally.

\[ R = k_f f^3 + R_0 \]

\[ E = (k_f f^3 + R_0)(C/f) = k_f f^2 C + R_0 C/f \]

- Question: Does reducing frequency (and voltage) increase or decrease total energy spend on a task?
Dynamic Voltage Scaling (DVS): The Critical Frequency

- There exists a minimum frequency below which no energy savings are achieved

\[ E = k_f f^2 C + R_0 C/f \]
\[ \frac{dE}{df} = 2k_f f C - R_0 C/f^2 = 0 \]

\[ f = 3 \sqrt[3]{\frac{R_0}{2k_f}} \]
DVS Algorithm 1: Static Voltage Scaling

1. Calculate the critical frequency
2. Calculate the minimum frequency at which the task set remains schedulable
   - Example: If EDF is used and the utilization is 60% at the maximum frequency \( f_{\text{max}} \), then the frequency can be decreased to \( 0.6 \cdot f_{\text{max}} \).
3. Let \( f_{\text{opt}} \) be the larger of the above two
4. Operate the system at the smallest frequency at or above \( f_{\text{opt}} \).
DVS Algorithm 2: Cycle-conserving DVS

- What if a task finishes early?
  - Re-compute the utilization based on the reduced execution time.
  - Calculate the minimum frequency at which the task set is schedulable using the new utilization.
  - Update task execution times to the WCET when new invocations are released.
In the preceding discussion, we assumed that task execution time at frequency $f$ is $C/f$, where $C$ is the total cycles needed.

In reality some cycles are lost waiting for memory access and I/O (Off-chip cycles).

- Let the number of CPU cycles used be $C_{cpu}$ and the time spent off-chip be $C_{off-chip}$.
- Execution time at frequency $f$ is given by $C_{cpu}/f + C_{off-chip}$. 

Recap

**DVS**

- **Reduce Frequency Only**
  - Processor Sleeps when Idle
    - Bad idea!
  - Processor Always On
    - Good idea!

- **Reduce Frequency and Voltage**
  - Processor Sleeps when Idle
    - Good idea down to a Critical Frequency only
Processor Performance States (P-States)

- **P0** max power and frequency
- **P1** less than P0, voltage/frequency scaled
- **P2** less than P1, voltage/frequency scaled
- ...
- **Pn** less than $P(n-1)$, voltage/frequency scaled
Processor “Sleep” States (C-states)

- **C0**: is the operating state.

- **C1** (often known as *Halt*): is a state where the processor is not executing instructions, but can return to an executing state instantaneously. All ACPI-conformant processors must support this power state.

- **C2** (often known as *Stop-Clock*): is a state where the processor maintains all software-visible state, but may take longer to wake up. This processor state is optional.

- **C3** (often known as *Sleep*) is a state where the processor does not need to keep its cache, but maintains other state. This processor state is optional.
Energy expended on wakeup, $E_{\text{wake}}$

To sleep or not to sleep?

- Not to sleep (for time $t$):
  \[ E_{\text{no-sleep}} = (k_v V^2 f + R_0) t \]

- To sleep (for time $t$) then wake up:
  \[ E_{\text{sleep}} = P_{\text{sleep}} t + E_{\text{wake}} \]

To save energy by sleeping: $E_{\text{sleep}} < E_{\text{no-sleep}}$

\[ t > \frac{E_{\text{wake}}}{k_v V^2 f + R_0 - P_{\text{sleep}}} \]
DPM and the Problem with Work-conserving Scheduling

- No opportunity to sleep ☹️

Task 1 (C=2, P=12)

Task 2 (C=1, P=16)

Minimum sleep period
DPM and the Problem with Work-conserving Scheduling

- Must batch! 😊

Task 1 (C=2, P=12)

Task 2 (C=1, P=16)

Minimum sleep period
Device Forbidden Regions

- Treat sleep periods like the *highest-priority* sporadic task. Use *response time analysis* for schedulability. Problems?
  - A Valid solution, but pessimistic.
    (Called: Device Forbidden Regions. Published in RTAS 2008.)

Task 3 (C=11, P=16)

Task 1 (C=2, P=12)

Task 2 (C=1, P=16)
How Many Processors to Use?

- Consider using one processor at frequency $f$ versus two at frequency $f/2$
- Case 1: Total power for one processor
  - $k_f f^3 + R_0$
- Case 2: Total power for two processors
  - $2 \{k_f (f/2)^3 + R_0 \} = k_f f^3/4 + 2 R_0$
- The general case: $n$ processors
  - $n \{k_f (f/n)^3 + R_0 \} = k_f f^3/ n^2 + n R_0$
The general case: $n$ processors

- $Power = n \{k_f (f/n)^3 + R_0\} = k_f f^3/ n^2 + n R_0$
- $dPower/dn = -2 k_f f^3/ n^3 + R_0 = 0$

$$n = \frac{3}{2k_f f^3} \sqrt[3]{\frac{2k_f f^3}{R_0}}$$
The rate of change of temperature is proportional to the difference between input power and output power (via cooling):

\[
\frac{dT}{dt} = P_{in} - P_{out}
\]

\[P_{in} = f(DVS, sleep)\]

\[P_{out} = g(T)\]
Classical Feedback Control Loops

Desired Set Point → Controller (Policy) → Actuator (Mechanism) → Process → Output

Sensor → Measured Output

Feedforward
Stability – Recap

Phase equation:  \( \sum_i p_i(f) = \pi \)  \( \rightarrow \)  \( f \) is obtained

Gain equation:  \( \prod_i g_i(f) \) must be less than 1 for stability

\[
\begin{align*}
T_r & \quad \quad e \quad \quad \quad T_m \\
\text{Thermometer} & \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{Measured Temperature}
\end{align*}
\]

\[
\begin{align*}
g_1(f) / p_1(f) & \quad \quad \text{Proportional Controller (Policy)} \\
g_2(f) / p_2(f) & \quad \quad \text{Heater/Cooler (Mechanism)} \\
g_3(f) / p_3(f) & \quad \quad \text{Room}
\end{align*}
\]

\[
\begin{align*}
g_4(f) / p_4(f) & \quad \quad \text{Thermometer}
\end{align*}
\]
# Summary of Basic Elements

Input = \( \sin (wt) \)

<table>
<thead>
<tr>
<th>Element</th>
<th>Gain</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrator</td>
<td>( \frac{1}{w} )</td>
<td>(-\pi/2)</td>
</tr>
<tr>
<td>Differentiator</td>
<td>( w )</td>
<td>( \pi/2 )</td>
</tr>
<tr>
<td>Pure delay element (Delay = D)</td>
<td>1</td>
<td>(-wD)</td>
</tr>
<tr>
<td>First order lag (time constant = ( \tau ))</td>
<td>( \frac{K}{\sqrt{1+(\tau w)^2}} )</td>
<td>(-\tan^{-1}(w \tau))</td>
</tr>
<tr>
<td>Pure gain (Gain = K)</td>
<td>( K )</td>
<td>0</td>
</tr>
</tbody>
</table>

Note:

\[ w = 2 \pi f_{osc} \]

Where \( f_{osc} \) is the loop frequency of oscillation.
Steady State Error

At steady state the system “catches up” – phase shift is zero.

\[ T_r - e g_1(f) g_2(f) g_3(f) g_4(f) = e \]

\[ e = \frac{T_r}{1 + \prod_i g_i(f)} \]
Example

When a processor executes a task, its temperature increases with normalized frequency, $f$, with a time constant $= 1$ minute. An increase of 0.1 in $f$ causes a temperature increase of 3 degrees. An on-chip sensor reports temperature with a 10 second delay. Design a frequency controller such that processor temperature stays at or around 80 degrees.
Solution:
Step 1: Plot the Control Loop

Read the question/narrative carefully and identify all the blocks involved.
Solution:
Step 2: Identify the Type and Parameters of all Blocks

Identify all the block types and their parameters.

\[ K_c, D = 10s, K_p = 30, \tau = 60s \]
Solution:
Step 3: Compute the Natural Frequency of Oscillation from Phase Equation

Substitute in the phase equation and solve for \( f \).

\[-2\pi f (10) – \tan^{-1} (2\pi f 60) = -\pi\]

Thus, \( f = 0.027 \)
Solution:
Step 4: Compute Controller Gain from Gain Equation

Substitute in the gain equation and solve for $K_c$

$$K_c \cdot 30 / \sqrt{1 + (2\pi f \cdot 0.60)^2} = 0.5$$

where $f = 0.027$

Thus, $K_c = 0.17$

In other words, the control policy should set the processor frequency $f$ to $0.17 \cdot e$, where $e$ is the difference between 80 and the actual processor temperature.
Solution:
Note: Steady State Error Calculation

The steady state error is $80/(1 + 30 \times 0.17) = 13$ degrees.
Example

- When a processor executes a task, its temperature increases with normalized frequency, $f$, with a time constant $= 1$ minute. An increase of 0.1 in $f$ causes a temperature increase of 3 degrees. An on-chip sensor reports temperature with a 10 second delay. Design a frequency controller such that processor temperature stays at or around 80 degrees.

- How would your design change if the energy-optimal frequency of the processor was 0.5?
Example

- When a processor executes a task, its temperature increases with normalized frequency, $f$, with a time constant = 1 minute. An increase of 0.1 in $f$ causes a temperature increase of 3 degrees. An on-chip sensor reports temperature with a 10 second delay. Design a frequency controller such that processor temperature stays at or around 80 degrees.

- How would your design change if the energy-optimal frequency of the processor was 0.5?

- How would your design change if in addition the processor executed real-time tasks using RM?
Example

- When a processor executes a task, its temperature increases with normalized frequency, $f$, with a time constant = 1 minute. An increase of 0.1 in $f$ causes a temperature increase of 3 degrees. An on-chip sensor reports temperature with a 10 second delay. Design a frequency controller such that processor temperature stays at or around 80 degrees.

- How would your design change if the energy-optimal frequency of the processor was 0.5?

- How would your design change if in addition the processor executed real-time tasks using RM?

- Does your control loop ensure zero steady state error? If not, redesign.
Example

- When a processor executes a task, its temperature increases with normalized frequency, $f$, with a time constant = 1 minute. An increase of 0.1 in $f$ causes a temperature increase of 3 degrees. An on-chip sensor reports temperature with a 10 second delay. Design a frequency controller such that processor temperature stays at or around 80 degrees.

- How would your design change if the energy-optimal frequency of the processor was 0.5?

- How would your design change if in addition the processor executed real-time tasks using RM?

- Does your control loop ensure zero steady state error? If not, redesign.

- What is the effect of sensor delay on frequency of oscillation?