What Is Feedback Control?

- **Definition**: Applying input to cause the system variable to conform to a desired value called the reference.
- **Everyday examples**
  - Car cruise control: gas input → speed = 60 mph
  - Air Conditioning: power input → temperature = 70°F

**Advantages of Feedback Control Theory**

- A science with over 100 years of history
- Has been successfully applied in all engineering fields
- Car cruise control, air conditioning and more
- Why feedback control theory?
  - Best way to design control loops
  - Well-established algorithms to choose control parameters
  - Guaranteed stability
  - Control design based on requirements
  - Steady state error, setting time
  - Quantitative way to analyze the impact of system variation on control performance

**CPU Utilization Control in Real-Time Systems**

- Why CPU utilization control?
  - Overload protection
  - Real-time priorities > system priority
  - Utilization > 100% → Operating system freezes
  - Meet deadlines for periodic real-time tasks
  - CPU utilization < schedulable threshold → meet all deadlines
- Control method
  - Reference: CPU utilization = 70%
  - Actuation: Task period

**Feedback (Closed-Loop) Control**

[Diagram showing a control system with controller, actuator, monitor, and reference.]

**Recap of Last Class**

- Feedback control
  - What is feedback control?
  - Why feedback control?
  - Design of a utilization controller
- System modeling
- Performance specs/metrics
- Controller design
- Summary
Control Design Methodology

1. **Modeling** → **System Model** → **Controller** → **Control Algorithm**
2. **Requirement Analysis** → **Performance Specifications**

---

Distributed Real-Time Systems

- An (end-to-end) task is composed of multiple subtasks running on multiple processors
  - Message/event
  - Remote method invocation
- Subtasks are subject to precedence constraints
  - Task = a chain/tree/graph of subtasks
  - E.g. ship navigation

---

The Common End-to-End Task Model in Distributed Real-Time Systems

- Periodic task \( T_i \) = a chain of subtasks \( T_{ij} \) on different processors
  - All subtasks run at a same rate
  - End-to-end deadline is divided into a set of sub-deadlines
- Task rate can be adjusted within a range
  - E.g., online video can trade-off between quality and rate
    - Higher rate → better video quality & higher CPU utilization

---

End-to-End Utilization Control

- CPU utilization: a trade-off
  - Too high → system overload → possible crash
  - Too low → poor performance (e.g. poor video quality)
  - Utilization ≤ schedulable bound → meet all sub-deadlines
- End-to-end utilization control
  - Utilizations of all processors ≤ bounds → meet all end-to-end deadlines
  - Constraints on task rates
  - Stability assurance

---

Challenges of End-to-End Utilization Control

- Multi-Input-Multi-Output (MIMO) control
- Utilizations are coupled due to end-to-end tasks
  - Rate change affects all processors in the task chain
- Constraints on task rates
- Stability assurance

---

EUCON – Centralized Control Algorithm

- EUCON (End-to-end Utilization CONtrol)
  - Designed based on Model Predictive Control (MPC) theory
  - Invoked periodically to control the utilizations of all processors

---
Design Steps of Model Predictive Controller
1. Derive the dynamic model of the controlled system
2. Design the controller
   Design objective:
   \[
   \min_{\{u(k), k(k ≥ 0)\}} \sum_{k=0}^{n} (B_i - u_i(k))^2
   \]
   subject to rate constraints: \( R_{\min} \leq r_i(k) \leq R_{\max} \) (1 ≤ j ≤ m)
3. Analyze system stability

Step 1: Dynamic System Model
- New utilization = old utilization + change
- Utilization change = actual exec time x task rate change

System model:
\[
\begin{align*}
   u_j(k) &= u_j(k-1) + g_j \Delta r_j(k-1) \\
   u_j(k) &= u_j(k-1) + g_j \Delta r_j(k-1)
\end{align*}
\]
- \( c_j \): estimated execution time of \( T_j \)
- \( g_j \): actual execution time / estimation
- Models uncertainty in execution times

Step 2: Controller Design

Step 3: Analytical Stability Assurance
- Stability: converge to desired bounds from any initial condition
- Uncertainty gain: \( g = \) actual execution time / estimation
- Stability condition: tolerable range of uncertainties

The Effect of System Gain
- System is designed based on estimated exec times
- In run-time, real exec times are different from estimations
- The difference is called system gain

Utilization Modeling on One Processor
- CPU utilization is the sum of the utilizations of all tasks
- Real task execution times differ from their estimations in a ratio \( g \):
  - Running on a different processor
  - Influenced by external inputs

In the \( k \)th time period
\[
\begin{align*}
   u(k) &= u(k-1) + g(k-1) \Delta r_j(k-1) \\
   u(k) &= u(k-1) + g(k-1) \Delta r_j(k-1)
\end{align*}
\]

CPU utilization in a real-time server
- Task 1
- Task 2
- Task 3

CPU utilization of task \( i \): execution time
- In execution time = task rate
- \( g_i \):

The Core of MPC is a constrained optimization solver

Cost function: \( V(k) = \sum \Delta r_i(k+1) - \Delta r_i(k-1) \)
Experiments: Workload Uncertainties

- 12 tasks (25 subtasks) running on 4 Pentium IV processors (Linux)

![Graph showing execution times changing at runtime]

- Disturbance from external resource contention

**FC-ORB**

Feedback Controlled Object Request Broker

- End-to-end utilization control
  - Maintains desired utilizations on all processors under uncertainty
  - FC-ORB implements EUCON in a real system
- End-to-end ORB architecture
  - Specialized for rate adaptation due to utilization control
- Safe task migration
  - Improves reliability in terms of both system functions and real-time performance

**Why CPU Utilization Control?**

- Overload protection
  - Utilization too high → deadline misses & system crash
- Under-utilization avoidance
  - Utilization too low → poor application QoS
- Uncertainties
  - Unpredictable exec times (e.g. influenced by sensor data)
  - External resource contention (e.g. Denial of Service attacks)

- **Goal 1:** achieve desired utilization under uncertainty
  - Utilization ≤ schedulable utilization bound → meet deadlines
  - Highest possible utilization while meeting deadlines

**Goal 2: Performance Portability**

- Existing real-time middleware
  - Support functional portability
  - Lack performance portability
  - Manual utilization configuration → time-consuming
- Integration of real-time middleware with utilization control
  - Guaranteed utilization regardless of OS/HW/computing capacity

**Goal 3: Safe Task Migration**

- Task migration causes overload and deadline misses
  - Overload → system crashes
  - OS frozen by higher-priority real-time threads
  - Deadline miss is equivalent to function failure

- **Need** utilization control!

**End-to-End Utilization Control Service**

- Implements EUCON (End-to-end Utilization CONtrol)
- Designed based on Model Predictive Control (MPC) theory
- Provides functional and performance portability

![Diagram showing CPU utilization and task migration control]
End-to-End Object Request Broker

- End-to-end task implementation
- Release guard – enforce periodic releases of subtasks
- Priority management
  - Rate adaptation → continuous priority changes
  - Thread-per-priority → high overhead
  - Need dynamically move subtasks between threads
- Thread-per-subtask
  - Change priority only when the order of task rate changes

FC-ORB: Priority Mapping and Management

- 6 priorities; 3 subtasks in the system

Safe Task Migration

- Fault model: permanent processor failure
- Focus: integration of utilization control with fault-tolerance
- Subtasks have backups on different processors
- Automatic controller reconfiguration

Experimental Setup

- 12 tasks (25 subtasks) and 4 Pentium IV processors
- KURT Linux 2.4.22
- RMS (Rate Monotonic Scheduling)
- Subtasks on Norbert have backups on other processors

Goal 1: Robust Utilization Control

- Execution times change at runtime
- Desired utilization: 73% (0.73)
- Disturbance from external resource contention
Goal 2: Performance Portability
- Same utilization – portable performance
- Even on different systems with different computing capacity
- Real exec times are twice longer than normal (running on slow machines)
- Desired utilization: 73% (0.73)

Real exec times are 1/4 of normal (running on fast machines)

Goal 3: Safe Task Migration
1. Norbert fails.
2. Tasks migrated to other processors
3. System has overload!!
4. Control utilization by adjusting task rates

Summary
- EUCON provides analytical utilization guarantees despite uncertain execution times
- The Model Predictive Control (MPC) approach
- Handles coupling among processors
- Enforces constraints on task rates
- Analyze tolerable range of variation in execution times
- FC-ORB real-time middleware
- Implements EUCON in a real system to handle
  - Unknown or varying execution times
  - External disturbances
- Provides performance portability, despite different OS/HW/computing capacity
- Enables safe task migration