CPEN 333:
System Software Engineering

INSTRUCTOR: Ali Mousavifar

System:

whole compounded of **several parts** or members

- parts are distinct “enough” to be considered separate
- a common purpose or unified task

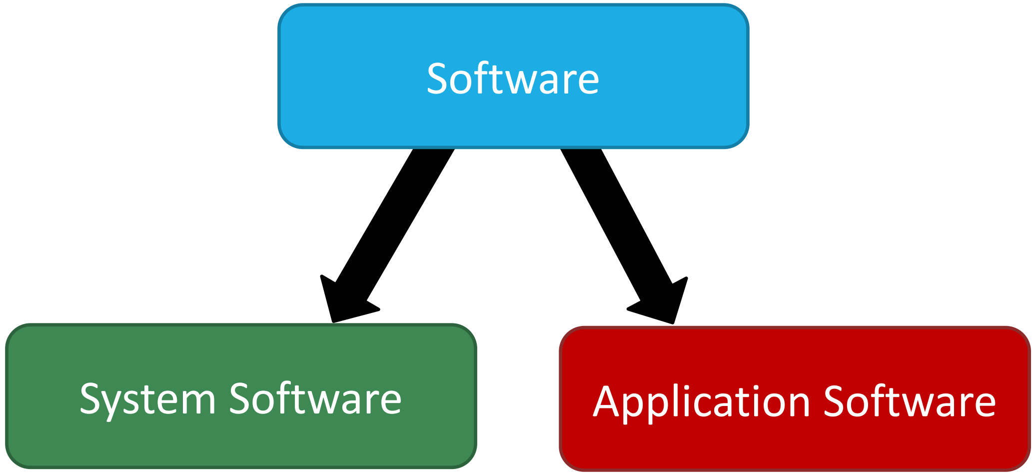
Software:

the **programs** and other **operating information** used by a computer.

- computer instructions
- non-executable data

System Software:

software that runs or controls systems

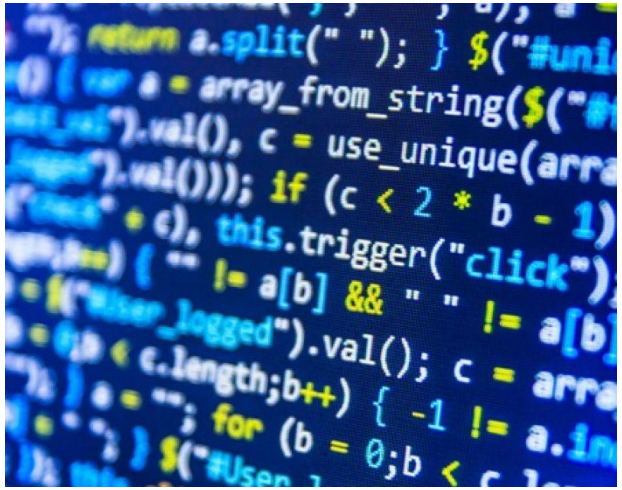


System Software:
- system-centered
- runs in background, controls things "fairly" autonomously
- interaction is not main use-case

Application Software:
- user-centered
- runs in foreground
- initiated by user
- performs specific task directly for user

Engineering:

the use of **knowledge** in order to purposefully **design**, develop and **build** a product



http://www.salford.ac.uk/news/articles/2017/software-spin-out-in-global-growth-deal

- requires planning
- specific tools and methodologies
- important for large and real-world systems
- critical systems need to be proven on paper

System Software Engineering:

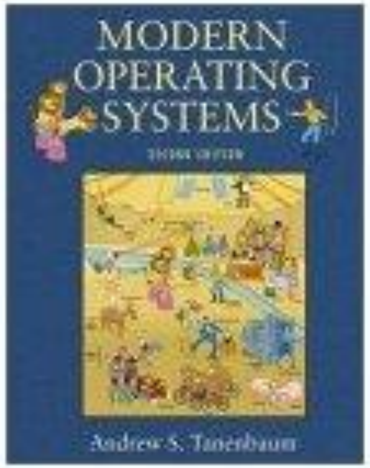
deliberate design and construction of software that runs or controls systems.

- Methods for components of system can communicate
  Inter-Process **Communication**
- Methods for coordination of actions
  Inter-thread/Inter-process **Synchronization**
- Tools and methodologies for designing, communicating those designs, and testing
  Unified Modelling Language, Testing Frameworks

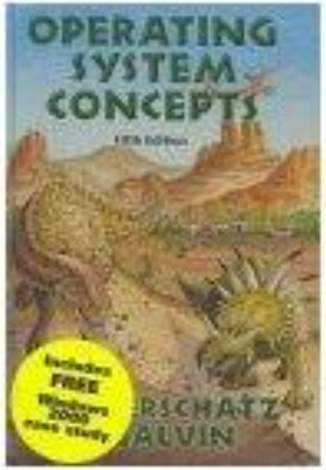
CPEN 333: System Software Engineering

Website(s): https://canvas.ubc.ca/courses/77997

Textbooks:
(optional)



A. Tanenbaum



Silberchatz, Galvin, Gagne

CPEN 333: System Software Engineering

References:

- Microsoft C# Documentation: https://docs.microsoft.com/en-us/dotnet/csharp/
- UML Documentation: https://www.omg.org/spec/UML/2.5.1/PDF
- Generate UML online/app: https://plantuml.com/

Labs:

There is a 2 hour lab session every week (starting next week).

- 3-4 labs will include **lab activities .**
- The TAs will be present to give assistance and/or guidance.
- Labs are designed so that they are completed during the 2 hour session. However, if you need more time, you can submit it within 1 week from the beginning of your Lab section (i.e. in one week).
- Labs are graded by the TA, so be be prepared to answer questions
- Labs are to be completed **individually**
- You will also need a TurnItIn account (stay tuned)
- You will also need a github free account

Project:

Will involve: system design document, and working implementation of a simulated system software tool.

- will be due during the last week of class
- can be completed **in groups of 3-4**
- option between **provided system** or a **custom** one of your choosing
- more details to come around week 4 or 5

Midterm:

Will be mainly hands-on programming, and will cover most of the technical content.

- most questions will come directly from in-class **discussions** or **learning objectives**
- 20% of the final grade
- will occur around week 8 or 9
- more details to come

Final:

Will be mainly hands-on programming, and will cover most of the technical content.

- Most questions will come directly from in-class **discussions** or **learning objectives**
- 25% of the final grade
- more details to come

TAs:

- Elaine Yao (elainey@ece.ubc.ca, github: ElaineYao)
- Geetika Batta (geetika.batta@gmail.com, github: geetikabatta)
- Minh To (tnnhatminh@gmail.com, github: minhto2802)
- Rafiuzzaman Mohammad (xeon.rafi@gmail.com, github: rafizaman)
- Parisa Beigi (pabeygi@student.ubc.ca, github: parisa-beygi)

Grade Breakdown:

- Participation and collaboration with peers: 3%
- Quizzes: 12%
- Labs: 20%
- Project: 20%
- Midterm: 20%
- Finals: 25%

Lecture 1:
Introduction to System Software Engineering

Learning Goals

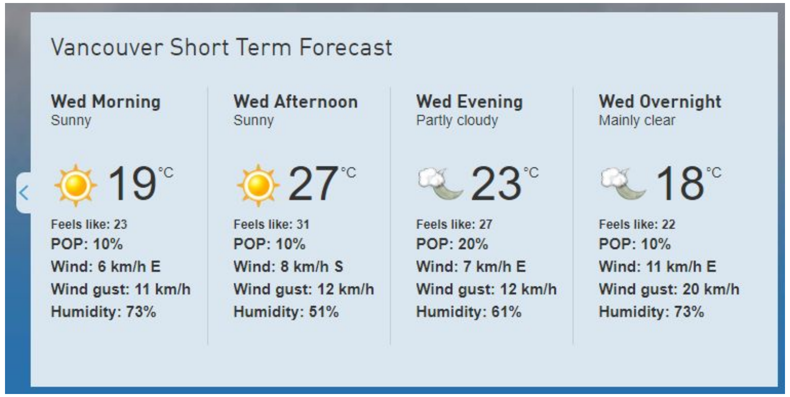
- Define a **system**, **system software**, and **system software engineering**
- Distinguish between **hard** and **soft** time constraints
- Distinguish between **event-driven** and **time-driven** systems, and give examples
- Describe the difference between **interrupts** and **polling** for detecting and handling events
- List advantages and disadvantages of both interrupts and polling
- Discuss why **testing** real-time systems can be challenging

What is a real-time system?

By Raysonho @ Open Grid Scheduler / Grid Engine - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=39098282

http://ethw.org/Tracking_the_Ice_Hockey_Puck_-_FoxTrax_(Glow_Puck)

https://www.tesla.com/en_CA/autopilot



www.theweathernetwork.com

What is a real-time system?

There is no widely accepted single definition
☹️

- A real-time system is generally a controlling system, often embedded into equipment so that its existence is not obvious. It takes in information from its environment, processes it and generates a response.

- A real-time system reacts, responds, and alters its actions to affect the environment in which it is placed.

What is a real-time system?

- A real-time system implies that there is something significant and important about its response time.
- A real-time system has a guaranteed, deterministic worst-case response time to an event under its control.
- A real-time system is one where the correct answer at the wrong time is the wrong answer.

What is a real-time system?

credit: the Times UK

Real-Time limit: 15 ms

What is a real-time system?

https://www.macrumors.com/2017/07/31/apple-denied-dismissal-facetime-ios-6-lawsuit/

Real-Time limit: 200 ms

What is a real-time system?

Real-Time limit: ?

Conclusion: depends on application context.

Real-time classification: Hard vs Soft

Hard Real Time Systems

- Have “hard” deadlines (i.e. are **time-critical**)
- Failure to satisfy **worst-case response time** leads to **system failure**
- Specs will mention **maximum response time**, and **effects of failure**

Examples: Pace-makers, ABS brakes, auto-pilot, games

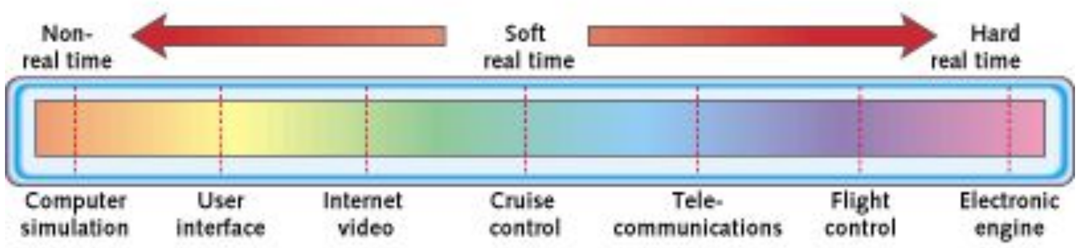
Real-time classification: Hard vs Soft

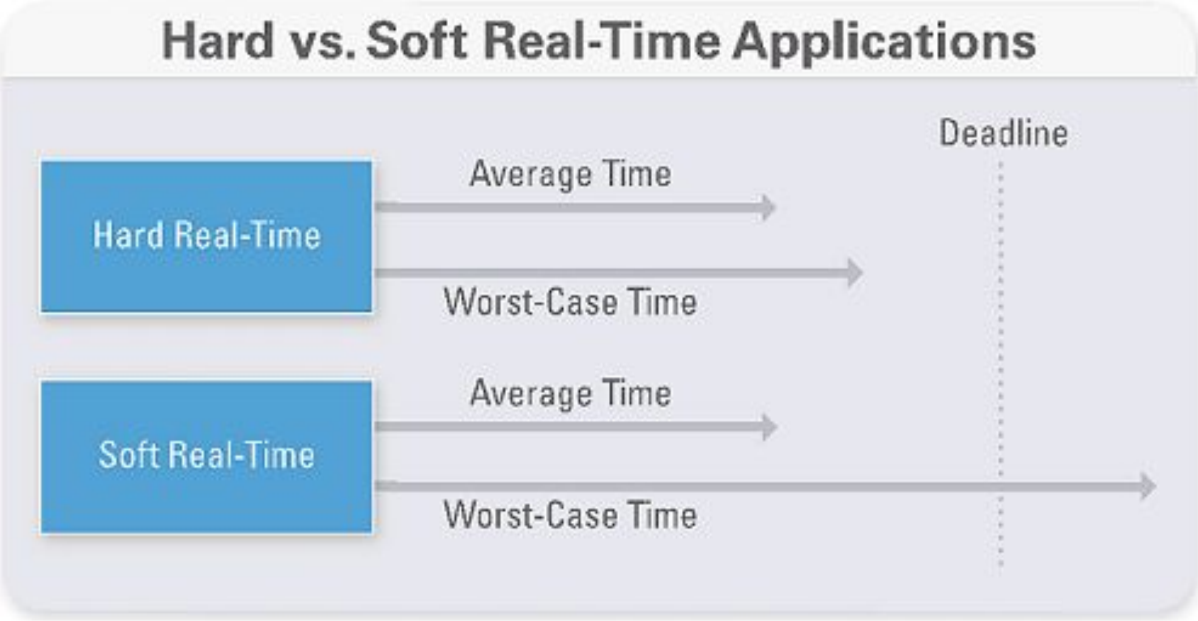
Soft Real Time Systems

- Have “soft” deadlines (i.e. are not time-critical)
- Failure to satisfy **deadlines** leads to **system degredation**
- Specs will mention **average response time**, and **measures of degredation**

Examples: Elevators, cruise control, ATMs, thermostats

Real-time classification: Hard vs Soft





Real-time classification: Event vs Time

Event-Driven Systems

An input sensor is responsible for detecting the occurrence of an **event** that impacts the system's success, and **triggers** a response

Time-Driven Systems

The system processes inputs and reacts at **specified times**, such as at a periodic interval.

Real-time classification: Event vs Time

Event-Driven Systems

Events can generally be **detected** in one of two ways:

- a **switch** or **button** that generates an **interrupt signal**
- a sensor that responds to **status enquiries**, which can be **polled** by a controlling process

Event-Driven Systems: Interrupts

An **interrupt signal** tells the CPU there is an important event

The CPU…

- **suspends** its current thread of execution
- **saves** the current state
- calls an **interrupt handler** or **interrupt service routine (ISR)**
- **loads** previous state and **resumes** original thread

Event-Driven Systems: Interrupts

Advantages of interrupts:

- system is **notified immediately** of event, so can be **handled immediately**
- **deterministic** response times
- interrupts can be **prioritized**
- response time is **independent** of **number of sensors**

Disadvantages of interrupts:

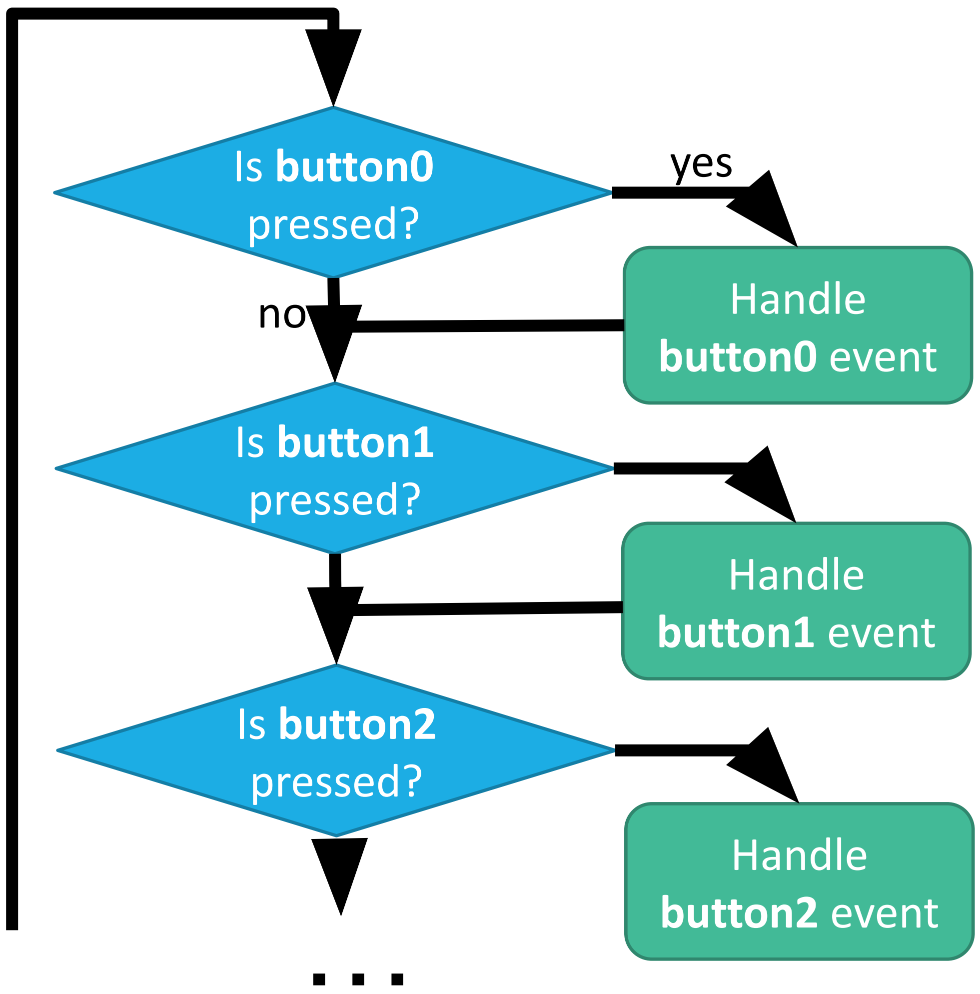
- complex hardware is needed (sensors, system-level handlers for ISRs, etc)
- difficult to test and debug
    - interrupts are unpredictable, ISR and main code run **asynchronously**

Event-Driven Systems: Polling

The main program continuously probes the sensors to check their status

More **choice** in handling:

- can handle in **main thread**,
- or to launch a separate **asynchronous thread**



Event-Driven Systems: Polling

Advantages of polling:

- Simpler code, more flexibility
- Easier to debug since event detection is predictable

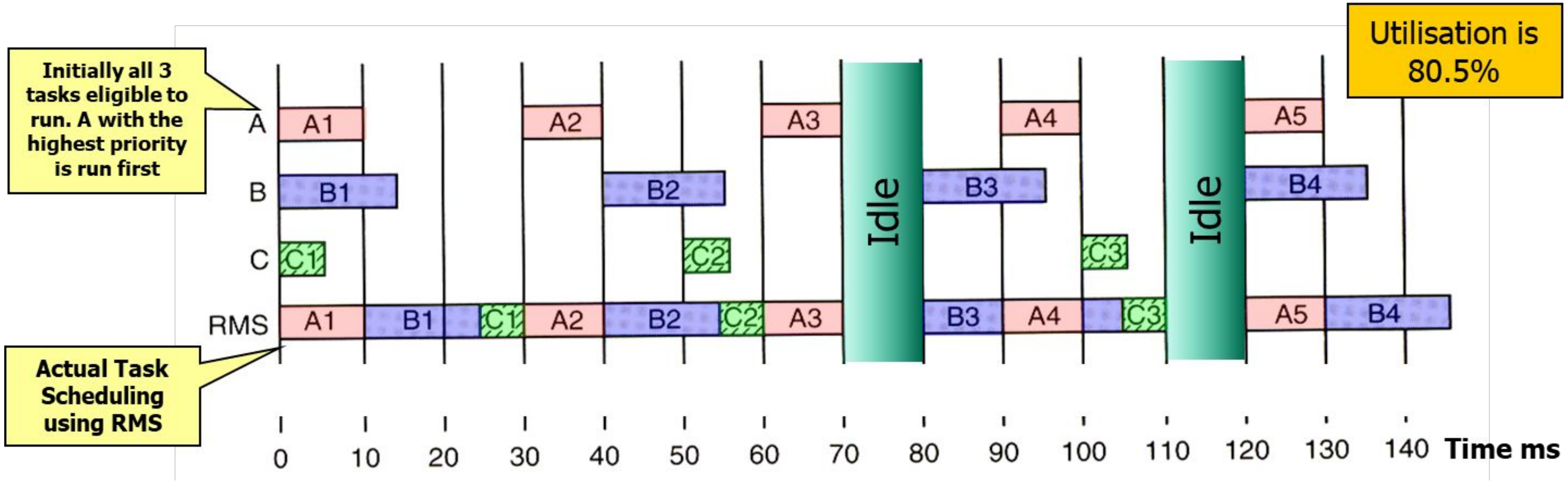
Disadvantages of polling:

- Sensor events may be missed
- Difficult to prioritize events
- Adding more sensors degrades performance and response times
- Consumes a lot of CPU time, particularly if events are few and far between

Time-Driven Systems

Certain actions **occur** at **specified times**, or at **periodic intervals**

e.g. Task A has to be performed every 30ms,
Task B every 40ms
Task C every 50ms



Testing Real-Time Systems

Can be challenging because will often involve **asynchronous** and **unpredictable** "real-world" events.

What is an elevator going to be doing 5 seconds after we push the button?

What in an airplane-autopilot going to be busy with 20 minutes into a flight?

- Exhaustive testing is prohibitively expensive and time-consuming
- If system fails, may be difficult to reproduce conditions
- Debugging is a challenge: break-points and stepping interferes with timings and sequences

Ariane 5

Exploded on its maiden flight in 1996

An exception occurred when **a 64-bit floating point** number was assigned to a **16 bit integer** and it was out of range


```c
#include <stdio.h>

int main(void)
{
    double x = 5.324e6 ;
    short int y = x ;

    printf("X = %g, Y = %d\n", x, y) ;
    return 0 ;
}
```

```
C:\Windows\system32\cmd.exe
X = 5.324e+006, Y = 15584
Press any key to continue . . .
```

Therac 25

- Radiation machine used to treat cancer patients in Canada and USA
- Two types of radiation could be delivered which required different **scattering** hardware. Errors occurred when the operator changed the type of radiation but the software did not employ interlocks to **wait** for correct scattering H/W to move into place.
- Resulted in massive patient overdoses and even death.

Activity: In small groups, come up with an example of a real-time system, and outline the system software that controls it.

Think about:

- design of the system software
- real-time context
  - hard vs soft
  - event-driven or time-driven