# Real-Time challenges and concepts:

## Simple Linux real-time services compared to non-real-time

Dr. Sam Siewert

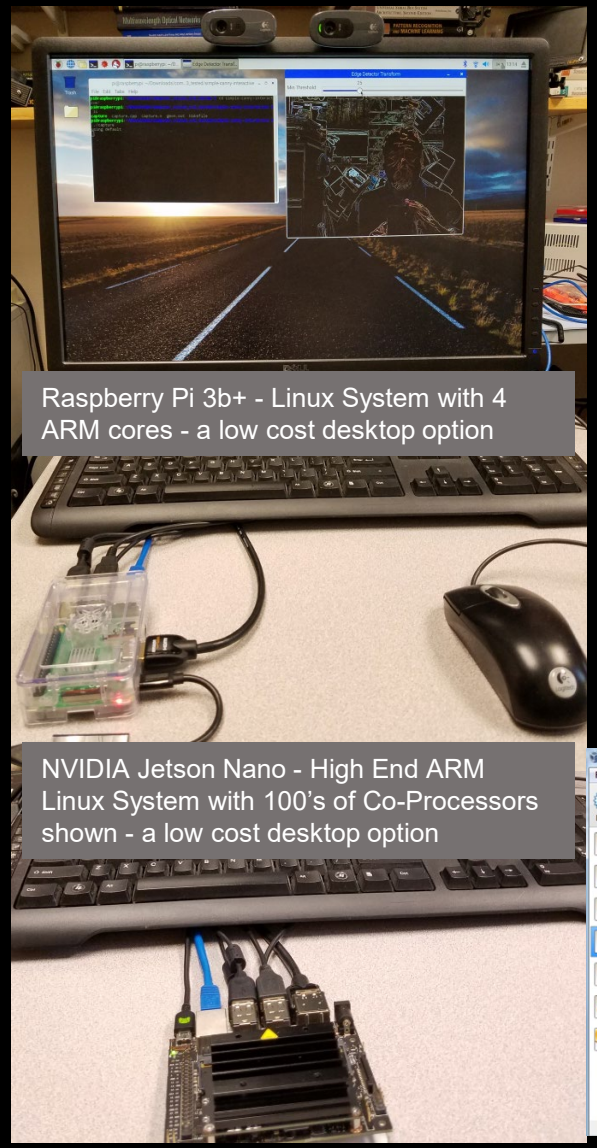
Electrical, Computer and Energy Engineering

Embedded Systems Engineering Program

## Segment Outline

- More Home Lab Options and Details
- Review of basic definitions and Rate Monotonic Least Upper Bound Intro
- Simple POSIX threads (non-Real-Time) demonstration
- Scheduling concepts for threads, tasks, and processes
- Course Outline for Series and Scaffold of Knowledge for Project

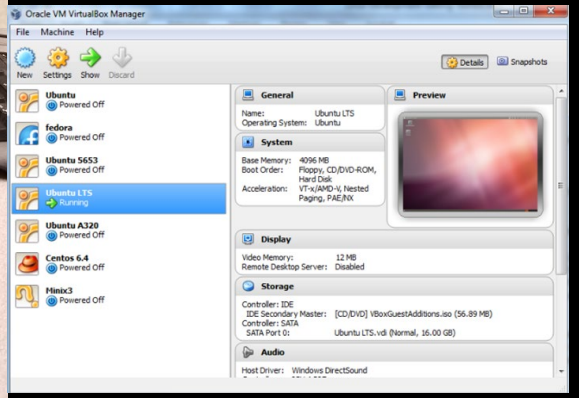
- Option #1 - Raspberry Pi 3b+
  - $80 kit
  - CanaKit for Raspberry Pi 3 B+ on Amazon
  - You own it, can be used for entire course
- Option #2 - NVIDIA Jetson Nano
  - $99 for students
  - JetPack Linux installation
- Option #3 – Virtual-Box with Ubuntu LTS
  - Use Windows or Mac PC
  - Software Development ONLY
  - No real-time execution or camera streaming
  - Learn Linux with minimal cost and risk
  - Real-time testing with native Linux (Option #1, 2)
- Option #4 - Linux Laptop (Native installation)
- Option #5 - Intel NUC with Ubuntu LTS



Raspberry Pi 3b+ - Linux System with 4 ARM cores - a low cost desktop option

NVIDIA Jetson Nano - High End ARM Linux System with 100's of Co-Processors shown - a low cost desktop option

Virtual Box Linux
Functional Development



- **Firmware vs. Software**
- **Embedded Systems**
- **Real-Time Theory and Practice**
- **See Coursera Resources Terminology.pdf**
- **Also in Book (Notes) in Appendix**
- **For all $S_i$: $C_i$, $T_i$, $D_i$, Compute U, m=number of S**

Firmware: software that runs from power-on/reset, from a non-volatile memory device and directly interfaces to hardware

$$U = \sum_{i=1}^{m} (Ci / Ti) \leq m(2^{\frac{1}{m}} - 1)$$

The Rate Monotonic Least Upper Bound

Simple analysis to get started quick …

# What History Teaches Us

- Apollo 11, Launched July 16, 1969
  - On descent, Lunar Module experienced a computer overload
  - Alarm 1201, 1202 (CPU overload s.t. Sum($C_i/T_i$) > 1.0)
  - Landed July 10, 1969 [By ignoring alarms - lucky?]
- Rate Monotonic Theory - Not yet formalized (math model)

https://www.space.com/26593-apollo-11-moon-landing-scariest-moments.html

Apollo 11's Scariest Moments: Perils of the 1st Manned ...

www.space.com

NASA's Apollo 11 moon landing was a human spaceflight feat, but also a dangerous journey. See some of the scariest moments of first manned moon landing.

$$U = \sum_{i=1}^{m} (Ci / Ti) \leq m(2^{\frac{1}{m}} - 1)$$

"Scheduling Algorithms for Multiprogramming In a Hard-Real-Time Environment",
C.L. Liu
[MIT], James W. Layland
[NASA JPL], Jan 1, 1973

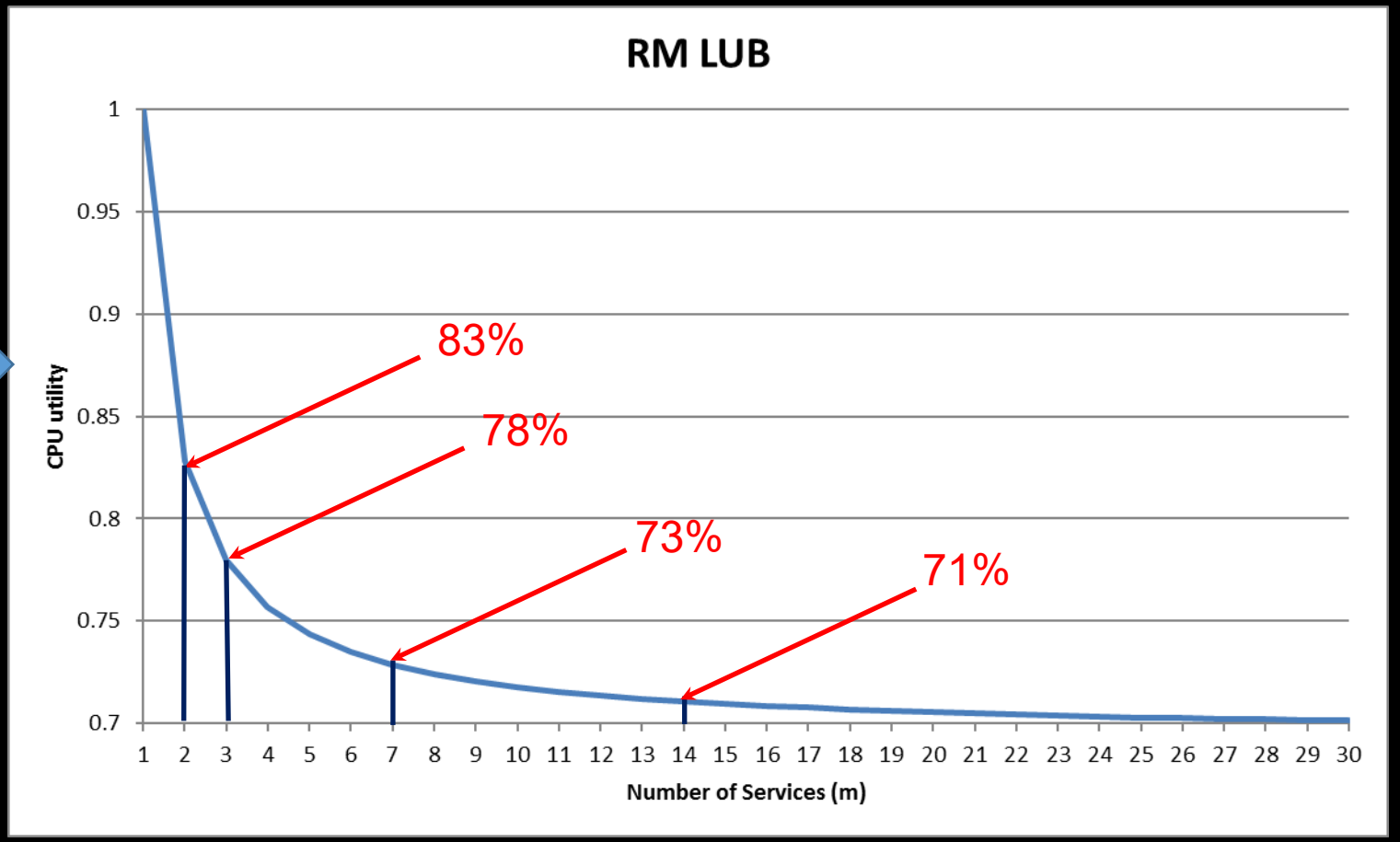
Asymptotes to ln(2), 69.3%, or ($N^{th}$ root of 2 - 1) x m

More Services means we need more margin for safety

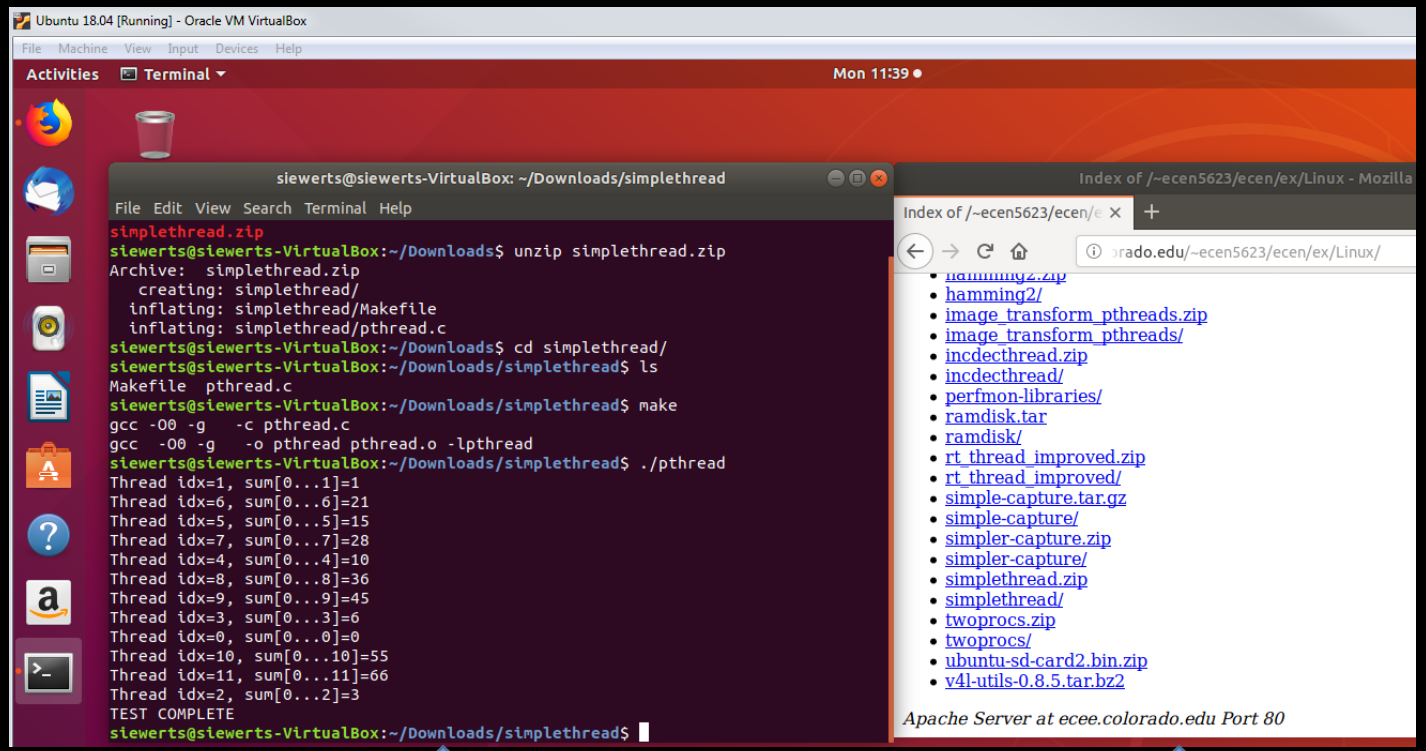
RMA for Beginners - Keep in Mind RM LUB is not Exact!

RM LUB
- Safe
- Inexact bound

$$U = \sum_{i=1}^{m} (Ci / Ti) \leq m(2^{\frac{1}{m}} - 1)$$



- **Non-RT POSIX Threads**
  - E.g. download, build, and run on VB-Linux (or R-Pi)
  - Each thread sums digits between 0 and thread #
  - Prints to monitor
- **Note that the order of completion of each thread is not predictable**
- **Not predictable response (not RT)**




Unzip download, do “make”,
Run ./pthread
(note that “./” is required and can be throught of as meaning “right here” for code you want to run)

Download code from Supporting materials Web page - Example Code

- **POSIX Threads - Basic Example Code for Multi-threading**
  - First Pthread program - start with http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/simplethread/
  - Next Pthread program - using multiple threads with different code http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/incdecthread/

- **Linux Two Process Example**
  - Two Process Example FYI, we will not do multi-programming http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/twoprocs/

- **POSIX Threads - Using Affinity to Assign a Core**
  - Pthread program with mapping to CPU cores by thread http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/simplethread-affinity/

## A Thread of Execution

Thread (of execution) - a thread is simply the trace of a CPU's PC (Program Counter) over time not including context switch code execution by an OS or RTOS. State information may or may not be associated with a thread of execution, but the value of the PC before a context switch is the minimum state that must be maintained on a system which includes preemption.

## Attributes of a Thread in Linux

- NPTL - Native POSIX Threads Library
- Threads are mapped onto Linux Kernel tasks
- Threads can be Real-Time (SCHED_FIFO) or non-Real-Time (SCHED_OTHER)
- Threads will run on default core assigned by OS
- Threads may be migrated to least busy core by OS load balancing
- Threads can be assigned to a specific core (affinity)

- **A Task is a Thread + more context**

  Task - a thread with normal thread state including stack, registers, PC, but also including signal handlers, task variables, task ID and name, priority, entry point, and a number of state and inter-task communication data contained in a TCB (Task Control Block) for VxWorks RTOS or Linux kernel descriptor.

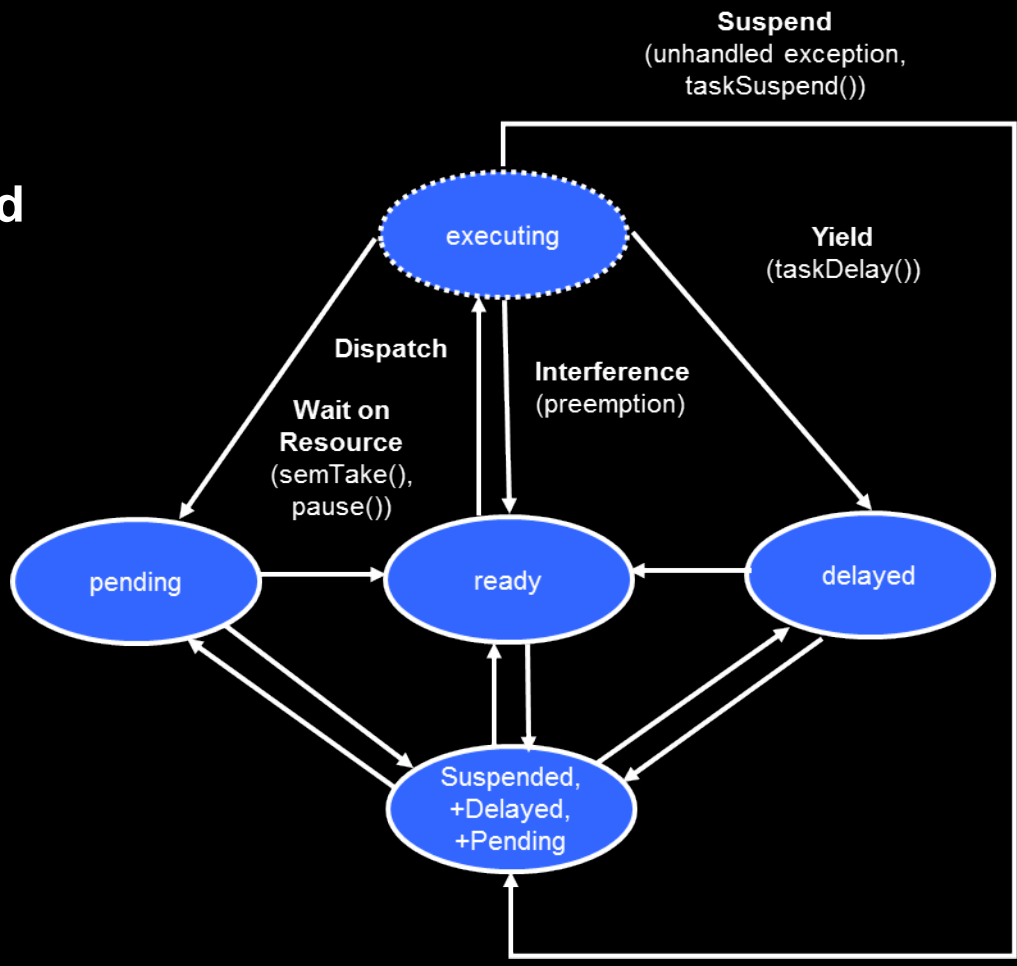
- **Attributes of a Task in Linux vs. RTOS**
  - Only exists in Linux kernel space
  - For an RTOS, this is the main execution context is kernel space (no user space)
  - Tasks can not be created, controlled or accessed in Linux user space
  - Tasks can implement a Linux process or threads within a process
  - https://en.wikibooks.org/wiki/The_Linux_Kernel/Processing
  - Kernel modules can create kernel tasks
  - Kernel tasks will be created for a user space process and threads it contains as needed for Linux Process and NPTL execution model

- **A Process contains 1 or more threads in a protected address space**

  Process - a thread of execution with stack, register state, and PC state along with significant additional software state such as copies of all I/O descriptors (much more than an RTOS task +TCB for example) including a protected memory data segment (protected from writes by other processes).

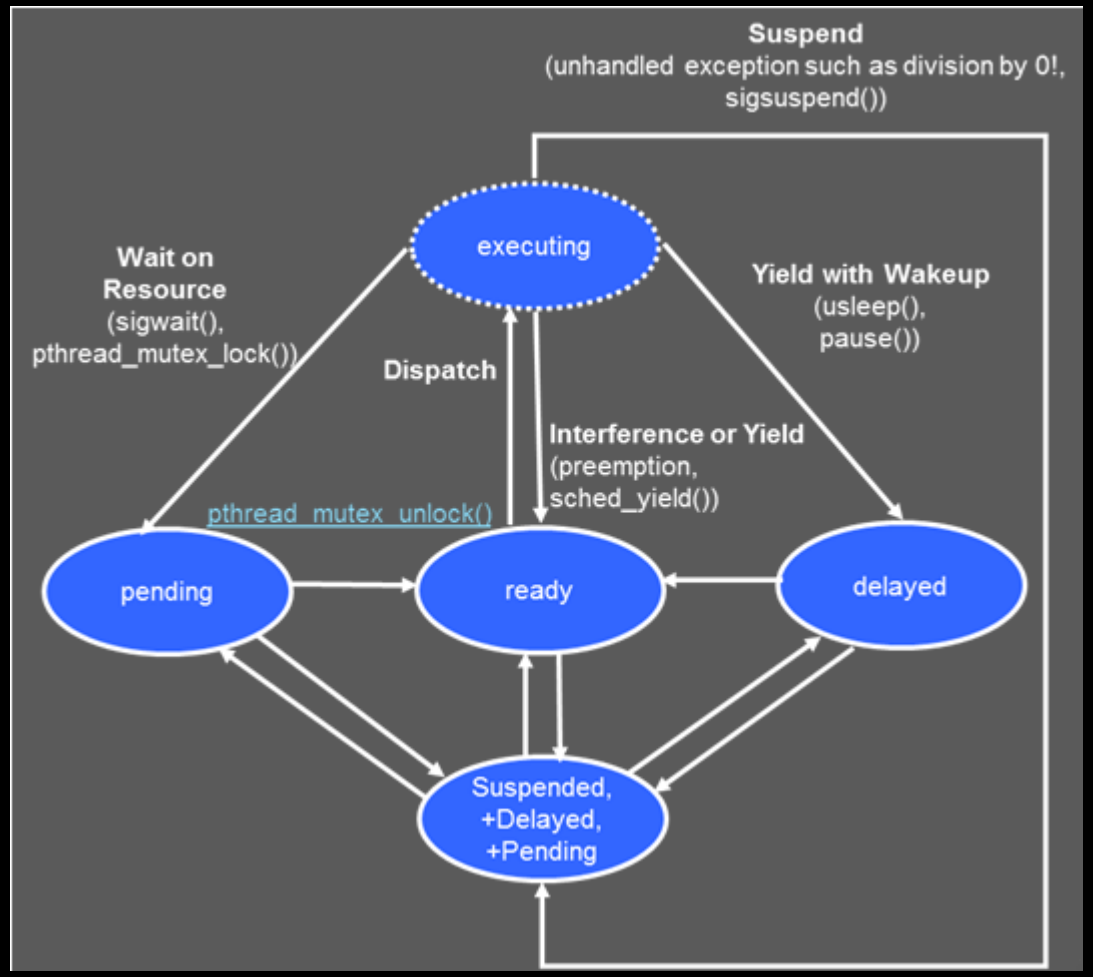
- **Attributes of a Process**
  - Only exists in Linux user space
  - Process main thread is mapped onto a Linux kernel task
  - Serves as a container for 1 or more threads in a common address space
  - Protected address space (wild write in another process can not impact)
  - Additional POSIX threads created are mapped via NPTL onto Linux kernel tasks
  - The Process is an old model for execution in Linux for multi-programming
  - Very safe, isolates and insulates the OS from bad user code and prevents one application from destabilizing another when user code is buggy

- **RTOS Scheduling**
- **Fixed Priority Preemptive**
  - Tasks State Based on Resources Required
    - Needs CPU Only = Ready
    - Non-CPU Resource Required = Pending
  - Task Context Switch on System Call or Interrupt – Dispatch from Ready Queue
  - Dispatch with Rate Monotonic Priority
  - RM LUB Feasibility
  - Feasibility Tests
    - Scheduling Point Algorithm
    - Completion Test Algorithm
    - Over LCM of Periods
- **Dynamic Priority Preemptive**
  - Earliest Deadline First
  - Least Laxity First




E.g. VxWorks RTOS Scheduling State Machine
(Chapter 7, VxWorks Kernel Programmer's Guide)

- Linux NPTL POSIX RT Thread Scheduling
- Fixed Priority Preemptive
  - Common Scheduling State Machine like RTOS
  - Difference is that User-Space Threads are mapped onto Kernel-Space Tasks
  - Task Context Switch on System Call or Software Signal (Software Interrupt)
  - Dispatch with FIFO (Rate Monotonic) Priority
  - System calls are different, but have same effect
- Can Set Processor Core Affinity
- Can Implement Dynamic Priority Preemptive
  - Earliest Deadline First
  - Least Laxity First




E.g. Linux NPTL Thread Scheduling State Machine (https://computing.llnl.gov/tutorials/pthreads/)

- **More Complex RT Systems - Creative**
  http://ecee.colorado.edu/~siewerts/Video/
- **Camera Peak-Up**
  - Camera tilts and pans to track object
  - Keeps target in center of field of view
- **Target Tracker with Designation**
  - fixed wide field of view camera
  - laser pointer tilts and pans to track target
- **Stereo Vision Tracker**
  - Tracking Speed
  - Intensive Image Processing
- **Claude Shannon's Inclined Plane Juggling**
  - Stepper Motor Control
  - Machine Vision tracking of balls in flight
  - Adjustment to changes in incline in real-time
  - Open or closed loop
- **Correct Time –** NIST NTP RF Broadcast F2 Atomic Clock Time Standards USNO Time

E.g. Juggling Balls on an Inclined Plane
(https://www.youtube.com/watch?v=LsLFYhI6gZY)

- **Standard Project (Last Course in Series)**
  1. Real-Time Embedded Systems Concepts and Practices (this course)
  2. Real-Time Embedded Systems Theory and Analysis
  3. Design for Mission-Critical and Software Real-Time Applications
  4. Real-Time Embedded Systems Project

- **Synchronize internal Software State with External**
  - Observing an external clock
  - Two clock domains
  - Simple case of an externally observable process

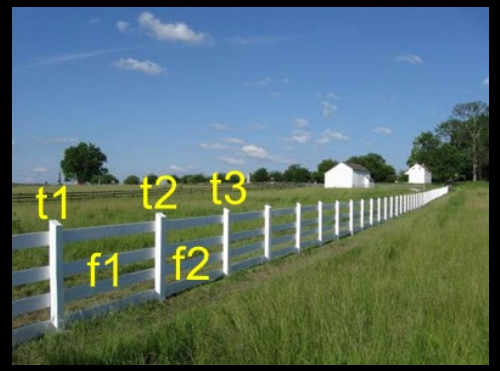
- **Observe Physical Process**
  - With Clock in View
  - Which Clock frame is Correct?
  - How Do We Know?

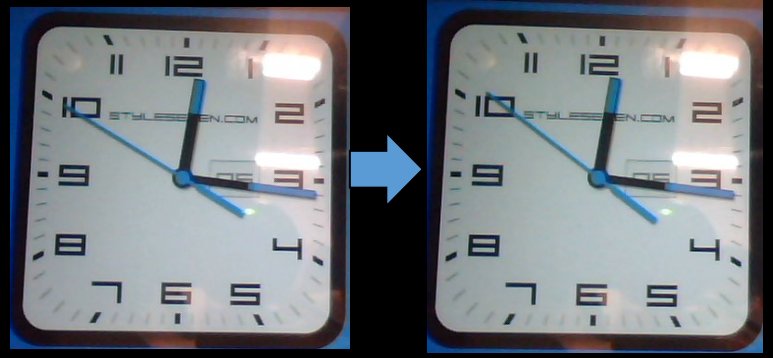
- **Observation at**
  - 1 Hz (under sampling)
  - 10 Hz (over sampling)
  - Max Hz (e.g. 30Hz limit of most webcams)

- **Off by less than a Second in 30 Minutes or More**
  - E.g. 230 milliseconds of drift (accumulation of jitter) by analysis
  - Camera?
  - I/O?
  - Processing?

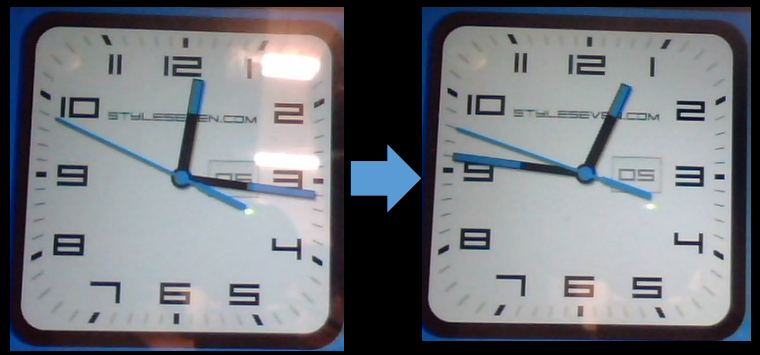
Oversample
- Save stable frames only
- None with hand in flight
- No repeats
- No skips

61 frames for 60 sec
Start=End



Frame N | Frame N+1



Frame 1 | Frame 1800

Frame 1 + 1800 = 1 sec slow
Frame 1 + 1799 = as expected
Frame 1 + 1800 for End=Start
(**same second hand position**)

E.g. Observing the “ticks” of a simple UTC Clock

## Summary

- Learning more about your Home Lab
- RT Embedded Applications with Linux vs. RTOS
  - Linux NPTL POSIX Real-Time Extensions
  - Predictable response (+/- 1 msec is expected)
  - RTOS is better, almost deterministic (+/- microseconds)
  - All software has some non-determinism, we'll find out why in future courses
- Simple POSIX threads creation, join, scheduling attributes and affinity
- Scheduling concepts for threads, tasks, and processes
- Scaffold of Knowledge for Project - last course in series