Real-Time Systems

Introduction to Scheduling of RT Services

Dr. Sam Siewert

Electrical, Computer and Energy Engineering

Embedded Systems Engineering Program

University of Colorado Boulder

So why SW for HRT Systems?

- ASIC and FPGA State-Machine Solutions Offer Hardware Clocked Deterministic Solutions
- FPGAs Can be Updated with New Bit-streams (Synthesized HDL to Reconfigurable Logic Elements)
- Software (Firmware) Remains Simplest for Filed Upgrade (Reconfigurable at Run-time)
- FPGAs Can be Costly Per Unit
- Cost of Software Engineering vs Hardware Engineering

REVIEW - 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

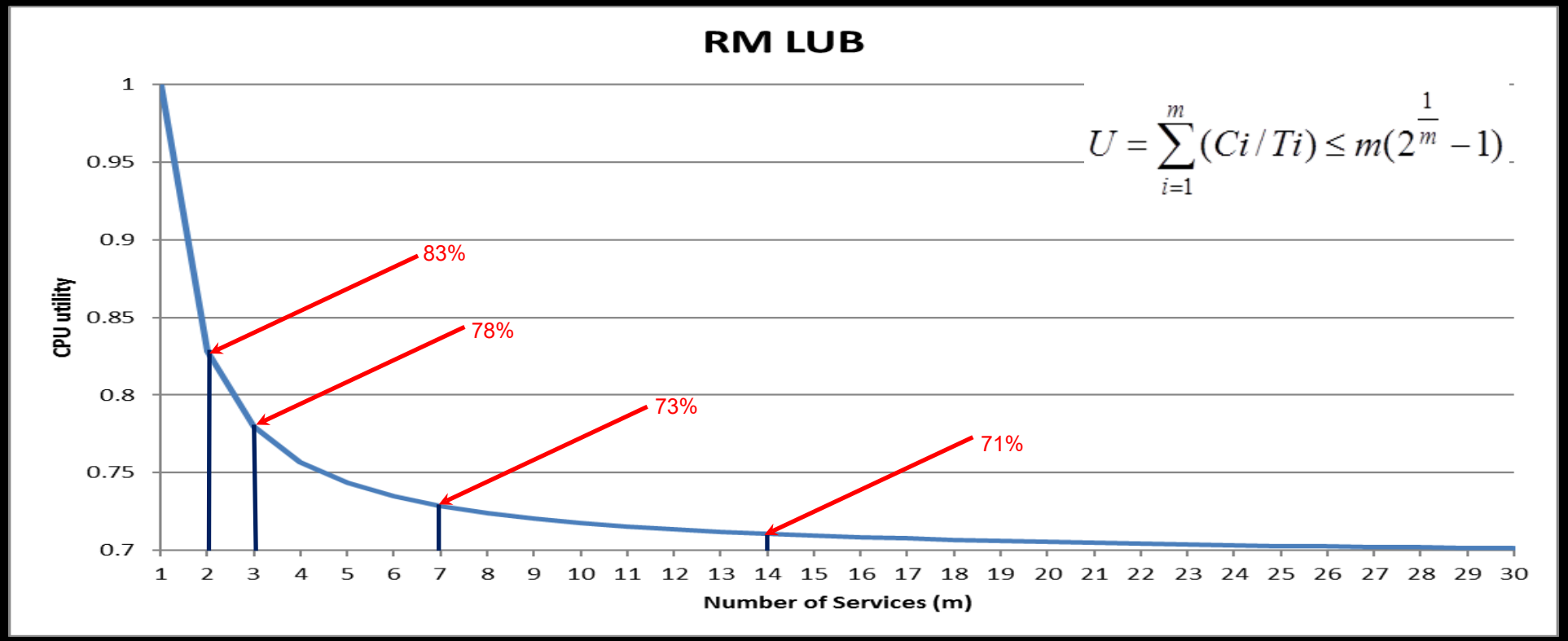
NASA's Apollo 11 moon landing was a human spaceflight feat, but also a dangerous journey. See some of the scariest moments of first manned moon landing.

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

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

REVIEW - What is the RM LUB?

- 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!



Outline - Learning Objectives

- CPU Scheduling Overview
  - Linux CFS [Completely Fair Scheduler]
  - Linux CFQ [Completely Fair Queue for I/O]
  - We really want an “unfair” scheduler for Real-Time
- RTOS Schedulers and POSIX RT – Linux, FreeRTOS, VxWorks
- RT Services – How to Define?
- Linux POSIX RT Thread Analysis and Debug
  - AMP - Cooperative Single Cores - Asymmetric Multi-processing
  - SMP - Multi-Core Load Balanced Symmetric Multi-processing
  - SCHED_FIFO
  - Synchronization
  - Issues from “Fair” Operating Systems vs. Same in “Unfair” RTOS

Key Terminology - See Glossary

- **Kernel 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.

- **RTP or 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 a task TCB for example) including a protected memory data segment (protected from writes by other processes).

- **Context Switch** - When a CPU is multiplexed (shared) by more than one *thread* of execution and the scheduler provides *preemption*, when the scheduler does preempt a thread in order to *dispatch* another, it must save state information associated with the currently executing thread (e.g. register values including *PC*) so that this *thread* can later be *dispatched* again to restore its thread of execution without a state error.

- **Preemption** - when the current thread executing on a CPU is placed back on the ready queue by the scheduler and state information saved so that a different thread can be allocated the CPU.

- **Dispatch** - when an RTOS scheduler selects a thread ready to run, restores state associated with the thread, and transfers execution control back to the thread's last PC if it was preempted earlier (or to its entry point if the thread is ready to run for the first time).

Key Terminology - See Glossary

- **AMP** – Asymmetric Multi-Processing – Multiple CPU Cores with Local memory and message passing interconnection network with application specific task workloads running on Multiple OS Instances or SMP with Thread Affinity so Threads are "assigned" to core, disabling load balancing for those threads

- **SMP** – Symmetric Multi-Processing – Multiple CPU Cores with common shared memory (uniform access) and One OS level interrupt handler mapping and task load balancing.

- **NUMA** – Non-Uniform Memory Access - Each CPU has local and non-local memory with different access latencies (e.g. Intel Xeon with QPI).

- **SMT** – Simultaneous Multi-threading – Super-scalar CPU micro-parallelism designed to support threads of execution.

- **Flynn's Taxonomy** – {Single Instruction, Multi-Instruction} x {Single Data, Multi-Data}

Fair vs. Unfair Scheduling

- CFS (Completely Fair Scheduler) in Linux – Fair!
  - Linux 2.6.23 Introduction of CFS
  - Fundamental Concepts of Fair Schedulers – “Everyone Get’s a Slice of Pie (Time), Some Just Get Slices More Often”
  - Requires PIT (Programmable Interval Timer) to Raise Time Based Interrupts – The Quantum
  - Scheduling Changes Can Be Made at Each Quantum “Tick”
  - All Tasks (Threads) Should Make Progress – Fair
  - Does Not Provide Predictable Response

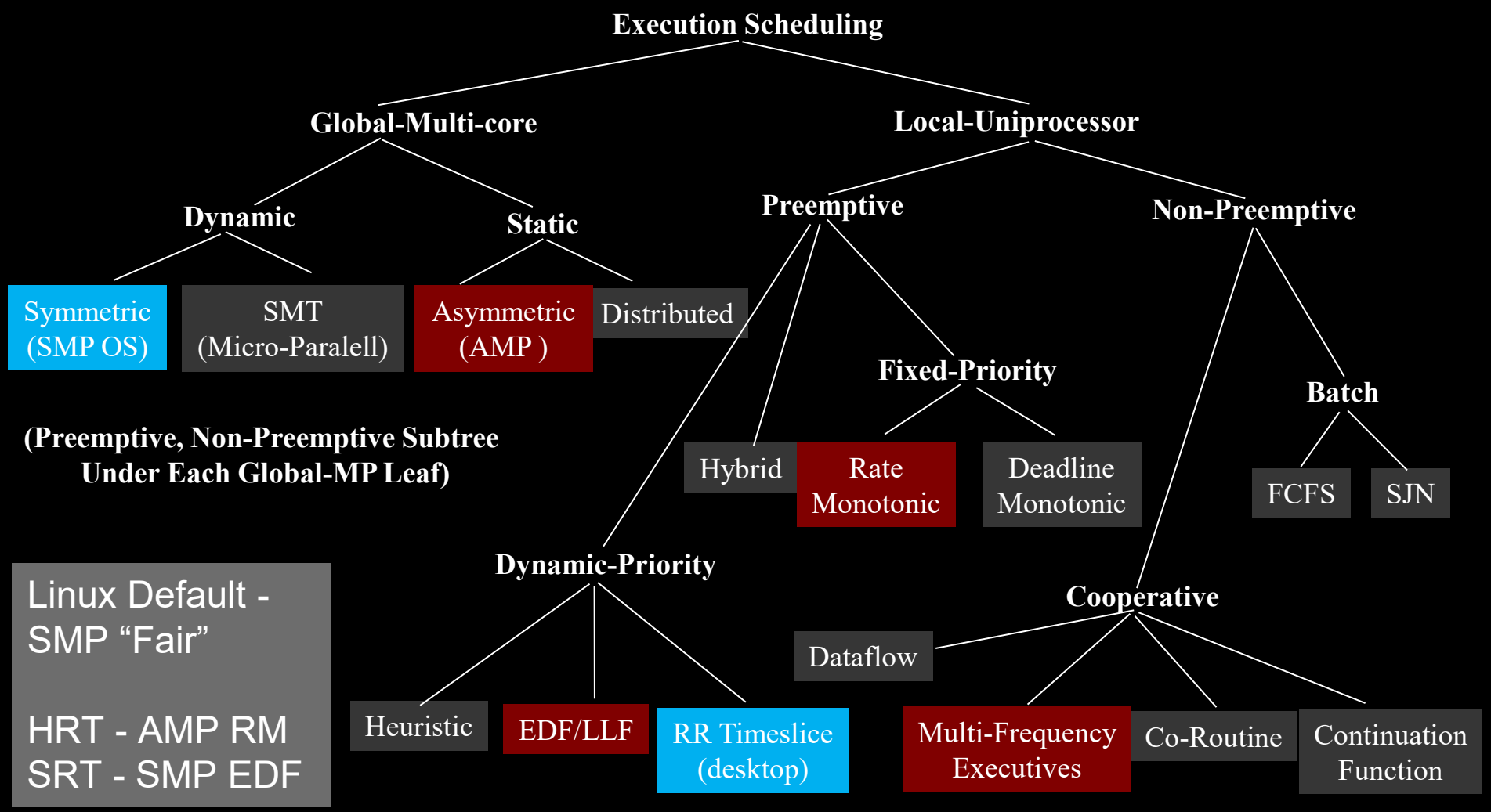
- Priority Preemptive Run-to-Completion RTOS – Unfair!
  - Tasks (Threads) Run at Priority Until:
    - Complete and Exit
    - Yield, Sleep, Activate Higher Priority Task or Block on Secondary Resource (System Call)
    - Interrupt Raised

Flynn's Computer Architecture Taxonomy

	Single Instruction	Multiple Instruction
Single Data	**SISD** (Traditional Uni-processor)	**MISD** (Voting schemes and active-active controllers)
Multiple Data	**SIMD** (e.g. SSE 4.2, GP-GPU, Vector Processing)	**MIMD** (Distributed systems (MPMD), Clusters with MPI/PVM (SPMD), AMP/SMP)

- GPC has become MIMD with SIMD Instruction Sets and SIMD Offload (GP-GPU)
  - NUMA vs. UMA (Trend away from UMA to NUMA or MCH vs. IOH)
  - SMP with One OS (Shared Memory, CPU-balanced Interrupt Handling, Process Load Balancing, Mutli-User, Multi-Application, CPU Affinity Possible)
  - MIMD - Single Program Multi-Data vs. Multi-Program Multi-Data
- Embedded has traditionally been SISD or MIMD AMP
  - Need for SMP in Embedded?
  - SIMD Offload and Acceleration Quite Useful (DSP, Image Processing, Error Correction Codes, Encryption, etc.)
- GP-GPU is a new class SPMD – Single Program, Multiple Data (thread gridding) – not covered in this course

CPU Scheduling Taxonomy



Elements of a Scheduling Class

- Scheduling Policy
  - How is Dispatch Decision Made?
  - Non-Preemptive, Cooperative or Batch (Hard Coded)
    - Main Loop Executives
    - Dataflow Schedule
    - Cooperative Sequential Processes, Co-Routines
  - Preemptive
    - Fixed Priority Encoding - RTOS
    - Dynamic Priority - RTOS with Programmed Priorities
    - Heuristic (Fuzzy Logic Scheduler, Heuristically Guided Iterative Repair)

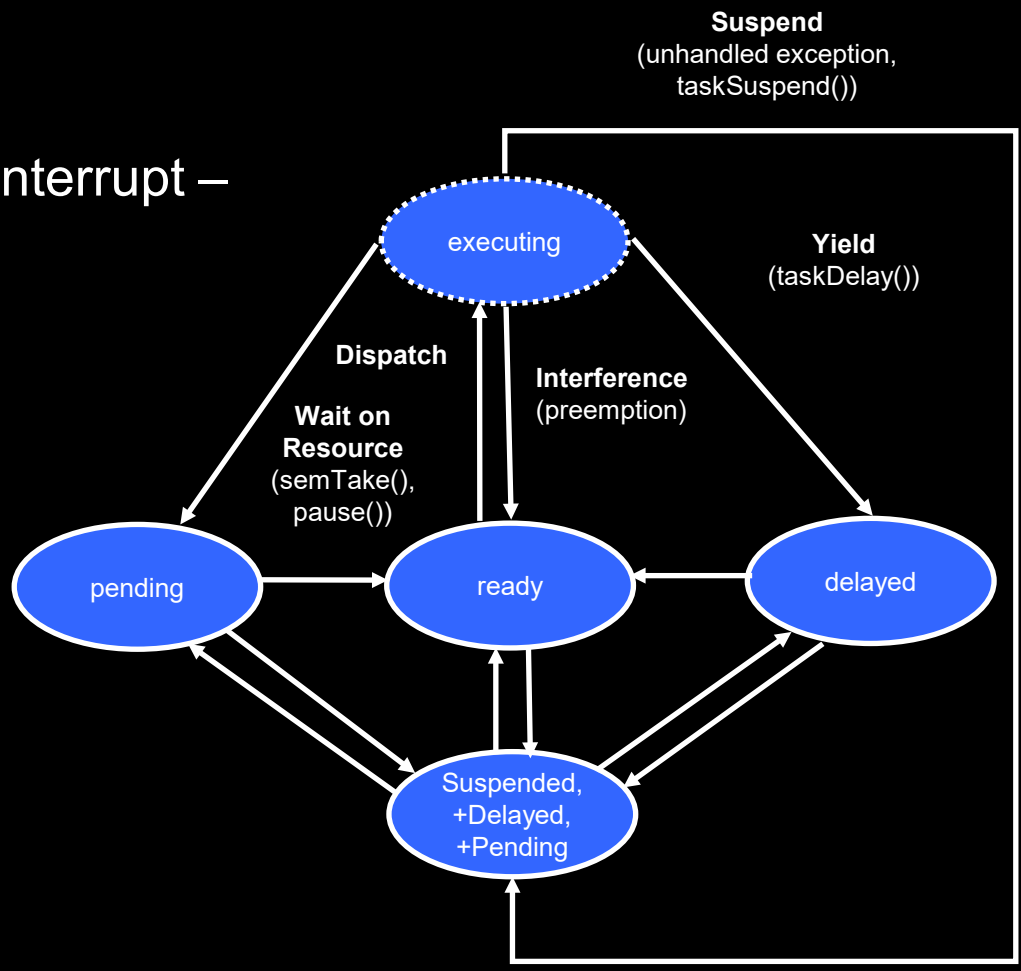
- Scheduling Feasibility Determination
  - Will Schedule Work?
  - Can a Set of Services Be Scheduled Given:
    - CPU Resources Available
    - I/O Resources Available
    - Memory Resources Available

- Ability to Tune Schedule
  - If Actuals Differ From Expected
    - WCET Expected vs. Observed
    - Maximum Release Frequency for a Service – Expected vs. Observed

Typical RTOS Scheduling Policies

- 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

Note: Different from Chapter 7 In VxWorks Kernel Programmer's Guide



RTOS Kernel Tasking / Linux NPTL

- An RTOS task or Linux NPTL thread is an Implementation of a Service
  - E.g. Anti-Lock Braking Digital Control
  - Created and Put Into Service, Awaits Event (Data Ready) For a Release
  - Memory Resident (TCB / Process descriptor (container), stack, code, data, semaphores, message queues)

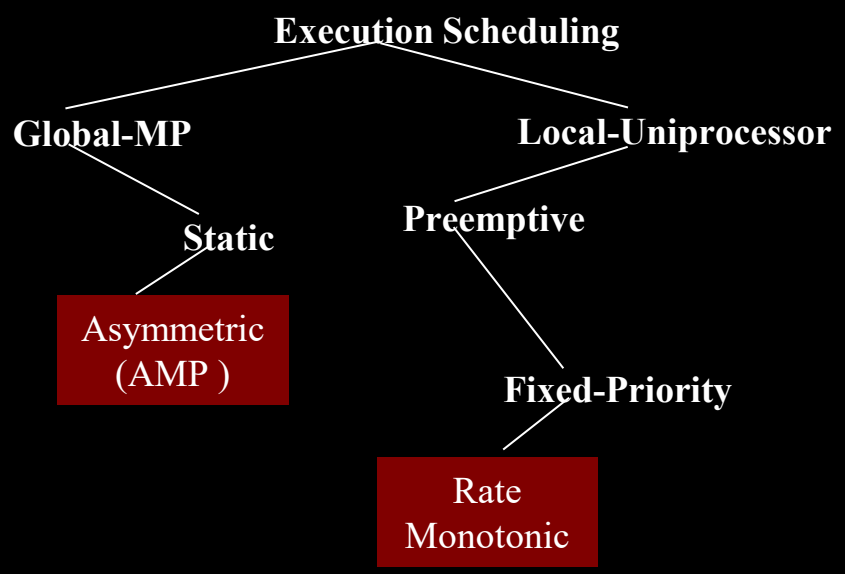
- RTOS or Linux Kernel Dispatcher Runs…
  - On Entry Point into Kernel Following Initialization
  - Loops (Idle) Until Task(s) in Ready Queue
    - Dispatches Highest Priority Task from Ready Queue
    - Idle is NOT a Task Context (Kernel Context)
  - Tasks or Threads become Ready and Are Dispatched When …
    - Sequencer or Interrupt Service Routines Release (E.g. semGive(), sem_post())
    - As a Result of a System Call (E.g. mq_send(), taskActivate(), pthread_create())
    - Following Return from an Interrupt Service Routine

- Dispatch Forces a Context Switch
  - Task/Thread State of Preempted is Saved (RTOS VX_FP_TASK for floats)
    - Coming from Idle, No Preemption, Simply Dispatch
    - Registers, Stack, PC (Program Counter or Instruction Pointer) Saved
  - Task State of Dispatched Task is Restored
  - PC Set to Last Text Address of Dispatched Task

- NPTL Documentation - manpage, NPTL paper (Red Hat - out of date)

RTOS Local Preemptive

- Fixed Priority RTOS/NPTL RM
- *Advantages…*
  - Deterministic Latency Control (Deadline = Period)
  - Well Established Theory, Tools, and Resources
  - Predictable Overload Deadline Failures
  - By Lehoczky, Shah, Ding Theorem Can Schedule up to 100%, By RM LUB to 70%
  - Known Latency/Throughput Tuning Methods (e.g. Period Transform, Sporadic Servers, Slack Stealers)
- *Disadvantages…*
  - RM LUB is Pessimistic
  - Sacrifices Throughput for Margin
  - Requires Overhead of Preemption
  - Context Switch and ISRs
  - Can Be Complex for Global-MP Architectures
    - No Load Balancing
    - Asymmetric Service Allocation to CPUs
    - Message Passing vs. Shared Memory



Assign Threads (Services) to Cores

Each awaits service request
Runs to completion
Awaits next request

Priority is RM Policy

Local Preemptive with Dynamic Priorities

- Dynamic Priority RTOS (EDF, LLF)

Advantages…

  - 100% Utilization and High Throughput

Disadvantages…

  - Hard to Implement – LLF is Easier than EDF, Catastrophic failure modes

- Linux/Unix “Fair” RR/Time Slice

Advantages…

  - Best Effort Fair - Everyone Gets a Slice Eventually

Disadvantages…

  - Huge Overhead on Quantum Preemption and Context Switch
  - Complex Scheduler - Do we really care about fairness anyway?

Liu and Layland Paper

Liu, Chung Laung, and James W. Layland.
"Scheduling algorithms for multiprogramming in a hard-real-time environment."
Journal of the ACM (JACM)
20.1 (1973): 46-61.

11,711 citations

Describes RM fixed priority

Describes RT Dynamic priority as “Deadline Driven” scheduling

Both are still used today

RM fixed - HRT
RT Dynamic priority - SRT

Why Use an RTOS? Or POSIX RT Threads?

- RTOS / NPTL Provides Common Scheduling Framework for:
  - Fixed Priority Preemptive
    - Rate Monotonic
    - Deadline Monotonic
  - Dynamic Priority Preemptive
    - EDF
    - LLF
  - Non-Preemptive Cooperative
    - Multi-Frequency Executive
    - Dataflow