EMBEDDED OPERATING SYSTEMS

Embedded Linux on Beaglebone Black

Scheduling

- Scheduling
  - Process of assigning ***resources*** to perform ***tasks***

- Resources
  - CPU, memory, network links, registers, buses
- Tasks
  - Processes, threads

- Carried out by a **Scheduler**

- Schedulers aim to:
  - Achieve optimum resource utilization
  - Allow resource sharing between multiple users
  - Achieve a target quality of service

Scheduler Goals

- A Scheduler may aim at one/more goals:
  - Maximizing throughput (work done per unit time)
  - Minimizing wait times (time for work to get started)
  - Minimizing latency (system response time)
  - Maximizing fairness (distributing CPU in a just manner)

- In **real-time** environments
  - Schedulers have to meet ***deadlines***
    - Time by which an activity **should** be complete

Scheduling in Linux

- The Linux scheduler
  - Has a queue of threads all ready to run
  - It schedules them on CPU(s) as they become available

- Scheduling is on ***thread basis***
  - Not on process basis!

- Each thread has a ***scheduling policy***
  - Either ***real-time*** or ***time-shared***

- Real-time threads have ***priority***
  - Concept of pre-emption
- Time-shared threads have ***niceness***
  - Concept of entitlement to CPU time

When does scheduler run?

- The Linux scheduler runs when:
  - A thread is blocked by calling sleep()
    - Or any other blocking call
  - A time-shared thread exhausts its time-slice
  - An interruption causes a thread to be unblocked

- Linux seeks a balance between
  - Fairness (by time-sharing)

    and
  - Determinism (by using real-time)

Time-sharing vs. Real-time

- Time-sharing
  - Based on principle of fairness
    - Each thread should get its fair amount of CPU time
    - No thread should be hog the system
    - If a thread runs for too long, it is put to the back of the queue
    - Threads doing more work are given more resources
  - Automatically adjust to varying workloads

- Real-time
  - Based on concept of priority
    - When the demand is for deterministic behavior
    - Minimal guarantee that thread will meet its deadline
    - Real-time threads pre-empt time-shared threads
    - Scheduler decides which to run based on priority
    - Most RTOS schedulers are built this way

Time-shared Policies

- CFS (Completely Fair Scheduler)
  - Scheduler used for time-sharing
    - Calculates a running tally of time consumed by a thread
    - Compares it with its **entitlement** and adjusts accordingly
    - A thread exceeding entitlement is suspended
  - Policies
    - SCHED_NORMAL (SCHED_OTHER)
      - Default policy, most threads run with this policy
    - SCHED_BATCH
      - Time granularity is larger, to reduce context switching overheads
    - SCHED_IDLE
      - Run only when no other threads run; lowest possible priority

Time-sharing: Niceness

- Some time-shared threads are more important
  - Indicated by its ***niceness*** property
    - A number between 19 and -20
    - 19 is really nice, -20 is really not nice
  - A thread is nice if its loads the CPU less!
  - Nice value can be changed for
    - SCHED_NORMAL and SCHED_BATCH threads
  - Only root user can change the niceness property
  - Bash has commands: ***nice, renice***

Real-time Policies

- Meant for achieving determinism
  - Run the highest-priority real-time when it is ready
  - Real-time always pre-empts time-shared
  - Priority in the range of 1 (lowest) to 99 (highest)

- Policies
  - SCHED_FIFO
    - Run to completion
    - Once started, thread runs unless
      - Preempted / blocked / terminated
  - SCHED_RR
    - Round robin between threads at same priority
      - By time slicing (usually 100ms)

Balance: Real-time – Time-share

- Problem with real-time
  - A **rogue** real-time process with **high priority**
  - Can prevent
    - All lower priority real-time threads
    - All time-shared threads
  - From running / getting resources
- System ***locks up*** and becomes erratic

- Linux scheduler solution:
  - Reserve 5% of time
    - For non-real-time threads
    - Configurable using
      - /proc/sys/kernel/sched_rt_period_us
      - /proc/sys/kernel/sched_rt_runtime_us
  - Use watchdogs

Choice of Policy

- For most systems, time-share works fine
- A thread may qualify for real-time if:
  - It has a strict deadline for deliverable
  - Missing the deadline compromises system efficacy
  - It is event-driven
  - It is not compute-bound (CPU hog)

- Choice of RT policy
  - Rate Monotonic Analysis (RMA)

THANK YOU!