Real-Time & Linux

Sources:
“Real-time Systems” by (Jane Liu, 2000) Ch 2
”HOWTO build a simple RT application” by the Linux Foundation
https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/application_base

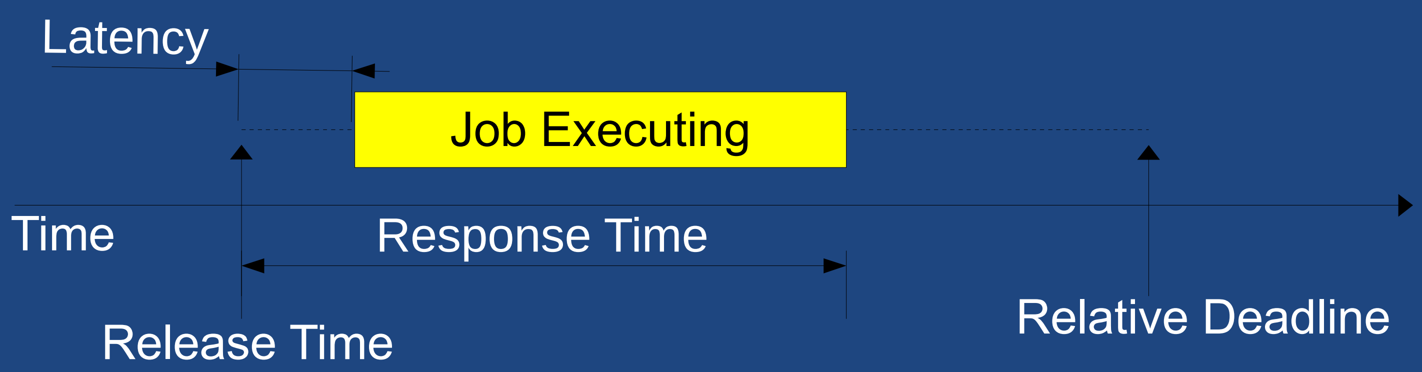
Topics

1) What is Hard vs Soft real-time?
2) How can we know when a task will run? (Deterministic Latency)

Hard vs Soft Real-Time

Timing Constraints

- Job
  - ..
  - Example: calculating the statistics over hundreds of light-intensity samples each second.
- Real-Time (RT) systems have jobs that must be started and completed by certain times.



- Job's timing constraint: its release time and relative deadline

Common Definitions

- Common definitions
  - Hard RT
    missing a timing deadline is considered
    a fatal flaw in the system.
    - Ex: collision avoidance system on a train yields a crash.
  - Soft RT
    missing a timing deadline yields
    degraded performance.
    - Ex: video playback yields a stutter
- Poor definition because it’s subjective:
  it depends on defining how fatal "late" is.

Our Definitions of RT

- Hard Realtime
  - User requires..
  - "Guaranteed Services"
    Mathematical/logical proof or exhaustive simulation required
  - Hard real-time is about..
- Soft Realtime
  - User only requires..

    (statistical analysis)
  - "Best effort Services"
    Ex: Average # missed deadline < 2 per minute.
  - Soft real-time is about..

Goals of RT

- What is latency?
  - Latency is..
  - We often care about critical tasks such as responding to high-priority interrupts (interrupt latency)
- Goal
  - low and deterministic latency
- Example:
  - Battery Management System:
    over-current detection triggers bank shutdown
  - Effect of non-deterministic latency in this example
  - [Draw a picture]

Hard RT: Scheduling Guarantees

- Example
  - Airplane flight control needs reliable timing to:
    - Read sensors
    - Compute “control-laws” to generate responses
    - Send responses to actuators
- OS guarantees
  - ..
- How?
  - Each new job comes with a duration and a deadline
  - System only allows new job if it can guarantee it can complete it by the deadline

Deterministic Latency

Deterministic Latency

Deterministic low latency RT requires:

- ..
  - support low-latency response
  - requires preemptible kernel with short critical sections
- ..
  - Avoid non-deterministic latencies on RT path
  - Use OS features for memory & scheduling

OS: Linux RT Patch

- Linux RT patch: PREEMPT_RT
  - Goal is to "minimize the amount of kernel code that is non-preemptible." (https://lwn.net/Articles/146861/)
- Patch has been cleaning up Linux kernel for years
  - Many of its features are on the "mainline" and have improved Linux for general uses (ex: better audio)
  - RT Patch makes kernel interruptible almost everywhere
- [DRAW]: syscall & context switch process

  1) App executes sys-call

  2) Kernel provides services; returns to app

  Any time: Kernel timer invokes context switch

Application Req for Deterministic Latency

- Step 1:
  - OS supports low latency
    (just saw that!)
- Step 2:
  - RT application takes steps to prevent nondeterministic latencies
  - Example sources of non-deterministic delays
    - memory faults
    - scheduling delays and context switches
    - priority inversion (later)

App 1) Memory Locking

- Swap Memory
  - A computer's memory (RAM) is divided up into pages. When running low on memory, OS swaps pages out to disk (swap file).
  - Even without swap file, OS can "swap" our executable code's memory page because it's already on disk.
- Page fault
  - If page is swapped to disk,..
- Problem
  - Page faults are..

App 1) Memory Locking solution

- Solution: Memory Locking
  - Ask the kernel to
    ..

```
/* Lock all current and future pages
   preventing being paged to swap */
if (mlockall( MCL_CURRENT | MCL_FUTURE )) {
    perror("mlockall failed");
    exit(-1); // Or handle error
}
```

- Run this code before any RT processing starts

SOURCE: Memory for Real-time Applications: the Linux Foundation
https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/memory#memory_locking

App 2) Stack Memory

- Each thread has its own stack in memory.
  - If spawning many threads, can..
- Problem
  - If all pages are locked in RAM, we must ensure we don't exhaust available memory.
  - Spawning new thread allocates new memory; if locked to RAM then triggers a page fault.
- Solution
  - ..
  - Understand memory use of each thread, and.. (default ~8mb)

SOURCE: Memory for Real-time Applications: the Linux Foundation
https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/memory#memory_locking

App 2) Stack Memory

- Set thread stack size:

```
static void create_rt_thread(void)
{
   pthread_t thread;
   pthread_attr_t attr;

   /* init to default values */
   if (pthread_attr_init(&attr))
      error(1);

   /* Set a specific stack size   */
   int size = PTHREAD_STACK_MIN + MY_STACK_SIZE;
   if (pthread_attr_setstacksize(&attr, size))
     error(2);

   /* Finally start the actual thread */
   pthread_create(&thread, &attr, rt_func, NULL);
}
```


SOURCE: Memory for Real-time Applications: the Linux Foundation
https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/memory#memory_locking

App 3) Dynamic Memory

- Problem
  - Dynamically allocating or freeing memory can ..
- Solution
  - RT critical paths should not dynamically allocate or free memory.
  - Instead, preallocate all memory for RT paths:
    - init() functions dynamically allocate memory
    - Non-RT code allocate memory, pass pointer to RT path
    - ..

App 4) Priorities and Scheduling

- OS schedules tasks (jobs) based on its scheduling algorithm and task priority.
- Problem
  - Some tasks are more time critical, and must be run sooner than others.
- Solution
  - Assign each task a reasonable priority
  - ..
- More to come on this!

Summary

- Real-time
  - Hard RT requires scheduling guarantees
  - Soft RT requires a best-effort with low latency
- OS Features
  - Preemptable kernel with priorities for tasks
- App Features
  - Memory locking to prevent page faults
  - Task stack memory management to reduce memory pressure
  - No dynamic memory allocation/free on RT path
  - Task priorities