Task Schedulers

Asked 16 years, 3 months ago Modified 15 years, 11 months ago

Viewed 917 times

4

Had an interesting discussion with some colleagues about the best scheduling strategies for realtime tasks, but not everyone had a good understanding of the common or useful scheduling strategies.

For your answer, please choose one strategy and go over it in some detail, rather than giving a little info on several strategies. If you have something to add to someone else's description and it's short, add a comment rather than a new answer (if it's long or useful, or simply a much better description, then please use an answer)

- What is the strategy - describe the general case (assume people know what a task queue is, semaphores, locks, and other OS fundamentals outside the scheduler itself)
- What is this strategy optimized for (task latency, efficiency, realtime, jitter, resource sharing, etc)
- Is it realtime, or can it be made realtime

Current strategies:

- Priority Based Preemptive
- Lowest power slowest clock

-Adam

operating-system kernel scheduling

Share

Improve this question

Follow

edited May 23, 2017 at 12:01

Community Bot

1 • 1

asked Sep 8, 2008 at 15:40

Adam Davis

93.5k • 60 • 271 • 333

2 Answers

Sorted by: Highest score (default)

8

+100

As described in a paper titled Real-Time Task Scheduling for Energy-Aware Embedded Systems, Swaminathan and Chakrabarty describe the challenges of real-time task scheduling in low-power (embedded) devices with multiple processor speeds and power consumption profiles available. The scheduling algorithm they outline (and is shown to be only about 1% worse than an optimal solution in tests) has an interesting way of scheduling tasks they call the LEDF Heuristic.

From the paper:

> *The low-energy earliest deadline first heuristic, or simply LEDF, is an extension of the well-known earliest deadline first (EDF) algorithm. The*

> *operation of LEDF is as follows: LEDF maintains a list of all released tasks, called the "ready list". When tasks are released, the task with the nearest deadline is chosen to be executed. A check is performed to see if the task deadline can be met by executing it at the lower voltage (speed). If the deadline can be met, LEDF assigns the lower voltage to the task and the task begins execution. During the task's execution, other tasks may enter the system. These tasks are assumed to be placed automatically on the "ready list". LEDF again selects the task with the nearest deadline to be executed. As long as there are tasks waiting to be executed, LEDF does not keep the processor idle. This process is repeated until all the tasks have been scheduled.*

And in pseudo-code:

```
Repeat forever {
    if tasks are waiting to be scheduled {
        Sort deadlines in ascending order
        Schedule task with earliest deadline
        Check if deadline can be met at lower
speed (voltage)
        If deadline can be met,
            schedule task to execute at lower
voltage (speed)
        If deadline cannot be met,
            check if deadline can be met at higher
speed (voltage)
        If deadline can be met,
            schedule task to execute at higher
voltage (speed)
```

```
        If deadline cannot be met,
            task cannot be scheduled: run the
exception handler!
    }
}
```

It seems that real-time scheduling is an interesting and evolving problem as small, low-power devices become more ubiquitous. I think this is an area in which we'll see plenty of further research and I look forward to keeping abreast!

Share Improve this answer

Follow

answered Sep 8, 2008 at 16:09

Sean

4,670 ● 1 ● 21 ● 18

2

One common real-time scheduling scheme is to use priority-based preemptive multitasking.
Each tasks is assigned a different priority level.
The highest priority task on the ready queue will be the task that runs. It will run until it either gives up the CPU (i.e. delays, waits on a semaphore, etc...) or a higher priority task becomes ready to run.

The advantage of this scheme is that the system designer has full control over what tasks will run at what priority. The scheduling algorithm is also simple and should be deterministic.

On the other hand, low priority tasks might be starved for CPU. This would indicate a design problem.

Share Improve this answer

Follow

answered Sep 16, 2008 at 17:37

Benoit

38.9k 24 85 117