Real-Time Operating Systems and Languages

Real-Time and Embedded Systems (M)

Tutorial 4

UNIVERSITY
of
GLASGOW

Tutorial Outline

- Review of problem set 3
- Review of lectures
- Discussion and examples
- Question & answer

Review of Problem Set 3

- Question 1: Briefly explain the differences in budget consumption and replenishment rules between a deferrable server and a polling server. Describe how this can be expected to change the response times of any aperiodic jobs [3 marks].

Review of Problem Set 3

- Question 2: Consider a system of three periodic tasks $T_1 = (3, 1)$, $T_2 = (4, 0.5)$ and $T_3 = (10, 2)$. Demonstrate that this system can be scheduled using the rate monotonic algorithm and the earliest deadline first algorithm. [3 marks]

Review of Problem Set 3

- Question 3: The system from question 2 must also support three aperiodic jobs:
  - A_1 released at time 0.5
  - A_2 released at time 12.25
  - A_3 released at time 17.
- Each aperiodic job executes for 0.75 units of time. The system is scheduled using the rate monotonic algorithm, with a server task, $Ts = (5, 0.5)$ to schedule these aperiodic jobs.
- Show the resulting schedules for sufficient time to illustrate how the aperiodic tasks are scheduled and to demonstrate correctness (or otherwise) of the schedule, and calculate the response times for each of the aperiodic tasks, if the server is scheduled as a) a polling server; or b) a deferrable server.

Review of Problem Set 3

- Question 4: A simple sporadic server can be defined for EDF systems, as well as for rate monotonic systems. Explain the difference between the consumption and replenishment rules for a simple sporadic server in a deadline driven system, compared to a rate monotonic system. [3 marks]

Review of Lectures

- Implementing real time systems
  - Key concepts and constraints
  - System architectures:
    - Cyclic executive
    - Microkernel with priority scheduler
- Implementing scheduling algorithms
  - Jobs, tasks, and threads
  - Priority scheduling of periodic tasks
    - Rate monotonic
    - Earliest deadline first
  - Priority scheduling of aperiodic and sporadic tasks

Review of Lectures

- Real-time operating systems and languages
  - Clocks and timing
    - Clocks and the concept of time
    - Delays and timeouts
  - Scheduling
  - C and POSIX, Real-time Java and Ada
- Real-time system as part of a larger system
  - Two level scheduler
    - Discussion of concepts
    - Advantages and disadvantages
    - Case study: RTLinux
- Real-time on general purpose systems
  - Need for flexible applications
  - Implementation strategies
  - Scheduling

Key Learning Outcomes

- Understanding how real-time operating system schedulers are implemented; different implementation styles and their trade-off
- Understanding of how to use real-time operating systems; their features and the limits of practical scheduling
- Understanding of why flexible applications are needed; and some of the issues to consider when designing such applications

Discussion and Examples (1)

- Many real-time systems comprise tasks which have to be executed periodically. An operating system can support such tasks through the abstraction of a “periodic thread”, or they can be implemented using a standard thread that loops with an appropriate period.

  Compare advantages and disadvantages of the two approaches.

Discussion and Examples (2)

- The POSIX 1003.1b API provides the ability to schedule processes at a range of fixed priority levels, according to either FIFO or round-robin scheduling disciplines. Using a diagram of the scheduler's priority queues, explain how both real-time and non-real time tasks may be supported by a single operating system. Your diagram should show tasks in the ready, executing, sleeping and blocked states, and you should explain when transitions between states occur

Discussion and Examples (3)

- Explain how the presence of non-real-time tasks might disrupt the operation of real-time tasks running on the same system.

Discussion and Examples (4)

- You are debugging a soft real time networked multimedia application running on Linux. The application uses the POSIX real time extensions to select high priority FIFO scheduling, so that it has priority over non-real time tasks in the system and can achieve smooth video playback.

  Unfortunately, the application is faulty and goes into an infinite loop when trying to decode certain video sequences, never blocking or yielding the processor.

  What would be the effects of this on other tasks running on the system? How would you debug the program?

Any Further Questions?