Introduction to Lab 3
Response Time Analysis using \texttt{FpsCalc}

Jakaria Abdullah

28 September 2015
Lab 3: Response Time Analysis using \texttt{FpsCalc}

- **Lab goals:**
  - Practice response time analysis
  - Manual calculation, critical instant charts, tool \texttt{FpsCalc}
  - Integrate context switch overhead, blocking, jitter

- **Lab preparation:**
  - Lab will be done on Thu, 01.10., in rooms 1515
  - Have a look at the lab homepage
    \url{http://www.it.uu.se/edu/course/homepage/realtid/ht15/lab3}
  - Possibly print out assignment description (11 pages PDF)

- **Lab report:**
  - Answers (incl. diagrams) to the questions
  - To lab 3 submission page, studentportal
  - \textit{Deadline: Thu, 07.10., 23:59}
Clarifying Concepts

Schedulability Analysis

- *General problem* for real-time systems
- Given: Task set $\tau$, scheduling strategy $S$ (like RM or EDF)
- Question: Will all tasks always meet their deadlines?

Utilization Bound

- *One particular method* to do schedulability analysis
- Based on system’s utilization bound $U := \sum_{i=1}^{n} \frac{C_i}{T_i}$
  - For EDF: $U \leq 1 \iff \tau$ schedulable (sufficient and necessary)
  - For RM: $U \leq n(2^{1/n} - 1) \implies \tau$ schedulable (only sufficient!) (part 1)

Response Time Analysis

- *Another method* to do schedulability analysis (and more)
- For each task $\tau_i$, calculate its worst case response time $R_i$
  - If $R_i \leq D_i$ for all $\tau_i \in \tau$, then $\tau$ schedulable
  - Can be a pessimistic bound, then only sufficient (parts 2-5)
Response Time Analysis

- Given task set \( \tau \), how to calculate response times \( R_i \)?
- For *fixed priority scheduling* (including RM or DM):

\[
R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j
\]

- What do these parts mean?
  - \( C_i \) is \( \tau_i \)'s own computation time (bound)
  - \( \sum_{j \in hp(i)} \) is sum over all *higher priority* tasks
  - \( \left\lceil \frac{R_i}{T_j} \right\rceil \) is number of preemptions of \( \tau_j \) over \( \tau_i \)
  - \( \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \) is total time \( \tau_j \) preempts \( \tau_i \)
- Formula gets more complex considering overheads, blocking and jitter
- ... and is *recursive!*
RTA: Solving The Recursive Formula

- Want to find **fixed point** $R_i$ such that:

\[ R_i = C_i + \sum_{j \in hp(i)} \left\lfloor \frac{R_i}{T_j} \right\rfloor \cdot C_j \]

- Can be done **iteratively**:
  1. Start with $R_i^0 := 0$
  2. Iterate $R_i^{k+1} := C_i + \sum_{j \in hp(i)} \left\lfloor \frac{R_i^k}{T_j} \right\rfloor \cdot C_j$
  3. . . until no change
  4. Fixed point found $\implies$ happy 😊

- This is tedious work, let’s use a computer for that!

- **FpsCalc** is a tool for this purpose
  - Rest of introduction: How to use FpsCalc
Available on all Solaris Servers in the department:
http://www.it.uu.se/datordrift/faq/unixinloggning

How to call it:

```
/it/kurs/realtid/bin/fpscalc < program.fps [-v]
```

- Note the “<”!
- `-v` for more verbose output (debugging etc.)
- Save result in a file:
  
  ```
  fpscalc < program.fps > result.txt
  ```

More info:

http://www.idt.mdh.se/~ael01/fpscalc/
**FpsCalc**: Program structure

- **FpsCalc** programs contain (one or more) system blocks
- Inside each system block:
  - One declarations block
  - One semaphores block (optional)
  - One initialise block
  - One formulas block

**Example: FpsCalc program**

```plaintext
system my_RM_system {
  declarations {
    ...
  }
  initialise {
    ! This is a comment
    ...
  }
  formulas {
    ...
  }
}
```

Jakaria Abdullah
Lab 3: RTA
28 September 2015
**FpsCalc: declarations Block**

- Declare tasks and variables
- Variable types:
  - scalar: Just one value
  - indexed: array of scalars, indexed by task names
  - priority: array of task priorities
  - blocking: array for blocking times (because of semaphores)
    - Only one variable each of priority and blocking allowed
- Names i and j are reserved

**Example: declarations Block**

```plaintext
declarations {
    tasks A, B, C, D;
    scalar AuxVar;
    indexed Period, Deadline, CompTime, RespTime;
    blocking BlockingTime;
    priority Prio;
}
```
**FpsCalc: semaphores Block**

- Specify *which* semaphore used *by whom* for *how long*
- When set, blocking times are calculated automatically

**Example: semaphores Block**

```plaintext
1 semaphores {
2    semaphore (S1, A, 3.0);
3    semaphore (S1, B, 1.0);
4 }
```
**FpsCalc: initialise Block**

- Assign initial values to variables
- If not specified: Implicitly 0

**Example: initialise Block**

```plaintext
 initialise {
    AuxVar = 5.0;
    Deadline[A] = 10.0;
    Deadline[B] = 12.0;
    CompTime[i] = 3.0;  ! For all tasks
}
```
FPSCalc: formulas Block

- The “program”: Recursive formulas
- Left hand side: Variable, possibly indexed by \( i \)
- Right hand side: use “+”, “-”, “*”, “/” and:
  - \( \sigma(hp, expression) \): Sum over higher priority tasks, \( j \)-indexed
    - “\( \sigma(hp, R[i]/T[j]) \)” means: \( \sum_{j \in hp(i)} R_i/T_j \)
  - Same for \( ep \), \( lp \) and all (equal priority, lower priority, all tasks)
  - \( \text{ceiling}(expression) \): For ceiling function \((\lceil \cdot \rceil)\); same for \text{floor}
  - \( \text{min}(exp1, exp2) \): For minimum function; same for \text{max}

Example: formulas Block

```python
1 formulas {
2     RespTime[i] = CompTime[i] + BlockingTime[i] +
3         \sigma(hp, \text{ceiling}(RespTime[i]/Period[j])) * CompTime[j];
5 }
```
Lab Assignment

- **Part 1: Rate Monotonic Scheduling**
  - Work with the utilization bound
  - Get used to FpsCalc

- **Part 2: Priority Orders**
  - Compare RM, DM and other orders

- **Part 3: Context Switch Time**
  - Extend formula with context switch overhead

- **Part 4: Blocking**
  - Extend formula with blocking time
  - Model semaphores and work with synchronization protocols

- **Part 5: Jitter**
  - Extend formula with jitter

**Some hints:**
- Focus is on the theory and concepts
  - FpsCalc is just a helping tool to make things easier
- Use a print-out of the assignment description
The End

Questions?