Real-Time Programming

Lecture 1

Farhang Nemati

Spring 2016

Real-Time Programming

- Teacher:
  - Farhang Nemati
    - Room T2232
    - Email: farhang.nemati@oru.se
- Lab Assistant :
  - Farhang Nemati
- Where to find course related stuff:
  - Blackboard

About Me

- BSc in Computer Engineering, University of Tehran, Iran, 1998
- Work in Industry, Iran, 1998-2003
- MSc in Computer Science, Uppsala University, Sweden, 2006
- PhD in Real-Time Systems, Mälardalen University, Sweden, 2012
- Software Engineer, Atlas Copco, Sweden, 2012-2013
- Software Specialist, BMW, Germany, 2013-now
- Senior Lecturer, Örebro University, Sweden, 2015-now

Lectures and Labs

- 14 lecture sessions
  - Each session 2 * 45 minutes
- 10 guided labs sessions
  - Each session 2 * 45 minutes
- Lab: T004
- 15 self-practice sessions. Lab T004 is booked for you during these sessions
- If you are not registered yet, talk to Jenny Tiberg as soon as possible!
- Find your group mate for labs now and register in one of the groups in Blackboard.

Criteria to Pass the Course

- Approved written exam
  - 4-6 questions, 40 points
  - Required: 20 points for grade 3, 30 for grade 4, and 35 for grade 5
- Approved labs
  - There are 4 labs
  - Required: all 4 labs are handed in, in time

Literature

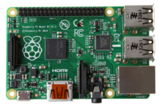
- Real-Time Systems and Programming Languages. Alan Burns and Andy Wellings
- (In Swedish) Realtidsprogrammering, ISBN: 91-44-03130-0. Ola Dahl
  - Only electronic version exists
- (Optional) Hard Real-Time Computing Systems. Giorgio Buttazzo
- Lecture slides
- Lab instructions

Preliminary Course Contents

- Course Introduction
  - Lab environment, programming tools, etc.
- Introduction
  - What are Embedded Systems?
  - What are Real-Time Systems?
- Programming for Embedded Systems and Real-Time Systems
  - Concurrent programming
  - Inter-process communication, Synchronization, etc.
  - Execution Time analysis
- Scheduling Algorithms
- Resource Sharing Protocols
- Design and Analysis of Real-Time Systems
  - Petri net
  - Response time analysis
- (Optional) Real-Time Systems on Multi-processors

Embedded Systems

Embedded Systems



Embedded Systems

- Interacting with physical world
- Long time running
- Concurrency
- Limited resources
- Limited access

Concurrency in Embedded Systems

- Increase reaction time to physical environment
- Parallelism, i.e., run on multiple processors
- Independence among different tasks, e.g., periodic execution of some tasks

Real-Time Systems

- Functional correctness as well as timing constraints have to be satisfied, e.g., deadline for doing some operation has to be met.

Types of Real-Time Systems

- Hard Real-Time System
  - Safety-Critical Systems
- Soft Real-Time Systems

Real-Time Systems

- Most of the embedded systems have timing constraints (Why?)
- Because of interaction with the physical world
- A Real-Time system is usually divided into tasks
  - Task?
- Execute tasks concurrently
- Who decides which task to run?
  - A scheduler
- How?
  - Fairness, priority, etc.

Real-Time Task

- A task is a set of operations that execute sequentially
  - A task can have a priority
  - Has an execution time: the total time it takes for the task to perform its operation
  - It might have a period (cycle)
  - It HAS a deadline: the latest time by which its execution has to be finished

Real-Time Task

- Types of Real-Time Tasks:
  - Hard task: its deadline should not be missed; missing a deadline might have severe or catastrophically consequences.
  - Soft task: missing deadline can be tolerated. Results after deadline can still be useful. However the performance could be degraded.
  - Firm Task: missing deadline can be tolerated. Results after deadline are useless.
  - A real-time system may contain a mixture of all types of tasks.

Execution of Real-Time Tasks

- Monolithic
  - Only one program. It runs all tasks sequentially
- Real-Time Operating System (RTOS)
  - An operating system suitable for real-time systems runs the tasks concurrently
  - E.g., VxWorks, RTLinux, FreeRTOS, etc.
- Real-Time Language (RTL)
  - A programming language suitable for real-time systems runs the tasks concurrently
  - E.g., Ada-95

Example1

- Assume we have a system that consists of 2 tasks which periodically perform some actions.
- Monolithic solution:

```
int mainProgram()
{
   while(1)
   {
       operationsOfTask1
       operationsOfTask2
       sleep(forSomeTime);
    }
}
```

Example1

- RTOS solution:

```
int mainProgram()
{
    createTask(task1, priority1);
    createTask(task2, priority2);
}
void task1()
{
   while(1)
    {
        operationsOfTask1;
        delay(forSomeTime);
    }
}
void task2()
{
   while(1)
    {
       operationsOfTask2;
       delay(forSomeTime);
    }
}
```

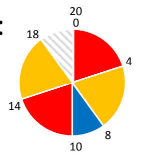
Example1

- RL solution, Ada-95:

```
procedure mainProgram is
task TASK1;
... <declarations> ...
task body TASK1 is
begin
... < operationsOfTask1 > ...
end TASK1 ;
task TASK2
... <declarations> ...
task body TASK2
begin
... < operationsOfTask2 > ...
end TASK2;
begin
null;
end mainProgram;
```

Example2; Car Control Tasks

- Assume we have 3 tasks that periodically control some parts of a car:
  - `engine:` runs every 10ms for 4ms 🟥
  - `speed:` runs every 10ms for 4ms 🟨
  - `fuel:` runs every 20ms for 2ms 🟦
  - How do we know the execution time of each task?
- Each task has to finish its execution by end of its period (deadline)
- Monolithic solution:



- What happens if each task `engine` and `speed` takes 5ms to run?

Car Control Example, Monolithic Solution

```
//The source code in this example does not correspond to the syntax of any language!
int main()
{
    while(1)
    {
        //runs for 4 ms
        controlEngineOperations;
        //runs for 4 ms
        controlSpeedOperations;
        //runs for 2 ms
        controlFuelOperations;
        //runs for 4 ms
        controlEngineOperations;
        //runs for 4 ms
        controlSpeedOperations;

        delayMillisecond(2);
    }
}
```

Car Control Example, RTOS Solution

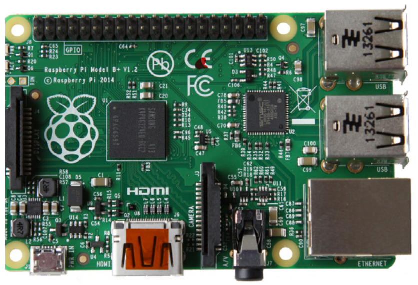
```
int controlEngine()
{
    while(1)
    {
        controlEngineOperations; //takes 4 ms
        delayMillisecond(6);
    }
}

int controlSpeed()
{
    while(1)
    {
        controlSpeedOperations; //takes 4 ms
        delayMillisecond(6);
    }
}
```

Car Control Example, RTOS Solution

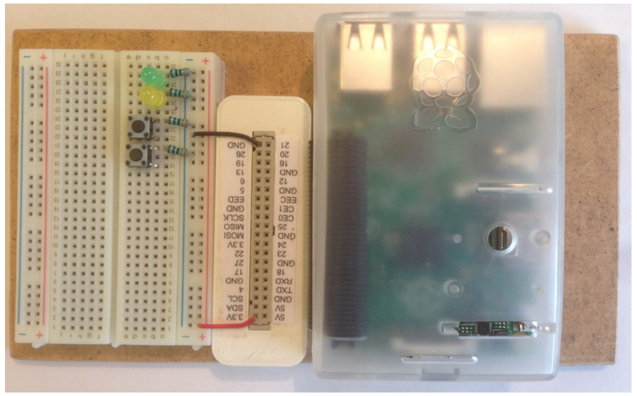
```
int controlFuel()
{
    while(1)
    {
        controlFuelOperations; //takes 2 ms
        delayMillisecond(18);
    }
}
int main()
{
    int engine, speed, fuel;
    engine = createTask(controlEngin, 10);
    speed = createTask(controlSpeed, 11);
    fuel = createTask(controlFuel, 12;

    waitForTasksToFinish(engine);
    waitForTasksToFinish(speed);
    waitForTasksToFinish(fuel);
}
```

Lab Equipment and Tools

- Raspberry Pi (Model B+)
  - 700MHz Broadcom BCM2835 CPU
  - 512 MB SDRAM @ 400MHz
  - 10/100 Ethernet RJ45 on-board network
  - Full size HDMI
  - 4 USB ports
  - Micro SD slot
  - 40 pins GPIO header

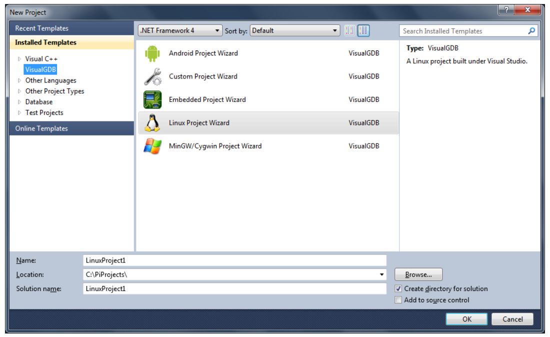


Lab Equipment and Tools



Development Environment

- Microsoft Visual Studio
- VisualGDB (Linux Edition)



New Project

Recent Templates
Installed Templates
- Visual C++
- VisualGDB
- Other Languages
- Other Project Types
- Database
- Test Projects

Online Templates

.NET Framework 4 — Sort by: Default — Search Installed Templates

Android Project Wizard	VisualGDB
Custom Project Wizard	VisualGDB
Embedded Project Wizard	VisualGDB
Linux Project Wizard	VisualGDB
MinGW/Cygwin Project Wizard	VisualGDB

Type: VisualGDB
A Linux project built under Visual Studio.

Name: LinuxProject1
Location: C:\PiProjects\
Solution name: LinuxProject1

[✓] Create directory for solution
[ ] Add to source control

Labs

- Lab 1: Hello World, Blinking LED
  - Familiarize yourself with programming and debugging the system
  - Using electronic devices, start with using LEDs
- Lab 2: Buttons, Breathing LEDs
  - Using Buttons
  - Pulse Width Modulation (PWM)
- Lab 3: Concurrency
  - Multiple tasks (threads), task priorities
  - Synchronization
- Lab 4: Car plate, Make a Car
  - 2 motors, buttons, lights, power supply etc.
  - Design of a complex system with concurrency constraints