MUNI
FI

IA158 - Scheduler

Jan Koniarik

March 26, 2020

Agenda

1. Project information
2. Scheduler requirements
3. API introduction
4. Example task

Project

- Part of the assessment of the course is a project - scheduler.
- Your goal is to write a scheduler in C.
- It has to schedule our three tasks and one of your own - four in total.
  - You can get creative with your task.
- Your scheduler has to meet all the requirements, and you have three attempts at submission.
  - The personal presentation will not be required.
- You are allowed to work in pairs, but we can not recommend it now

Changes

The project requirements changed: You do not have to use embedded hardware, and the presentation of the project is not required.

Scheduler requirements

Assumptions:

- Jobs are non-preemptible
- Tasks are periodic
- One processor
- No resources/priorities/precedence
- Synchronized

Requirements:

- Schedule our three tasks and one of your own
- The schedule has to be valid
- Overrun detection (not prevention)
  - It should be clear to potential users that overrun happend but do not stop the execution.
- The schedule is not hardcoded

Skeleton

There will be a simple C skeleton in the school information system with CMake as a build system.
The three given tasks are schedulable without a problem, and we believe it is easy to add the fourth task.
Your solution has to schedule all four tasks correctly, in case you have an error - the project will not be accepted, and you fail.

Preemptability

But the non-preemptable scheduling is NP-hard

We designed our tasks in a way that it is solvable with an algorithm that expects preemptable jobs. As for your task, it is your burden to design it in a way that it is not necessary. [1]

[1] But if you know how to implement it, you can do it.

Implementation

- I will check your implementation of the scheduler - write it in a way that I can understand it.
  - Code quality is a necessity for a good scheduler, as the code has to be easy to understand and debug.
  - You are free to structure the code however you see fit.
- Given that your result should be a desktop application, I will not restrict you from the usage of dynamic memory, but it should not be necessary.

API introduction

Interface

- The interface is in file *api.h* provided to you in the skeleton folder.
- Our tasks are defined in *tasks.h/tasks.c* in the skeleton, feel free to inspect them.
- Do not submit modified *tasks* files. We will check them on submission and replace them.

Tasks

The three provided tasks have these properties, all values are in milliseconds: [2]

led	period: 250, deadline: 50, max. exec. time: 1
uart	period: 251, deadline: 251, max. exec. time: 40
fac	period: 1499, deadline: 249, max. exec. time: 40

The led task will be implemented as an example in this seminar. However, our implementation is in the project for the project.

[2] We will not change the values for tests. Hardcoded solutions for these numbers will be denied.

Example task

Assignment

1. Download project skeleton IA158_skeleton.zip from IS
2. Check that you can compile it (find how to use CMake properly, you can do that)
3. Open *IA158_sched.c* file

Step 1: Write job function

The led task blinks the LED present on the board. For the desktop version, we will use print as a replacement.

```
#include "api.h"

uint32_t i = 0;

void led_job(void *) {
    printf("Status of green led: %i", i);
    i = (i + 1) % 2;
}
```

Using a global variable is ugly, but we will live with that for now.

Step 2: Write an instantion of task structure

We want the job of blinking LED to be executed at the 250ms period. This gives us 4 blinks per second. The maximum execution time is estimated at 1 ms. [3]

```
struct task LED_TASK_SIMPLE = {.period = 250,
                               .max_execution_time = 1,
                               .relative_deadline = 50,
                               .job = &led_job,
                               .data = nullptr};
```

[3] All time units are in milliseconds

Step 3: Write simple scheduler

As an example, we can show a simple execution of one task - simple while loop. In the example, we use busy wainting to ensure the period the task has specified. [4]

```
void run_single_task(struct task *task_ptr) {
    while (true) {
        uint32_t end_time = time() + task_ptr->period;
        task_ptr->job(task_ptr->data);
        SysTick_DelayMs(end_time - SysTick_GetTime());
    }
}
```

[4] SysTick* are our functions provided in sys_tick.h/sys_tick.c

Assignment

- Implement all three steps in previous slides.

More complex task

- We just made a really simple task with the scheduler capable of executing only that one task.
- This task only prints something based on the global variable.
- Usage of the global variable is not optimal - what if we would want to have multiple tasks with this job function?
  - That can be necessary for a lot of non-trivial tasks!
- We will fix that in the following modification!

Step 4: Data structure

The idea is to use different data for tasks with the same job function. For each task, remember a pointer for data and pass it to the function each time it is called. Given that we are working with C, we have to use a generic pointer to datasets - void*.

```
struct led_task_data {
    uint8_t i;
};
struct led_task_data LED_DATA = {.i = 0};
struct task LED_TASK = {.period = 250,
                        .max_execution_time = 20,
                        .relative_deadline = 50,
                        .job = &led_job2,
                        .data = (void *)&LED_DATA};
```

Step 5: Modify function

Now, we can use that data structure in the function itself:

```
void led_job(void *void_data) {
    struct led_task_data *data = void_data;
    printf("Status of green led: %i", data->i);
    data->i = (data->i + 1) % 2;
}
```

Assignment

- Implement the fourth and fifth steps.
- Make a new instance of led task, with different data instances and same function.

API Summary

Summary of the API:

- The basic unit is *struct task*, contains:
  - timing constraints - period, relative deadline and execution time
  - job function and job data
- API uses void pointer to pass data - you have to convert the pointer types manually

Project

- Now implement the schedulers for tasks defined in tasks.h, remember to add your own task (which can do whatever you want it to do).
- There will be examination dates in the IS for project submission - you have to sign up.
  - Once the reservation for the exam date ends (and you can no longer cancel it), I will open the homework vault.
  - The homework vault will be open until the examination date - you have to submit your project in that time period
  - In case you fail to submit the project, you lose one of the attempts for submission.

Communication

Given current conditions, I will establish multiple ways for you to communicate with me (Jan Koniarik):

email 433337@mail.muni.cz

discussion group in IS
`https://is.muni.cz/auth/go/fiyaqc`

Non-official way of communication: (That implies that you have no official guarantees about anything and that everything that I say on discord can be changed later)

discord `https://discord.gg/H6TsGJc` - join IA158 room, account Veverak with verified authority status

MUNI
FACULTY
OF INFORMATICS