MUNI
FI

IA158 - Scheduler

Jan Koniarik

March 12, 2021

Agenda

Project information

Scheduler requirements

API introduction

Example task

Project

- Your goal is to write a scheduler in C.
- It has to schedule our three sets of tasks:

  Custom A set of three predefined tasks, you have to add one task of your own. [1]

  Generated One is randomly generated for each of you.

  Sporadic Third is made of two normal tasks and one sporadic task.
- Your scheduler has to meet all the requirements, and you have three attempts at submission.
- You have to work alone.

[1] Please be creative

Scheduler requirements

Assumptions:

- Jobs are non-preemptible
- Tasks are periodic, and one is sporadic
- One processor
- No resources/priorities/precedence
- Synchronized

Requirements:

- Schedule our sets of tasks
- The schedule has to be valid
- Overrun detection (not prevention)
  - It should be clear to potential users that overrun happened but do not stop the execution.
- The schedule is not hardcoded
- You have to use our clock API to get actual time.

Skeleton

There will be a simple C skeleton in the school information system with CMake as a build system.
Each set of tasks is schedulable without a problem with algorithms based on the lecture. (But beware of your custom task!) Your solution has to schedule all three sets of tasks correctly, in case there is 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 is solvable with an algorithm that expects preemptable jobs. As for your task, it is your burden to design it correctly. [2]

[2] But if you know how to implement task switching, you can do it.

Implementation

- We will check your implementation of the scheduler - write it in a way that we can understand it.
  - Code quality (readability) is a necessity for a good scheduler, as the code has to be easy to understand and debug.
- You can not use dynamic memory.
  - During a normal semester, the task is to solve this on an embedded device.
  - On embedded, we want to avoid dynamic memory when it is not necessary.
  - You either know the number of tasks/jobs, or you have upper limits - there is no need for dynamic memory.

API introduction

Interface

- File *task.h* contains a definition of the structure holding the tasks, job pointers, and sporadic tasks.
- _*tasks.c/h* files provide a description of each set of tasks.
  - gen_tasks.c is available to you at `https://www.fi.muni.cz/~xkoniar/ia158/<uco>/gen_tasks.c`
- *clock.h/c* contains API to work with time.
- *scheduler.h/c* contains API and structure for the scheduler that you have to implement.

Tasks

We will explain how this works on the first task set and the sporadic task. We expect that you can handle the rest by yourself.
The three provided tasks have these properties. All values are in milliseconds: [3]

led period: 250, deadline: 50, max. exec. time: 1

uart period: 251, deadline: 251, max. exec. time: 40

fib period: 1499, deadline: 249, max. exec. time: 40

The led task will be implemented as an example in this presentation.

[3] 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 *main.c* file

Step 1: Write job function

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

```
#include "task.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 instantiation 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. [4]

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

[4] All time units are in milliseconds

Step 3: Write simple scheduler

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

```
void schedule_single_task(struct task *task_ptr) {
    while (true) {
        uint32_t end_time = clock_time() + task_ptr->period;
        task_ptr->job(task_ptr->data);
        clock_delay_ms(end_time - clock_time());
    }
}
```

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 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.

Scheduler interface

The scheduler itself will have to use our API and data structure. See file *scheduler.h*. We use one global instance of struct *scheduler* to represent the data of the scheduler. The definition is empty, but you should use it wisely for the scheduler. Sporadic tasks call the function ‘scheduler_on_sporadic‘ to add a sporadic event into the scheduler structure. See sporadic tasks for details.
The function ‘scheduler‘ is the core function of should do the scheduling. We expect that the function will never return. See ‘main‘ to understand how it is used.

Sporadic task

The third set of tasks contains a sporadic event. This occurs from within the other tasks at random moments. Remember to correctly deny the event in case you are not able to schedule the task.
Read the source code carefully. It gives hints that should help you find a proper solution for sporadic scheduling. It should be simple.

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
- Key function is ‘schedule‘ that executes the scheduling

Project

- Implement the scheduler for the task sets.
- 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

Prefered communication channels:

email 433337@mail.muni.cz

discussion group in IS

`https://is.muni.cz/auth/discussion/predmetove/fi/jaro2021/IA158/`

Or contact me in any other way you see fit. I will try to answer all of your questions, but I do not guarantee a fast response, and it may take me some time. Please be patient.

Experiment

As a little social experiment, please do send me an email once you finish *processing* this document. I promise that I won't judge you for when that happens.