Periodic Task Executor

2016/03/04	0.1	Initial Version	Muhammad Z
2016/06/11	0.99	Final Draft	Muhammad Z
2018/03/19	1.0	First release	Alex K.

Summary

The purpose of this project is to create a facility that can execute periodic recurrent tasks. This facility behaves like a single threaded scheduler for running repeating tasks.
You will gain experience in:

1. Module and API Design
2. Advanced C programming
3. Putting Data Structures/Algorithm to actual usage.

Work in Phases

You have 2 days for this project. Work should be in phases according to the following:

1. High Level Design & Planning Phase (60 min)
   This is a group work, each group will consist of 3-4 students.
   a. Read and understand requirements
   b. Create a Block diagram design of the modules you need to build. This should be drawn on paper/whiteboard
   c. Specify the public API for each module: what function should be there?
2. Detailed Level Design Phase (90 min)
   Individual work,
   a. Detailed module diagram in PDF format
   b. Fully commented C header files for each module.
3. Implementation
   a. Implement the Periodic Executor in C
   b. Provide unit tests

Deliverables

Each phase has its own deliverables:

- High Level Design
    - Block diagram on Paper
- Detailed Level Design
    - Detailed Design in PDF
    - C language header file for each module
- Implementation
  Directory containing:
    - Public header in *inc* subdirectory
    - Source code of executor in *src* subdirectory
    - Test code in *test* subdirectory
    - Makefile with these targets
        - all : compile sources and create a lib, compile and link test program
        - test : run test
        - clean
    - File with detailed design (pdf/png)

High Level Description

The Periodic executor allow the user to enqueue tasks to be executed. These tasks will be executed periodically (see below).
The user should be able to :

- Add a task to the executor specifying its period
- Run the executor - this call will start the periodic execution of the tasks.
  When *Run* is called, the task will be executed with initial delay equal to specified period.
- Each task executed will return a value indicating if it wants to be scheduled for another period or not.
- Pause the execution
  Currently running task if any will complete. No more tasks will be executed. Later, when *Run* is called it will start execution cycle as if it was called for the first time.

The executor should sort the tasks according to their execution due time and attempt to execute each task no earlier than its due time and try its best to execute it no later than its due time.

Example

Assume these tasks are added:

1. Task A, execute every 10 seconds for a total of 3 times
2. Task B execute every 13 seconds
3. Task C execute every 5 seconds for total of 4 times
4. Task D execute every 8 seconds for total of two times
5. Each task executes roughly 0.1 seconds

When the executor run method is called: (Time measured from this point)

- Time 0: nothing happen
- Time 5: execute task C, it returns 1, so it's rescheduled,
- Time 8: execute task D, it returns 1, so it's rescheduled,
- Time 10: execute task A it returns 1, so it's rescheduled
- Time 10.2 : execute task C - should have been at 10 but done after A was executed
- order is implementation defined
- Time 13: execute task B
- Time 15.2: execute task C (5 sec from last finish time)
- Time 16: execute task D, it returns 0, so it's removed from executor
- Time 20: execute task A
- Time 20.5: execute task C
  we should have executed at 20.2 but actually last time it finished at 15.5
- Time 26: execute task B
- Time 30: execute task A
- Time 39.5: execute task B - because of a drift in time

Detailed Requirements

R1. API

The executor should at least provide methods for:

- Create/Destroy of executor.
- Add new task.
- Start execution.
- Pause execution.

R2. Time calculation

Time calculation should be based on *timespec* data structure and using the platform sleep functions.

R3. Logging

The executor will use the logger previously developed. It assumes the main process already called zlog_init. The executor will use the logger entry having a unique name configurable by the user of the scheduler on creation.

R4. Efficiency

Use efficient data structures and algorithm to determine next task to run. Make the overhead of the executor as low as possible.

R5. Error Handling

Handle errors vigilantly on all system calls.

Design Tools

The design document should be presented as a pdf with block diagram of modules/components delivered as PDF file.
Use any of the following free tools to document the high level design of the project:

- https://docs.google.com/drawings
- https://www.gliffy.com/
- https://www.draw.io/