Periodic Task Executor

2016/03/04	0.1	Initial Version	Muhammad Z
2016/06/11	0.99	Final Draft	Muhammad Z

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 to actual usage.

Deliverables

The complete project delivered for review must provide:

1. Phase A
    a. High Level Design
        i. What sub-modules are there?
        ii. What is the public interface of the executor?
    b. Internal Data Structures to be used
2. Phase B
    a. High Level Design Document (PDF file)
    b. Control Flow in pseudo code
    c. Module header file
3. Phase C
    a. C Header File fully documented
    b. C source code files
    c. Makefile that creates a lib & test

d. A Test program with more than 5 tasks

High Level Description

The Periodic executor will execute tasks added to it. Each task can be executed more than one time. The period of the recurrence is set per task and each task can specify if it needs more execution cycles or it has finished.

The executor will arrange for the tasks to be executed according to following algorithm:
Whenever the executor run function is called, each task will be assigned a number representing the next time it will be executed. Basically this is the time now plus the specified interval.
The executor will chose the task that it's time to execute is the nearest and call it's function when the time has come.
Once the task function has returned, then if it needs another run, the time for the next time is calculated based on finish time + interval and it is inserted for another cycle. otherwise, the task is removed.

Definitions

1. Task
   A task is a representation of a function call. It consists of a function pointer and a context Argument which will serve a parameter when the function pointer is actually called.
   The user provided function return value is an integer specifying:
   - 0 : The function should be called again after the interval has passed.
   - Any other value : The function should be removed from the executor.
2. Periodic Execution
   Each task will be added to the executor will specify a time interval to be used for recurring execution. It will be specified in milliseconds.
3. Context
   Each task is added with a context, this is a generic pointer to parameter data that will be passed to the user function each time the task is executed.
4. Order of Execution
   Task is selected for execution based on the fact that it's execution is due. When the task is Completes one run and needs to be rescheduled then the time for next execution is calculated based on time of finish + period.

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

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

- Time 0: nothing happen
- Time 5: execute task C
- Time 10: execute task A
- 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 ms from last finish time)
- 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. Time calculation

Time calculation should be based on timespec data structure and using these functions:

- usleep
- clock_gettime

R2. Logging

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

R3. Efficiency

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

R4. Error Handling

Handle errors vigilantly on all system calls.

High Level 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:

- Visual Paradigm Community Edition for linux - https://www.visual-paradigm.com
- Umbrello UML Modeler - https://umbrello.kde.org/