REAL-TIME CLOCK PERIODIC TASKS CONCURRENT EXECUTOR

KIN SENG, L.

Department of Electrical Engineering, National University of Singapore, Lower Kent Ridge Rd, Singapore.
e-mail: lksark[at]hotmail.com

(Received 31st August 2023; accepted 22nd November 2023)

Abstract. Real-time clock periodic tasks concurrent executor program is being written to execute multiple timed periodic tasks concurrently using modern CPU multithreading and multicores feature at specific RTC periodically. By using RTC as an absolute reference, we can analyze the status of multiple isolated devices on selected time instances to know about overall system status or each individual systems status. Multithreading program able to execute multiple tasks concurrently, making RTC periodic tasks possible. While single threaded sequential program only run multiple tasks in sequence, one at a time. Each task runtime creates delays resulting subsequent periodic tasks no longer synchronous. Nevertheless, this program can run without RTC. It becomes an isolated system, capable of performing its own synchronous operations.
Keywords: *multithreading, real-time clock, concurrency, periodic tasks, sampling rate, synchronous operations*

Introduction

Periodic tasks are tasks which are executed at specified time intervals, again and again, with minimal human intervention; RTC periodic tasks are tasks which are executed repeatedly & automatically if timed at specified real-time clock intervals. Periodic tasks can be tasks of reading various types of sensory data, on/off relays, or executing software operations such as transmits data. Modern electronic systems generally have multiple hardware digital & analogue input & output; modern computers are capable of executing multiple software programs currently. When a system's multiple hardware sensors or multiple isolated systems are being sampled at the same time instance; systems hardware relays were triggered at the same time instance; software operations being executed at same time instance, these sensory data & operation outcomes often inherited strong correlations. Even if they are not executed at the precise real-time clock, we may able to conclude overall status of a system or identify any abnormalities among multiple isolated systems at that time instance. In contrast, when multiple data are sampled at different time instance; software or hardware operations are executed at different time instance. These data & operation outcomes typically have weak correlations and we may not able to derive substantial conclusion about the systems. Concurrent executor capable of executes multiple timed periodic tasks concurrently, hence making it useful in synchronous systems.

Materials and Methods

Download the source code from my web URL homepage, compile it using Microsoft Visual Studio and execute to reproduce the reported outcome. The source codes are written in Microsoft C# (Price, 2020). After compilations, execute the application. Microsoft operating system 'Command Prompt' window appear. We can see that task

‘A’ is executed every RTC second; task ‘B’ is executed every 2 seconds; task ‘C’, ‘D’, ‘E’ are being executed every 3, 4, 5 seconds respectively. The homepage for My online source code URL are:

(1) https://www.algoonline.net/MultithreadingWorkloadsDistribution/Multi-threading_workloads_distribution.htm
(2) https://github.com/lksark/Multithreaded-TCP-server-with-multiple-database-connections-pool
(3) https://github.com/lksark/-Multithreaded-Chat-Server-communicates-with-multiple-clients

In this program, periodic tasks are the functions to be executed by the computer when timed (*Figure 1*). In standard software procedure, whenever software threads are being created, they should be ended using ‘Thread.Join()’ after finished execution (Tanwar, 2019; Halbwachs, 2013; Deitel et al., 2002; Lisper, 1989). However, Periodic tasks are tasks which are executed at specified time intervals, again and again. Thus, the software worker threads are being placed inside while loop, they will not be ended until we manually set ‘RunProgram’ variable to false. Moreover, these software threads are not bind to specific tasks. Every second whenever there are any timed period tasks, executor program will assign equivalent count of software threads to execute them. ‘Thread.Wait()’ are used to allow software worker threads to idle when no tasks to execute (Tanwar, 2019; Halbwachs, 2013; Deitel et al., 2002; Lisper, 1989). Hence, we can re-use the software worker threads without need to repeatedly create & close them. ‘Thread.Wait()’ is one of the software thread states, other states are ‘Thread.Start()’, ‘Thread.Run()’, and etc. (Tanwar, 2019; Halbwachs, 2013; Deitel et al., 2002; Lisper, 1989). In this example program, every second software object ‘Simulated_RTC_tick’ will execute software object ‘SortedList_PeriodicTasks.notify_timed()’. The object will check if latest periodic tasks are timed. Assuming software object ‘Simulated_RTC_tick’ is an accurate RTC that tick every 1 second. Periodic tasks’ function is saved into ‘class_record_PeriodicTask’ array. To get to know which periodic tasks are timed, we use ascending order ‘SortedList’ class. ‘SortedList’ class represents a collection of key/value pairs that are sorted by the keys and are accessible by key and by index. Keys are all periodic tasks next targeted RTC to be executed; values are the ‘class_record_PeriodicTask’ array index.

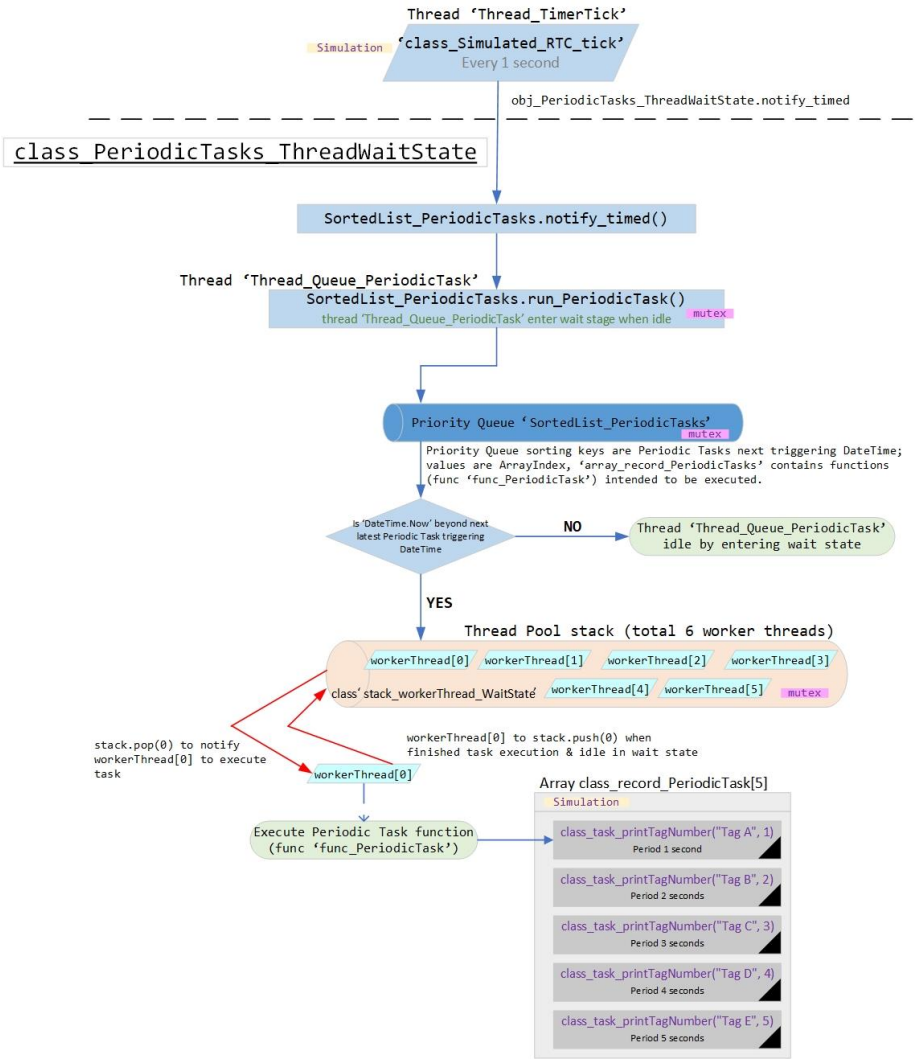

Figure 1. *Process flow diagram.*

Just a reminder, if declared periodic tasks' function as 'SortedList' value, abnormal execution would occur. Whenever periodic tasks are timed, 'SortedList_PeriodicTasks' software object can wake-up software worker threads from idle to execute timed periodic tasks. Software worker threads are reused without need of create new software threads & exit after finished execution. We don't have to declare a dedicated worker threads for every periodic task that would be too overwhelming. Declare sufficient count of software worker threads to ensure that all the timed periodic tasks can be executed concurrently. Vice versa, if the count of periodic tasks is small, we can declare a dedicated worker threads for every periodic task. Worker threads only execute tasks' function. Users should write the function to save the result data separately.

Test methodology (in progress)

Use hardware and software to verify the timed periodic tasks are executed concurrently. Samples high frequency sine-wave signal to validate controller's hardware & software are capable of running concurrently instead of sequentially. Get a controller with 2 analogue voltage inputs. Connect both analogue voltage inputs to the same sine-wave generator output. Run sine-wave generator frequency around analogue

input maximum sampling rate. Use controller's two analogue voltage input to sample sine-wave voltage values at maximum sampling rate. Both data collected at the same time instance should be approximately same.

Results and Discussion

Periodic tasks have many practical applications. Controller's hardware input sensors have various maximum allowable sampling rates; controller's hardware output devices have various triggering times. Often, we don't always sample data at maximum allowable sampling rate; nor we always continuously activate output devices. Hence, we can use periodic tasks to set user-defined sampling rate or periodically activate hardware output devices. By using RTC as an absolute reference, multiple isolated systems are able to execute hardware or software operations at the same RTC instance. Moreover, uses can evaluate single/multiple systems status by checking data collected, software or hardware operations outcome at selected time instance. Multithreading program able to execute multiple tasks concurrently, making synchronous operations possible. While single threated sequential program only run multiple tasks in sequence, one at a time. Each task runtime creates delays resulting subsequent periodic tasks no longer synchronous. When first task and last task finished execution time far apart, consequently these 2 periodic tasks' result possess weak correlation-ship or do not perform synchronised actions as planned. Hence, RTC periodic tasks concurrently executor program is handy in time sensitive systems. The program allows systems' multiple output devices or software to perform synchronous operations; the software enable systems to sample multiple data at the same time instance to derive meaningful conclusions.

Needless to say, the CPU running the program needs to have multithreading feature & sufficient count of logical processors to run multiple tasks concurrently. Moreover, controllers should able to execute all its inputs & outputs concurrently. Multiple isolated devices with antenna receiving GPS can perform periodic tasks at specific real-time clock (RTC) at the same time. Regardless, whether they are online or offline, communicating with each other or isolated. Hardware devices such as Digital/Analog inputs and relay outputs (e.g., mechanical contactors) can have wildly different runtimes. Some mechanical contactors can have extensive lengthy triggering time. Overall, they all are generally having much longer runtimes than CPU operations. We should aware of each system devices runtimes during consideration stage. RTC periodic tasks concurrent executor program is written to execute multiple timed periodic tasks concurrently using modern CPU multithreading feature. Multithreaded program able to execute multiple tasks concurrently, making RTC periodic tasks possible, while single sequential program only able to run multiple tasks sequentially, one at a time, these runtime delays make multiple periodic tasks not executed synchronously (Tanwar, 2019; Halbwachs, 2013; Deitel et al., 2002; Lisper, 1989). Nevertheless, the program can run without RTC. It becomes an isolated system, capable of performing its own synchronous operations.

Conclusion

Electronic systems are getting more & more complicated, they are having more & more hardware inputs & outputs, sensors & relays. Computers are having more & more

software installed. Inevitably we would want these periodic tasks to be execute concurrently. Modern multithreaded CPU and this program empower systems to perform more accurate synchronous actions. Human walking is an example of synchronous actions. We hear & see our surrounding environment & react during our walk; we walk by synchronously moving our legs & swinging our limbs. In addition, when we make systems' clock adhere to Real-Time Clock (RTC), we can plausibly make multiple isolated systems to perform synchronous operation on the selected time instances. Data analysis become meaningful when multiple isolated systems' data are sampled at the same RTC.

Acknowledgement

This research is self-funded.

Conflict of interest

The authors declare that there is no conflict of interest involve in this research study.

REFERENCES

[1] Deitel, H.M., Deitel, P.J., Nieto, T.R. (2002): Visual Basic. NET: how to program. – Prentice Hall 720p.
[2] Halbwachs, N. (2013): Synchronous programming of reactive systems. – Springer Science & Business Media 215: 140p.
[3] Lisper, B. (Ed.) (1989): Synthesizing synchronous systems by static scheduling in space-time. – Berlin, Heidelberg: Springer Berlin Heidelberg 111p.
[4] Price, M.J. (2020): C# 9 and. NET 5–Modern Cross-Platform Development: Build intelligent apps, websites, and services with Blazor, ASP. NET Core, and Entity Framework Core using Visual Studio Code. – Packt Publishing Ltd. 249p.
[5] Tanwar, S. (2019): Hands-On Parallel Programming with C# 8 and. NET Core 3: Build solid enterprise software using task parallelism and multithreading. – Packt Publishing Ltd. 35p.