**UDC 004.051**

**S.O. Diakov, A .S. Samoidiuk**

# ASYNCHRONOUS PROGRAMMING AS A MEANS TO INCREASE THE PRODUCTIVITY OF SMART HOME MANAGEMENT SYSTEMS

*Abstract*: This article considers the main problem of modern high-load systems, namely, the productivity of their work. The main purpose of this article is to find an approach to address the issue of high-load systems performance, particularly smart house management systems. These systems have become very popular in our time, the number of their users is growing every year, so the question of their performance is very relevant. In order to achieve this goal, I will consider existing problems and different ways of their solutions. The approach of asynchronous programming is proposed as one of the ways to solve a problem of productivity. It will greatly increase the bandwidth of servers and reduce the response time to customer requests - two common problems that arise when designing high-load systems. The relevance of the proposed solution will be tested experimentally, with the results presented in tabular form.

*Keywords*: Software architecture; C#; asynchrony; async; await; CRUD; design patterns; data modeling.

## Problem Statement

The smart house management system can be divided into two levels: the upper and the lower. The lower level is a collection of smart devices and the upper one is a system that processes the data it receives from these devices. This article focuses on the upper level of the smart home management system and the problems that arise in it during the processing of information.

As mentioned above, the upper level is responsible for receiving and analyzing data from sensors, sending data to the cloud storage, saving the results in its own database. And because of such a large list of tasks, there are problems with performance[5].


```
{
    "devices": {
        "thermostats": {
            "peyiJNo0IldT2YlIVtYaGQ": {
                "device_id" :"peyiJNo0IldT2YlIVtYaGQ",
                "structure_id" :"VqFabWH21nwVyd4RWgIgNb292wa7hG_dUwo2i2SG7j3-BOLY0BA4sw",
                "name" :"Hallway (upstairs)",
                "name_long" :"Hallway Thermostat (upstairs)",
                "last_connection" :"2016-10-31T23:59:59.000Z",
                "temperature_scale" :"C",
                "target_temperature_f" :72,
                "target_temperature_c" :21.5,
                "target_temperature_high_f" :80,
                "eco_temperature_high_f" :80,
                "eco_temperature_high_c" :24.5,
                "away_temperature_low_f" :65,
                "away_temperature_low_c" :19.5,
                "hvac_mode" :"heat",
                "previous_hvac_mode" :"heat",
                "ambient_temperature_f" :72,
                "humidity" :40,
                "hvac_state" :"heating",
                "where_id" :"UNCBGUnN24...",
                "is_locked" :true,
                "locked_temp_max_f" :80,
                "locked_temp_min_c" :19.5,
                "label" :"Pat's room",
                "where_name" :"Hallway",
                "sunlight_correction_enabled" :true,
            }
        }
    }
}
```


*Figure 1.* An example of information with which the top-level system works

The information processing system every second receives a large amount of information, like that shown in the figure. So, what can go wrong? In some cases, this greatly affects the speed of the application. For example, the user sends a request to a server which accesses the external API and the database and makes a response from the results obtained. It often happens that the API returns the information with delay. And while the server does not receive that info, it cannot perform other operations, such as querying the database, which also takes a lot of time depending on its size.

**An overview of existing solutions**

There are two programming models: asynchronous programming model and synchronous [6]. Each of them can work in a multi-threaded or single-threaded environment. Let's look at each of them and start with the synchronous model. In this programming model, a task list is assigned to a thread and it works on them one by one. When it finishes work on the first task, it transitions to the second. There is no way to finish work on the mid and start executing on other tasks.

```
[HttpGet]
public IActionResult GetData (string param1, string param2)
{
    // make call to external API
    var apiRequestRes = GetDataFromAPI(param1);

    // make request to database
    // will not start until the request to api is executed
    var dbRequestRes = GetDataFromDatabase(param2);

    return Ok (new List<string> {apiRequestRes, dbRequestRes});
}
```

*Figure 2.* An example of synchronous code

The figure shows the function on the server that processes requests. As we can see, it makes a request to the external API and database. And combines these two results for a response to the client. The problem here is that the request to the API can take from a few milliseconds to several seconds, and the query to the database will not even be sent. This is a fairly common problem in smart home management systems.

Another approach is multithreading. Here we have several threads, each of them working on their own task list. The number of available threads may vary depending on the amount of available resources [7]. In the figure below you can see the features of multi-threading. We have four threads and each of them has its own list of tasks and work independent from each other.

In the first case, the problem is obvious. Such an execution case takes too much time and does not allow the server to work on other tasks while waiting for a response from some resources. The second case is much more interesting. If there are several threads running in the process, then we can expect two main problems: deadlocks and race conditions. Deadlocks arise when two or more threads are waiting for the same resource and block each

other, since it is not specified which thread should receive the necessary resource in such a situation. Race conditions occur when two or more threads use the same resource and change it in an unintended manner, since the operating system may change the order of the threads. There are ways to resolve these and some other situations, but this is much complicated and not as efficient as the proposed solution.

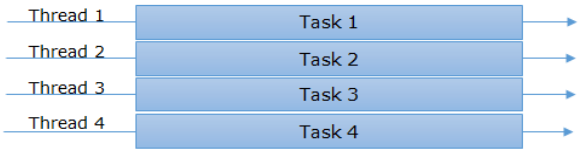


*Figure 3.* Task execution in multi-threaded synchronous programming model

**Proposed solution**

Asynchronous is a form of processing that allows you to continue to perform other tasks that do not wait for the transfer to complete. Asynchronous programming successfully solves many problems. One of the most important is the availability of the user interface. For example, a smart home control system, after the user has selected an action, the program sends a request to the server. And that process that can last quite a long time. If the application is working synchronously, the user will not be able to interact with the page until the result comes [8]. Asynchronous code allows you to hide these unpleasant effects from the user and keep the page live. In this case, the main execution thread is split into two branches. One of them continues to deal with the interface, while the other executes the request.

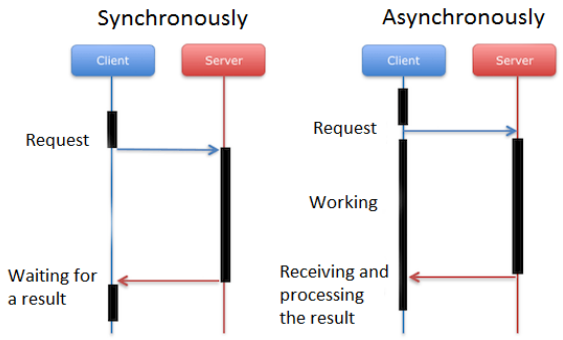


*Figure 4.* Comparison of synchronous and asynchronous models

Let's see how to write asynchronous code using the example of the C# language. First need to understand how the server handles requests. When the server receives a request, it assigns it one thread from the thread pool. Since the number of available threads is limited then if the thread wastes most of the time in waiting then it will greatly decrease system performance. But this does not work when writing asynchronous code. Since instead of

waiting for the completion of any request, it will be available to the thread pool. It will make the server faster and its throughput will increase significantly.

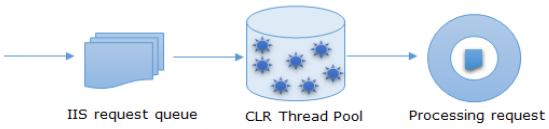


*Figure 5.* The process of processing the request by the server

How to write asynchronous code? The basis for working with asynchronous calls in C # are two keywords: async and await, the purpose of which is to simplify the writing of asynchronous code. They are used together to create an asynchronous method. The asynchronous method has the following features [9]:

- the method header uses the «async» modifier;
- the method contains one or more await expressions;
- the return type is one of the following: void, Task, Task<T>, ValueTask<T>.

The asynchronous method, like the usual one, can use any number of parameters or not use them at all. It is also worth noting that the word async, which is specified in the method definition, does not automatically make the method asynchronous. It only indicates that this method may contain one or more await expressions. Let's rewrite the method presented earlier on asynchronous.

```
[HttpGet]
public async Task<IActionResult> GetData(string param1, string
                                                        param2)
{
    // run task that make call to external API
    Task<string> taskApiCAll = GetDataFromApi(param1);
    // immediately make call to database
    Task<object> taskDbRequest = GetDataFromDatabase(param2);
    //wait for results
    await Task.WhenAll(taskApiCAll, taskDbRequest);

    return Ok (new List<string> {taskApiCAll.Result,
                                 (string) taskDbRequest.Result});
}
```

*Figure 6.* An example of asynchronous code

The main benefits of asynchronous programming are [10]:

- significantly increase the performance of your application, when you have long-running operations;
- organize your code with async / await, make it more readable;
- using the latest upgrades of the language features.

So, how this approach helps to improve smart home management systems performance? These systems operate with large amounts of data and work with API. It was noticed while the request is being executed, about 70-80% the time wasting while waiting for the dependent tasks. It can be maximally used in asynchronous programming, where, as soon

 ISSN 1560-8956

as a task is transferred to another thread (for example, SQL), the current thread saves state and is available to run another process, and when the SQL task is completed, any thread that is free, can do this task.

## Results

I have run several experiments to make sure that the proposed solution is effective. Results you can see on the figure below.

| Simultaneous queries | Synchronously | Asynchronously |
|---|---|---|
| 10 | 790 ms | 730 ms |
| 20 | 1220 ms | 1130 ms |
| 50 | 2300 ms | 1913 ms |
| 100 | 3829 ms | 3210 ms |
| 200 | 9434 ms | 5730 ms |

*Figure 7.* Comparison of synchronous and asynchronous methods

As you can see from the figure, asynchronous has a great advantage with many simultaneous queries.

## Conclusion

Older programming models such as the synchronous model cannot provide enough performance for modern systems. Therefore, a new asynchronous model comes to replace it. This model is very well suited for systems that work very closely, for example, with a database. Asynchronous code allows the server to do other tasks while it is waiting for the current task to respond. Therefore, this approach has many advantages, including an increase in server bandwidth.

This approach to programming was used to create a smart home management system and showed itself well. It was noticed a significant increase in the speed of the server response under heavy loads. Based on these results, I can say that the goal has been achieved.

## REFERENCES

1. Harper R. Inside the Smart Home / R. Harper. — New York City: Springer, 2003. — 280 с

2. Badica C. An overview of smart home environments: Architectures, technologies and applications / C. Badica, M. Brezovan // ResearchGate. — 2013.

3. Saponara S. Network Architecture, Security Issues, and Hardware Implementation of a Home Area Network for Smart Grid / S. Saponara, T. Bacchillone // Journal of Computer Networks and Communications. — 2012.

4. Coulouris G. Distributed Systems: Concepts and Design / G. Coulouris, J. Dollimore, T. Kindberg. — London: Pearson, 2011. — 1080 с

5. Wu C. Service-Oriented Smart-Home Architecture Based on OSGi and Mobile-Agent Technology / C. Wu, L. Fu // IEEE Transactions on Systems, Man, and Cybernetics, Part C. — 2007. — С.193-205.

6. Herlihy M. The Art of Multiprocessor Programming, Revised Reprint / M. Herlihy, N. Shavit. — Burlington: Morgan Kaufmann, 2012. — 536 с

7. Crenshaw D. The Myth of Multitasking: How "Doing It All" Gets Nothing Done / D. Crenshaw. — San Francisco: Jossey-Bass, 2008. — 144 с

8. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. — Boston: Addison-Wesley Longman Publishing Co., Inc., 2002. — 576 с

9. Blewett R. Pro Asynchronous Programming with .NET / R. Blewett. — New York: Apress, 2013. — 352 с

10. Cleary S. Concurrency in C# Cookbook: Asynchronous, Parallel, and Multithreaded Programming / S. Cleary. — Sebastopol: O'Reilly Media, 2014. — 208 с