Vol. 34, No. 01, pp. 193-204/Marzo 2021

ISSN-E 1995-9516
Universidad Nacional de Ingeniería
COPYRIGHT © (UNI). TODOS LOS DERECHOS RESERVADOS
http://revistas.uni.edu.ni/index.php/Nexo
https://doi.org/10.5377/nexo.v34i01.11297

Research of a multithreaded non-deterministic system model

Investigación de un modelo de sistema no determinista multiproceso

Dmitry V. Pashchenko[1*], Dmitry A. Trokoz[1], Alexey I. Martyshkin[2], Tatyana Yu. Pashchenko[3], Mikhail M. Butaev[4], Mikhail Yu. Babich[4]

[1] Penza State Technological University, 440039, Russia, Penza, 1/11 Baydukova proyezd/Gagarina ul., 1/11
[2] Department of Computational Automatons and Systems, Penza State Technological University, 440039, Russia, Penza, 1/11 Baydukova proyezd/Gagarina ul., 1/11
[3] Sub-department Management and Economic Security, Penza State University, 440026, Russia, Penza, Krasnaya Street, 40
[4]JSC Research and Production Enterprise «Rubin», 440000, Russia, Penza, Baydukova St, 2
[*]dmitry.pashcenko@gmail.com

(recibido/received: 25-September-2020; aceptado/accepted: 21-December-2020)

ABSTRACT

Managing the systems which behaviour is non-deterministic is one of the most important problems in modern management theory. Today, systems with structural and behavioural complexity are prevalent in all areas of human activity, and therefore, their research is of the utmost importance. Such systems, as opposed to deterministic systems, are called non-deterministic. They are characterised by difficult predictable behaviour determined both by external random influences, and within the systems themselves. A clear example of a non-deterministic system is crowds of people, factories, and computer networks and systems. The problem of non-deterministic behaviour directly within the context of professional activities can be seen using an example of building syntactic analysers. The aim of the paper is to design a class of systems oriented towards supporting elements of a discrete event model. The target of research is to simulate discrete event models. The subject of research is a creation of a discrete event model based on the behaviour of an undetermined finite state automaton. During the preparation of the paper, there was developed and practically implemented an algorithm for the application, which materializes the principle of working with threads. The results obtained in the paper are aimed at solving the problem of parallel data processing based on the parallelism of NFA's (non-deterministic finite automaton) behaviour when reading the input string characters. As a result, this should have a positive impact on the regulation of the simulation processes of a non-deterministic system, increasing its efficiency and stability. In conclusion, the algorithm of the application work is disclosed and conclusions about the effectiveness and efficiency of its development are drawn.

Keywords: multithreading system, non-deterministic automaton, parallel system, modelling, discrete-event simulation.

RESUMEN

La gestión de sistemas cuyo comportamiento no es determinista es uno de los problemas más importantes de la teoría de la gestión moderna. Hoy en día, los sistemas con complejidad estructural y de comportamiento prevalecen en todas las áreas de la actividad humana y, por lo tanto, su investigación es de suma importancia. Tales sistemas, a diferencia de los sistemas deterministas, se denominan no deterministas. Se caracterizan por un comportamiento difícil de predecir, determinado tanto por influencias aleatorias externas como dentro de los propios sistemas. Un claro ejemplo de un sistema no determinista son las multitudes de personas, fábricas y redes y sistemas informáticos. El problema del comportamiento no determinista directamente en el contexto de las actividades profesionales puede verse utilizando un ejemplo de construcción de analizadores sintácticos. El objetivo del artículo es diseñar una clase de sistemas orientados a elementos de soporte de un modelo de eventos discretos. El objetivo de la investigación es simular modelos de eventos discretos. El tema de investigación es la creación de un modelo de evento discreto basado en el comportamiento de un autómata de estado finito indeterminado. Durante la preparación del trabajo, se desarrolló e implementó prácticamente un algoritmo para la aplicación, que materializa el principio de trabajar con hilos. Los resultados obtenidos en el artículo tienen como objetivo resolver el problema del procesamiento de datos en paralelo basado en el paralelismo del comportamiento de NFA (autómata finito no determinista) al leer los caracteres de la cadena de entrada. Como resultado, esto debería tener un impacto positivo en la regulación de los procesos de simulación de un sistema no determinista, aumentando su eficiencia y estabilidad. En conclusión, se divulga el algoritmo del trabajo de la aplicación y se extraen conclusiones sobre la efectividad y eficiencia de su desarrollo.

Palabras clave: sistema multiproceso, autómata no determinista, sistema paralelo, modelado, simulación de eventos discretos.

1. INTRODUCTION

The problem of non-determinism directly within the professional activity framework can be seen using an example of building syntactic analysers. The first LR analysers could only perform one operation at a time. Naturally, this algorithm took a lot of time to execute the program, and this time rises exponentially depending on the quantity of parameters. In 1984, a GLR-analyser was developed to be able to process several processes in parallel. The author of this development was Masaru Tomita, a Japanese scientist. He proposed a generalised algorithm for the LR parser, which allows us, unlike its predecessor, to work with vague grammars. At the heart of his idea, there is the parallelization of stacks.

The aim of this paper is to design a class-oriented towards supporting elements of a discrete event model. The following tasks have been defined in order to achieve the set goals:

1. Analysis of the subject area.
2. Analysis and search for solutions to the objective.
3. Development of an application that implements work with data threads.
4. Analysis of results.

The target of research is a simulation of discrete event models. The subject of research is the creation of a discrete event model based on the behaviour of a non-determined finite state automaton.

The application algorithm that implements the principle of working with threads has been developed and practically implemented.

This development aims to solve the problem of parallel data processing based on the parallelism of NFA behaviour when reading the input circuit characters. As a result, this should positively impact the regulation of the simulation processes of a non-deterministic system, increasing its efficiency and stability.

Any system can be described according to a number of attributes: interaction with the external environment, system structure, nature of functions, nature of development, degree of organisation, complexity of behaviour, intended purpose and many others.

2. MATERIALS AND METHODS

2.1. Simulation modelling

Despite the variety of existing systems, most of them are complex ones. However, the description of such a system through analytical mathematical models will be rather imprecise (Dukhanov & Medvedeva, 2010). Among the simulation models, two groups of models can be distinguished: discrete-event and discrete-time models. The proper division is based on the control element. In discrete event systems it is an event, and in discrete time systems it is time. Discrete simulation models may depend on events as well as on time.
In terms of the abstraction level used for the representation of the system being simulated, modelling methods are divided by:
- Agent-based modelling;
- Discrete event modelling;
- System dynamics.

Agent-based modelling describes the behaviour of selected participants, who are called agents. People, animals, transport, companies, natural phenomena and others are considered as agents. The state of the system depends on the activities of each of the participants (Dadenkov & Kon, 2015).

System dynamics is a simulation approach that studies the behaviour of complex dynamic systems (Mikhailov, 2015).
Discrete event modelling, along with system dynamics, is considered a traditional method of simulating systems. It was developed back in the 1960s by the scientist Jeffrey Gordon. His approach is to characterise a system as a sequence of individual states that are separated by time intervals. In this way, we do not see the system as a whole (rejecting the continuity of time), but as a string of states that occur depending on a clearly defined signal, i.e. time. System can make a transition from a previous state to the next one only after a certain event has occurred. The system's behaviour pattern can be described in three terms: state, action, and event. After performing some action, the system changes states. The state of the system is only able to change thus when a certain event occurs (Zimina, 2017; Martyshkin et al., 2018).

Graphically, the discrete-event model of a system can be represented as a trajectory with a slider moving along it, which sets in motion after an event that has occurred. The term "process simulation" is also used as a synonym for discrete-event simulation, but they are separated due to differences in their interpretation. Specially developed simulation modelling systems are used to simulate systems using the discrete event modelling method. They are divided into general-purpose systems and field-specific one-dimensional (specialised) systems. Field-specific simulation systems usually work with systems that have a specific specialisation in a particular field, e.g. mechanical engineering, shipbuilding, computer networks, transportation flows and road traffic, etc.) (Pashchenko et al.,2016; Pashchenko et al., 2016; Pashchenko et al., 2015).

General-purpose systems have a basic set of tools for modelling. They are mainly involved in the modelling of mass service systems (MSS) processes (Martyshkin, 2016; Martyshkin, 2016; Martyshkin & Yasarevskaya, 2015). Common systems such as AnyLogic, GPSS (General Purpose Simulation System), Arena are among the general purpose simulation systems. The idea of GPSS belongs to Geoffrey Gordon, the founder of the discrete simulation approach; this system is one of the first of this kind still in use today. Modelling in the AnyLogic simulation environment has significant advantages and ease of use. It stands out the following features among the advantages of this program:

- The speed of development. Models are created with minimal time consumption;
- Saving money;
- User-friendliness of its interface;
- It provides a wide range of tools for working with discrete and continuous models;
- Opportunities to create interactive animation for better perception of models;
- Supports model experiments;
- Allows you to create models of any complexity and with any approach through a constantly updated tool base (Zimina, 2017).

2.2. Components of discrete event modelling

The following components must be defined for DES modelling:
- Essence - some object that is in an active state when an event occurs in the system.
- Activity - processes that take place in the course of modelling the system.
- Events - phenomena that bring the system to a new state at a certain point in time.
- Resources - discrete values or objects of the material world.
- Global variables - variables without limited access.
- Random number generator – a sequence of numbers given at random.
- Statistical collector - an algorithm that tracks data on the state of the system and generates statistics based on it.

2.3. Finite-state automaton

The so-called "five" (Q, Σ, δ, q_0, F) components are used for formal description of the automaton, where the first three components, Q, Σ, F, are sets (Biktashev & Vashkevich, 2013). Q contains many states of the automaton. Σ contains many input characters or the automaton's alphabet. F is a set of assumable states of the automaton. If each of these sets is finite, the name of the automaton is finite (Vashkevich & Biktashev, 2016). The remaining two components are δ and q_0. The character δ is used to denote the function of transitions, and q_0 describes the initial state of the automaton, with $q_0 \in Q$ (Vashkevich & Biktashev, 2011).

2.4. Non-deterministic finite automaton

It is mentioned in foreign sources as a Non-Deterministic Finite Machine (NFM) or Nondeterministic Finite Automaton (NFA) (Vashkevich & Biktashev, 2011; Vashkevich et al., 2015). In many ways, the NFA definition is similar to that of a deterministic finite-state automaton. For comparison, the definitions of the DFA and NFA are comparable in Table 1.

Table 1. Comparison of definitions of DFA and NFA

	DFA	**NFA**
Multiple states	Q	Q
Alphabet (many characters)	Σ	Σ
Initial state	q_0, ($q_0 \in$ Q)	q_0, ($q_0 \in$ Q)
Plenty of assumable states	F, (F ⊆ Q)	F, (F ⊆ Q)
Function of transitions	δ: Q × Σ → Q	$\delta: Q \times \Sigma \rightarrow 2^Q$

The formal definition of DFA (Vashkevich & Vashkevich, 1996; Volchikhin et al., 2013):

Q is the final set of all automaton states.
Σ is a finite set of characters, called the alphabet.
δ is a transition function, where $\delta: Q \times \Sigma \rightarrow Q$.
q_0 is the initial state of the automaton ($q0 \in Q$).
F is a set of finite states, where $F \subseteq Q$.
The formal definition of the NFA:
Q is the finite set of all states of the automaton.
Σ is a finite set of characters called an alphabet.
δ - transition function, where $\delta: Q \times \Sigma \rightarrow 2Q$.
q_0 is the initial state of the automaton ($q_0 \in Q$).
F is a set of finite states, where $F \subseteq Q$.

The table shows that both automata have a similar description, except for the transition function (Dubinin et al., 2016). According to the DFA definition, the transition function of the DFA matches only one state to a pair $\{qi, sj\}$, whereas in an NFA, this condition is violated, and the pair $\{qi, sj\}$ may match a set of states or be empty.

3. RESULTS AND DISCUSSION

A distinctive feature of a NFA is that the transition from one state to another is not clearly defined. This means that if there is a situation of alternative transitions, the automaton will make a simultaneous transition to several states under given conditions. This is the automaton's ability to process in parallel (Dubinin & Drozdov, 2016). Thus, a non-deterministic finite automaton is a generalisation of the deterministic finite automaton, because it can be in several states simultaneously (Staroletov, 2011). Summarising the above about the DFA and the NFA, we will present the final comparative table 2 (Martin Fowler, 2011).

Table 2. Comparison of DFA and NFA

DFA	NFA
If only one transition from one state to the next is possible for each entry character, this automaton is called deterministic.	If several transitions can be made from one state, this automaton is called non-deterministic.
The DFA does not allow for empty transitions (without an input character).	The NFA allows for empty transitions
There is a search with a return	No search with return
Occupies more memory	Occupies less memory
If the DFA finds itself in an accepting state after reading the input string, it accepts the last; otherwise, it does not accept it.	If the NFA finds itself in at least one accepting state from the whole set of states after reading the input string, it accepts this string.

A non-deterministic finite-state automaton is the simplest discrete-event model. To present the NFA as a model, it is necessary to set its initial and final (accepting) state and define the main events and conditions for the transition from one state to another (Pashchenko et al., 2020).

The NFA modelling usually takes place in two ways:
1) A table of transitions.
2) A transition diagram (transition graph) (Vashkevich, 2004; Pashchenko et al., 2020).
For example, let us build NFA models based on these methods.
Let us present some sequence of input tape characters that the NFA must-read. Let us take the character string {a, a, c, b, b, c} as such a sequence, which the NFA defines as the alphabet Σ. Figure 1 shows a

sequence of characters in the form of an input tape (string) (Mozgovoy, 2006).

a	a	c	b	c	b	c

Figure 1. Input tape

Initial NFA status $q_0 = a$, final status $F = c$. Based on this data, we have received the following image of the NFA in the form of a graph (Figure 2).

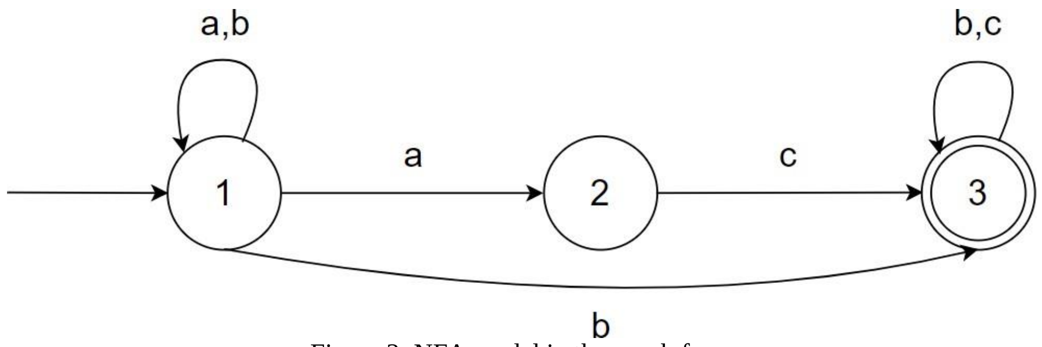


Figure 2. NFA model in the graph form

In accordance with the rules for simulation of the graphs, the automaton states were marked with circles, and the accepting state was marked with a double circle. Each state is numbered inside the circle. This NFA model has three states. The arcs showed as arrows show transitions between states as well as cyclic transitions, i.e., a transition to the same state. There are two cyclic transitions in the figure. The incoming arrow on the left indicates the initial state of the automaton. Each transition arc from the top has a designation for the character, following which the transition from one state to another is made.
The transition table is a method based on the transition function $\delta = \{qi, sj\}$, where qi is a state, sj is an input character. Therefore, each state of this NFA automaton can be described as follows.

$\delta(1, a) = \{1,2\}$ $\delta(1, b) = \{1,3\}$ $\delta(1, c) = \emptyset$

$\delta(2, a) = \emptyset$ $\delta(2, b) = \emptyset$ $\delta(2, c) = \{3\}$

$\delta(3, a) = \emptyset$ $\delta(3, b) = \{3\}$ $\delta(3, c) = \{3\}$

Convert it into a transition table (Table 3).

Table 3. NFA transition table

s_j q_i	**a**	**b**	**c**
1	1,2	1,3	-
2	-	-	3
3	-	3	3

The aim of the paper is to develop an application that, when an alternative transition occurs, would launch a new thread and continue to operate in multiple threads. In programming theory, this principle of working with data is called multithreading (Staroletov,2011; Martyshkin Alexey & Pashchenko Dmitry, 2019). By multithreading, we will mean the execution of several threads in one process. Thus, a process is implemented through the execution of threads (Kizilov et al., 2016; Kizilov et al., 2015). The advantages of multithreading applications include a significant increase in the speed at which the program is executed.

The first programs were single-processor. They were executed on single-core processors, i.e., the programme was executed in a single thread. However, users have a feeling of parallel execution of the program due to the processor's speed. This phenomenon is called the pseudo-parallel execution of threads. For true parallel execution of threads, it is necessary to have a multi-core processor, and then the division into threads takes place between the processor cores. Schematically, the work of such an application is shown in Figure 3.

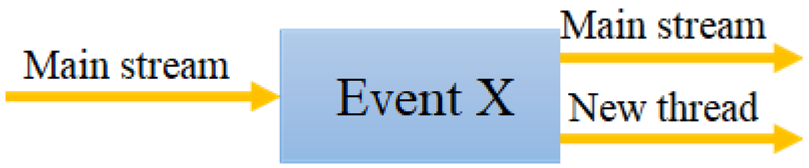
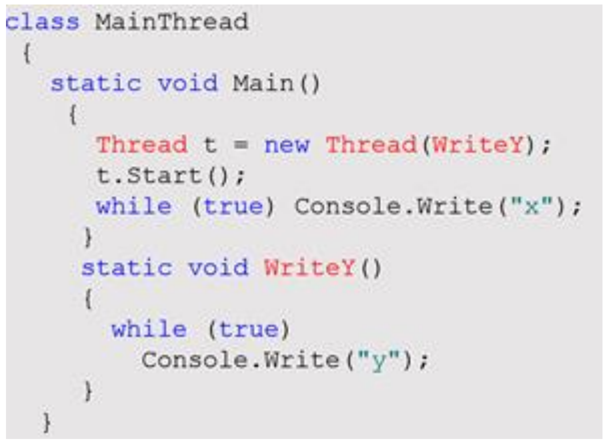


Figure 3. Running a multithreaded application

The process generates the main thread from which new threads are created, thus forming a hierarchy of threads. The MainThread is created within the class. A new thread t is created within this thread, which is additional to MainThread. Thread t executes the command "print y" and the main thread "print x". Both commands are executed simultaneously (Figure 4).



```
class MainThread
 {
   static void Main()
     {
       Thread t = new Thread(WriteY);
       t.Start();
       while (true) Console.Write("x");
     }
     static void WriteY()
     {
       while (true)
         Console.Write("y");
     }
 }
```

Figure 4. Generation of a main thread

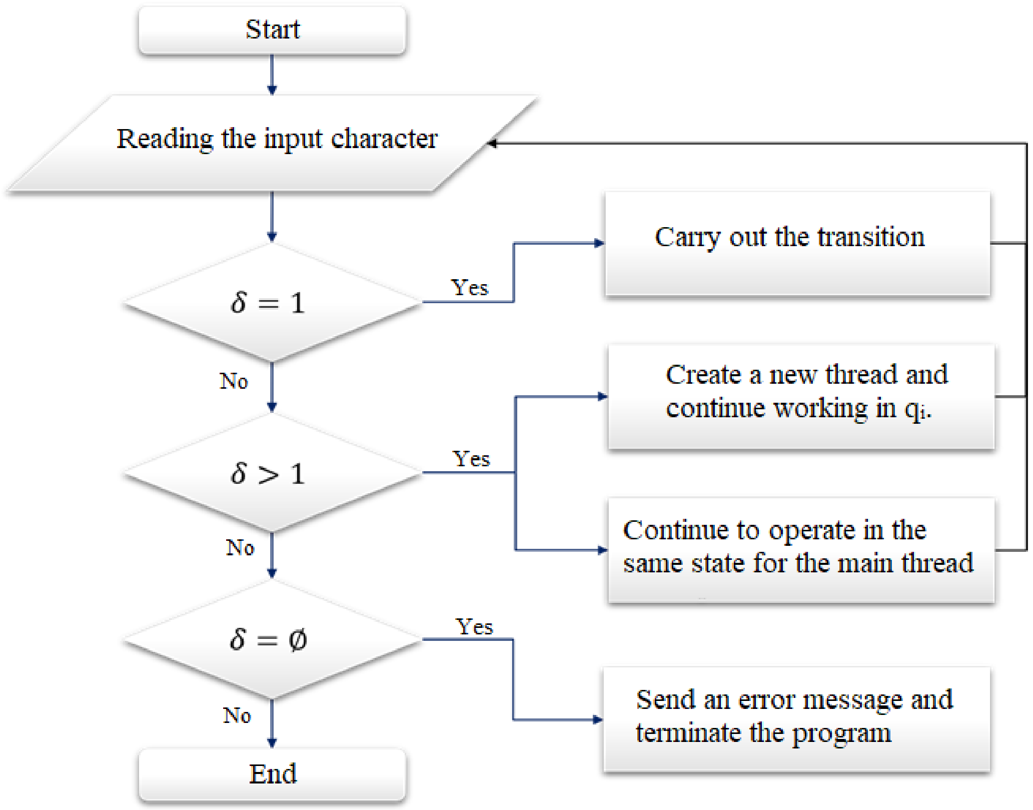
1.5. Development of an algorithm for simulation of a discrete event system and its software implementation.

The stages of development of a multithreaded application in C# are described. At the first stage, we dealt with the theoretical issues of the application operation. The principle of its operation is based on a discrete event model of NFA behaviour. The algorithm is then described using a block diagram describing the main steps of program execution. This is followed by the final result of the application development: its software implementation in the VisualStudio environment.

1.6. Drawing up the algorithm of the program operation

The functioning principle of the algorithm is based on 3 principles:
- Description of NFA behaviour;
- Input tape generation;
- Remembering the starting points for alternatives.
Theoretical development of an algorithm describing the program operation.
The model of operation of a discrete event system (NFA) is taken as a theoretical basis for development. Based on the description of NFA's behaviour, an algorithm for processing input data into output data was

developed (Figure 5).

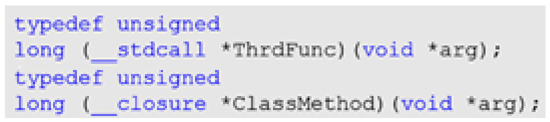


Start
Reading the input character
$\delta = 1$
Yes
Carry out the transition
No
$\delta > 1$
Yes
Create a new thread and continue working in q_i.
Continue to operate in the same state for the main thread
No
$\delta = \emptyset$
Yes
Send an error message and terminate the program
No
End

Figure 5. Algorithm for processing input data into output data

1.7. Software implementation of the algorithm

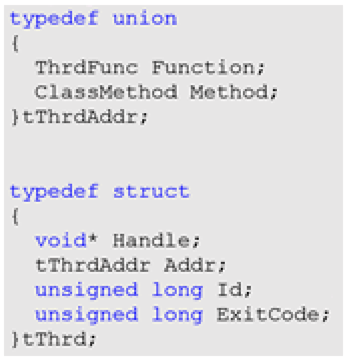
In order to realise this goal, it was necessary to perform calculations in a thread that would be within the class. In the first stage, the class in which the calculations are performed was declared (Figure 6).



```
typedef unsigned
long (__stdcall *ThrdFunc)(void *arg);
typedef unsigned
long (__closure *ClassMethod)(void *arg);
```

Figure 6. Definition of the class in which the calculations are performed

The mainstream was then created and launched via the CreateThread method (Figure 7).

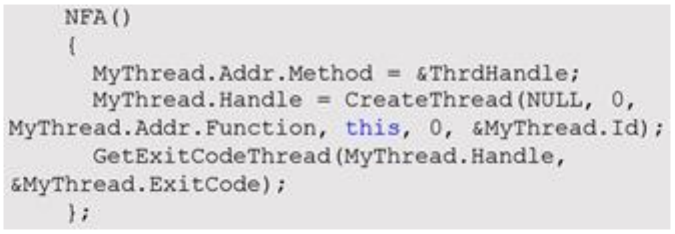


```
typedef union
{
   ThrdFunc Function;
   ClassMethod Method;
}tThrdAddr;


typedef struct
{
   void* Handle;
   tThrdAddr Addr;
   unsigned long Id;
   unsigned long ExitCode;
}tThrd;
```

Figure 7. Creating the mainstream

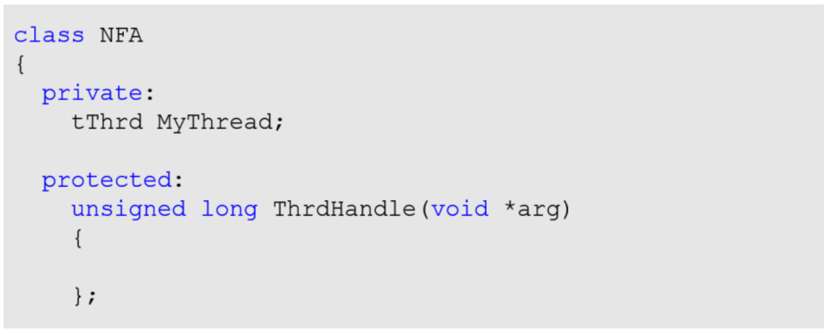
The problem with the transferred parameters using the CreateThread function was solved by bringing the method into static (Figure 8). Now there was no access error to class data, as the function was moved to the class.



```
    NFA()
    {
      MyThread.Addr.Method = &ThrdHandle;
      MyThread.Handle = CreateThread(NULL, 0,
MyThread.Addr.Function, this, 0, &MyThread.Id);
      GetExitCodeThread(MyThread.Handle,
&MyThread.ExitCode);
    };
```

Figure 8. The solution of the problem with transferred parameters using the CreateThread function

Thus, the resulting class looks like a complete system, inside of which there is a thread with data transferred through it. This greatly simplifies the need to search and access the necessary data since all of them are inside the class. Another advantage of the development is the ability to work with multiple threads at the same time, thus processing more data without increasing the time required to complete the process. At the same time, the user does not have the opportunity to interfere with the threads' work by using private and protected rules (Figure 9).



```
class NFA
{
  private:
    tThrd MyThread;

  protected:
    unsigned long ThrdHandle(void *arg)
    {

    };
```

Figure 9. Use of private and protected rules as an advantage of the proposed model

This reduces the risk of interruptions and faults in operation in a thread and loss of data caused by human factors.

3. CONCLUSION

The subject area was investigated. An analysis of the relevant literature has shown that the issue of management of systems is relevant, as most of them are complex in structure and have ambiguous behaviour. A system can be managed by regulating the processes that take place in it.

In order to put the objective of the research into practice, it was necessary to research the issue of multithreaded applications that manage data in threads. To organise work in the threads, it was necessary to declare the main thread within the class and specify the condition under which this thread creates an additional thread. The issue of division of functions between these threads was also considered to prevent conflict situations when switching between threads.

Multithreaded applications do work faster indeed, but their speed is ensured by multi-core processors. Therefore, it is not appropriate to consider multithreading as a means of reducing the time it takes to execute processor commands. However, the idea of parallel execution of commands is promising for application tasks.

ACKNOWLEDGMENTS

The reported research was funded by RFBR according to the research project № 19-07-00516**.**

REFERENCES

Biktashev, R.A., & Vashkevich, N.P. (2013). Models of the event non-deterministic automatons for formal presentation of the main properties of the control systems for parallel processes and resources. *Info-communication technologies, 11*(3), 95- 98.

Dadenkov, S.A., & Kon, E.L. (2015). Analysis of models and methods of agent-based and discrete event simulation. Bulletin of SPbGTU "LETI", 5, 35-41.

Dubinin, V.N., & Drozdov, D.N. (2016). Design and implementation of control systems for discrete event systems based on hierarchical modular non-deterministic automatons (Part 1. Formal model). *Bulletin of Higher Educational Institutions. Volga Region. Technical sciences, 1*(37), 28-39.

Dubinin, V.N., Budagovsky, D.A., Drozdov, D.N., & Artamonov, D.V. (2016). Design and implementation of discrete event management systems based on hierarchical modular non-deterministic automatons (Part 2. Methods and Means) Bulletin of Higher Education Institutions. *Volga Region. Technical sciences, 2*(38), 18-32.

Dukhanov, A.V., & Medvedeva, O.N. (2010). *Simulation modelling of the complex systems: lecture course.* Vladimir State University. - Publishing House of Vladimir State University, - 115 p.

Kizilov, E., Konnov, N., Nikishin, K., Pashchenko, D., Trokoz, D. (2016). Scheduling queues in the Ethernet switch, considering the waiting time of frames. *MATEC Web of Conferences, 44*, 01011.

Kizilov, E., Konnov, N., Pashchenko, D., Trokoz, D. (2015). *Modelling of QoS in the industrial Ethernet switches.* 2015 5th International Workshop on Computer Science and Engineering: Information Processing and Control Engineering, WCSE 2015-IPCE.

Martin Fowler. (2011). Subject-oriented programming languages: Translated from English - M.: I.D. Williams LLC, - 576 p.

Martyshkin Alexey, I., Pashchenko Dmitry, V. (2019). Development and research of means for the collection and reporting of open-source information for use in decision-making systems. *Journal of Computational and Theoretical Nanoscience, 16*(7), 3103-3114.

Martyshkin, A.I. (2016). Development and research of open-loop models the subsystem processor-memory of multiprocessor systems architectures uma, numa and suma. *ARPN Journal of Engineering and Applied Sciences, 11*(23), pp. 13526-13535.

Martyshkin, A.I. (2016). Mathematical modelling of Tasks Managers with the strategy of separation in space with a homogeneous and heterogeneous input thread and finite queue. *ARPN Journal of Engineering and Applied Sciences, 11*(19), pp. 11325-11332.

Martyshkin, A.I., & Yasarevskaya, O.N. (2015). Mathematical modelling of task managers for multiprocessor systems on the basis of open-loop queuing networks. *ARPN Journal of Engineering and Applied Sciences, 10*(16), pp. 6744-6749.

Martyshkin, A.I., Salnikov, I.I., Pashchenko, D.V., & Trokoz, D.A. (2018). Associative Co-processor on the Basis of Programmable Logical Integrated Circuits for Special Purpose Computer Systems. Proceedings - 2018 Global Smart Industry Conference, GloSIC 2018, 8570067.

Mikhailov, V.N. (2015). *Simulation modelling: Training and methodological manual.* - Orel: Publishing House of OF RANHGiS, - 164 p.

Modelling of the distributed multicomponent program systems and their testing on the basis of the automatic probability models / S.M. Staroletov. - Barnaul: Publishing House of Altai GTU, 2011. - 107 p.

Mozgovoy, M.V. (2006). Classics of programming: algorithms, languages, automata, compilers. Practical approach. - SPb.: Science and Technology, - 320 p.

Pashchenko, D., Trokoz, D., Konnov, N., Sinev, M. (2015). Formal transformation inhibitory safe Petri nets into equivalent not inhibitory. *Procedia Computer Science, 49*(1), c. 99-103. DOI: 10.1016/j.procs.2015.04.232.

Pashchenko, D., Trokoz, D., Sovetkina, G., Nikolaeva, E., Sinev, M., Dubravin, A., Konnov, N. (2016). The methodology of multicriterial assessment of Petri nets' apparatus. MATEC Web of Conferences, 44, 01009.

Pashchenko, D.V., Jaafar, M.S., Zinkin, S.A., Trokoz, D.A., Pashchenko, T.U., Sinev, M.P. (2016). Directly executable formal models of middleware for MANET and Cloud Networking and Computing. *Journal of Physics: Conference Series, 710*(1), 12024.

Pashchenko, D.V., Martyshkin, A.I., Trokoz, D.A. (2020). Decomposition of Process Control Algorithms for Parallel Computing Systems Using Automata Models. Proceedings - 2020 International Russian Automation Conference, RusAutoCon 2020, 839-845, 9208165.

Pashchenko, D.V., Trokoz, D.A., Martyshkin, A.I., Sinev, M.P., & Svistunov, B.L. (2020). Search for a substring of characters using the theory of non-deterministic finite automata and vector-character architecture. *Bulletin of Electrical Engineering and Informatics, 9*(3), 1238-1250.

Staroletov, S.M. (2011). Modelling of the distributed multicomponent software systems and their testing on the basis of the automatic probability models (in Russian). Barnaul: Publishing House of Altai STU, - 107 p.

Vashkevich, N.P. (2004). *Non-deterministic finite automata and vector-character architecture.* 24. - Penza: Penza State Technical University Publishing House, - 280 c.

Vashkevich, N.P., & Biktashev, R.A. (2011). Advantage of the formal language based on the concept of non-determinism in the structural implementation of the parallel systems of logical processes and resources management / Bulletin of higher educational institutions. Volga Region. *Technical sciences*, *1,* 3-11.

Vashkevich, N.P., & Biktashev, R.A. (2011). Dignity of formal language based on the concept of non-determinism for functional description and transformation of algorithms of processes and resources control in parallel systems. *Telecommunications, 1*, 18-25.

Vashkevich, N.P., & Biktashev, R.A. (2016). *Non-deterministic automatons and their use for implementation of systems of parallel information processing: Monograph - Penza*: Publishing House of PSU, - 394 P.

Vashkevich, N.P., & Vashkevich, S.N. (1996). Non-deterministic automatons and their use for synthesis of control systems. Part 1: Equivalent transformations of non-deterministic automatons: Training manual. - Penza: Publishing House of Penza State Technical University, - 88 p.

Vashkevich, N.P., Biktashev, R.A., Pashchenko, D.V., Kutuzov, V.V., Sauanova, K.T. (2015). Use of models of non-deterministic event automata for formal description of parallel algorithms of logical control. *Bulletin of NAS RK, 4,* 48-63.

Volchikhin, V.I., Vashkevich, N.P., & Biktashev, R.A. (2013). Models of the non-deterministic event automatons for the representation of the control algorithms of the interacting processes in the multi-processor computer systems on the basis of the monitor mechanism. *Bulletin of Higher Education Institutions. Volga Region. Technical sciences, 2* (26), 5-14

Zimina, L.V. (2017). Features of development of discrete-event models in AnyLogic toolkit. *Education and science without borders: fundamental and applied research*, *6*, 194-198.