ISSN 2076-2429 (print)
ISSN 2223-3814 (online)

UDC 004.942

P. Shvahirev, Assoc. Prof.,
O. Lopakov,
V. Kosmachevskiy,
V. Salii
Odessa National Polytechnic University, 1 Shevchenko Ave., Odessa, Ukraine, 65044; e-mail: kedrodess9@gmail.com

METHOD FOR ASSESSING OF RELIABILITY CHARACTERISTICS IN DESIGNING OF FAILURE-RESISTANT REAL-TIME OPERATING SYSTEMS

П.А. Швагірев, О.С. Лопаков, В.В. Космачевський, В.І. Салій. **Методика оцінки характеристик надійності в проектуванні автоматичних систем реального часу.** Протягом багатьох років додатки на основі ОС у режимі реального часу використовуються у вбудованих системах спеціального призначення, а останнім часом їх застосовують повсюдно - від бортових систем управління літаком, до побутових приладів. Розробка багатопроцесорних обчислювальних систем (БПС) зазвичай має на меті підвищити рівень надійності або рівень продуктивності системи до значень, недоступних або важких для впровадження в традиційних комп'ютерних системах. У першому випадку виникає питання про наявність спеціальних засобів забезпечення відмовостійкості комп'ютерних систем, основною особливістю (і перевагою) яких є відсутність будь-якого єдиного ресурсу, вихід з ладу якого призводить до фатального виходу з ладу всієї системи. Використання операційної системи в режимі реального часу завжди пов'язане з обладнанням, об'єктом, а також з подіями, що відбуваються на об'єкті. Система в режимі реального часу, як апаратно-програмний комплекс, включає датчики, що записують події на об'єкті, модулі введення / виведення, які перетворюють показання датчиків у цифрову форму, придатну для обробки цих показань на комп'ютері, і, нарешті, комп'ютер з програмою, яка реагує на події, що відбуваються на об'єкті. RTOS орієнтована на обробку зовнішніх подій. Саме це призводить до принципових відмінностей (порівняно із ОС загального призначення) у структурі системи, у функціях ядра, у побудові системи введення-виведення. RTOS може бути схожа за своїм користувальницьким інтерфейсом на операційні системи загального призначення, але за своєю структурою вона зовсім інша. Крім того, використання RTOS завжди є специфічним. Якщо ОС загального призначення зазвичай сприймається користувачами (а не розробниками) як готовий набір програм, то RTOS служить лише інструментом для створення конкретного апаратно-програмного комплексу в режимі реального часу. А отже, найширший клас користувачів RTOS - це розробники комплексів реального часу, люди, що проектують системи контролю та збору даних. Розробляючи та проектуючи конкретну систему в режимі реального часу, програміст завжди точно знає, які події можуть відбутися на об'єкті, і він знає критичні умови обслуговування кожного з цих подій. Ми називаємо систему реального часу (SRV) апаратно-програмним комплексом, який реагує в передбачувані часи на непередбачуваний потік зовнішніх подій. Система повинна встигнути відреагувати на подію, що сталася на об'єкті, протягом критичного для цієї події часу. Критичний час для кожної події визначається об'єктом і самою подією, і, звичайно, він може бути різним, але час реакції системи необхідно передбачити (обчислити) при створенні системи. Відсутність реакції в передбачуваний час вважається помилкою для систем у режимі реального часу. Система повинна встигнути реагувати на події, що відбуваються одночасно. Навіть якщо дві або більше зовнішніх подій відбуваються одночасно, система повинна встигнути реагувати на кожну з них протягом часових інтервалів, критичних для цих подій. У цьому дослідженні, розглядається ,як частина мережевої відмовостійкої технології, RTOS стає особливим типом програмного забезпечення для управління, яке використовується для організації роботи вбудованих додатків, які характеризуються обмеженими ресурсами пам'яті, низькою продуктивністю та вимогами гарантованого часу відгуку (T<4 мкс), високим рівнем доступності та наявністю засобів моніторингу.

Ключові слова: операційна система реального часу, відмовостійкість, критерій надійності операційної системи, ранг відмовостійкості, операційна система з монолітним ядром, самовідновлення операційної системи, агент відновлення

P. Shvahirev, O. Lopakov, V. Kosmachevskiy, V. Salii. **Method for assessing of reliability characteristics in designing of failure-resistant real-time operating systems.** For many years, real-time OS-based applications have been used in embedded special-purpose systems. Recently they have been used everywhere, from on-board control systems for aircraft, to household appliances. The development of multiprocessor computing systems usually aims to increase either the level of reliability or the level of system performance to values that are inaccessible or difficult to implement in traditional computer systems. In the first case, the question of the availability of special means of ensuring the fault tolerance of computer systems arises, the main feature (and advantage) of which is the absence of any single resource, failure of which leads to a fatal failure of the entire system. The use of a real-time operating system is always associated with equipment, with an object and with events occurring at an object. A real-time system, as a hardware-software complex, includes sensors that record events at an object, input / output modules that convert sensor readings into a digital form suitable for processing these readings on a computer, and finally, a computer with a program that responds to events occurring at the facility. The RTOS is focused on processing external events. It is this that leads to fundamental differences (compared with general-purpose OS) in the structure of the system as well as in the functions of the kernel and in the construction of the input-output system. The RTOS can be similar in its user interface to general-purpose operating systems, but it is completely different in its structure. In addition, the use of RTOS is always specific. If users (not developers)

DOI: 10.15276/opu.2.61.2020.13

usually perceive a general-purpose OS as a ready-made set of applications, then the RTOS serves only as a tool for creating a specific hardware-software complex in real time. Therefore, the widest class of users of RTOS is the developers of real-time complexes, people designing control and data collection systems. When designing and developing a specific real-time system, the programmer always knows exactly what events can occur at the facility, and he knows the critical terms for servicing each of these events. We call a real-time system (SRV) a hardware-software complex that responds in predictable times to an unpredictable stream of external events. The system must have time to respond to the event that occurred at the facility, during the time critical for this event. The critical time for each event is determined by the object and by the event itself, and, of course, it can be different, but the response time of the system must be predicted (calculated) when creating the system. Lack of response at the predicted time is considered an error for real-time systems. The system must have time to respond to simultaneously occurring events. Even if two or more external events occur simultaneously, the system must have time to respond to each of them during time intervals critical for these events. In this study, as part of a network fault-tolerant technology, the RTOS becomes a special type of control software that is used to organize the operation of embedded applications, which are characterized by limited memory resources, low productivity and the requirements of a guaranteed response time (T<4 μs), high level availability and availability of auto-monitoring facilities.

Keywords: Real-time operating system, fault tolerance, operating system reliability criterion, fault tolerance grade, operating system with a monolithic kernel, operating system self-healing, recovery agent

Introduction

Basically modern operating systems are still described on architectural principles and solutions, as at the beginning of their development. At that time, their choice was justified by the state of development, the development of computers and their cost, in favor of maximizing efficiency and density implementation, reduction of any overhead computational expenses [1, 2]. Currently, the situation has changed significantly. Still use principles that are no longer adequate in relation to the current state of development. First of all, we are talking about general approach to the design and implementation of operating systems in favor of ensuring reliability and fault tolerance [3, 4].

Analysis of publications

The design of a fault-tolerant architecture is based on a multi-layer modularity. It allows all system components to be split into independent software components that are dynamically linked at initial initialization. Based on the experience gained, the concept of a single fault-tolerant runtime environment was developed.

Studies reported at international symposia on software reliability show that the bulk of OS errors (more than 70 %) are concentrated in drivers. The error rate in them is 3…7 times higher than in ordinary codes. Renowned expert Andrew Tanenbaum estimated [1, 2] the number of bugs per 1000 lines of code in the Linux kernel from 6 to 16. According to the most conservative estimates, the Linux kernel has about 15.000 bugs [15].

Unsolved problem area is that when creating a specialized RTOS, the principle was not considered system failure prevention (fault tolerance), since recovery in a number of cases may be associated with significant processor time or interruption of the computing process. In addition, the mechanisms remained unresolved ensuring fault tolerance, the main of which are the protocols of voting and collective decision-making. The effect of increasing the number of processor elements (PE) for the total uptime of the system.

Purpose of the article is devoted to the study of reliability criteria for a specialized distributed real-time operating system for fault-tolerant systems with a fault tolerance rank of N (N–1), which means the ability of the system to function even if all elements of the system fail, except for one. The following tasks were set in order to cover the selected topic fully:

1. Analyzing existing real-time operating systems, highlighting the main functional requirements for them and giving a comparative description.

2. Conducting reliability analysis of fault-tolerant system and giving recommendations on the organization of system.

In the article necessary to consider the approaches that underlie the design of guaranteeing systems and that have found practical applications for critical applications. These approaches are embodied at following various hierarchical levels of system representation:

1. The level of architecture of the computing environment.

2. The level of the operating system or its equivalent monitor that implements system methods of fault tolerance.

ISSN 2076-2429 (print)
ISSN 2223-3814 (online)

3. The level of hardware or control memory, which remains “transparent” not only for the user but also for the OS.

Let analyze some of the modern approaches to improve the resiliency of operating systems.

Operating system architecture and methodology for calculating reliability characteristics

Swift and Bershad [1] propose to use the approach of driver isolation. They created a prototype subsystem called Nooks. Nooks [6] is a monolithic kernel subsystem that allows kernel extensions, such as dynamic device drivers and file system drivers, to run in isolation in the kernel of the operating system. In a traditional monolithic kernel, any error in the driver can damage important kernel structures. To reduce the risk of the consequences of errors, Nooks allows drivers to run in specific areas for which restrictions on recordings are set in the kernel address space. Nooks tracks all access attempts or failures and provide the ability to recover automatically. In their approach, the authors do not propose developing new system architecture, but rather improve the reliability of existing systems, mainly reducing the likelihood of fatal failures associated with system drivers. The authors highlight three key principles that Nooks complies. The first principle is compatibility. The architecture must be compatible with existing systems and extensions or with minimal changes (see Fig. 1).

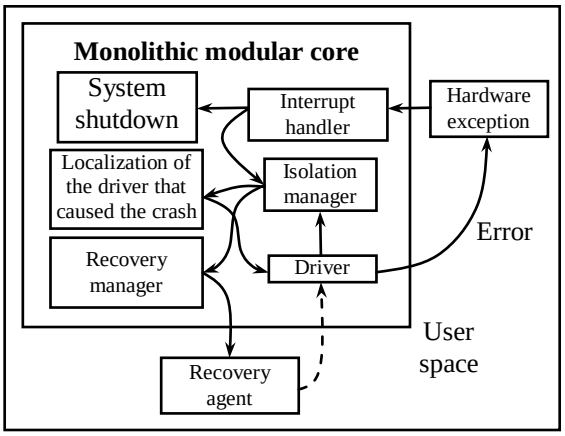

Fig. 1. Organization of an operating system with isolated drivers

Modern operating systems need to solve a large number of complex tasks. Their unreliability together with the complexity and size of the system increases sharply due to increasing factors that can lead the system to failure.

According to studies of software reliability, for every 1000 lines of code there are from 6 to 16 errors, while the size of a modern OS is more than 10 million lines of code.

The current situation is such that inefficient architectural solutions are used in the most advanced operating systems, which are unable to provide fully a high level of fault tolerance due to their specifics. The simultaneous combination and satisfaction of all requirements with a high level of ensuring fault tolerance as well as information security and speed is a complex research and practical problem [7]. The problem of improving the efficiency of fault tolerance mechanisms is the subject of many studies. As is well known, the most common OS architectures are monolithic and microkernel. The advantages of monolithic architecture are widespread use in the OS environment, as well as high speed. The main disadvantage is the lack of a breakdown of functionality that meets the requirements of fault tolerance. Almost any error in the core of the system leads to the failure of the entire system. One of the advantages of microkernel architecture is a higher fault tolerance in comparison with monolithic, which is provided by transferring the basic subsystem to the level of system processes. The main disadvantage of microkernel architecture [8] is significantly slower performance compared to monolithic, due to the expensive waiting time for switching protection and execution contexts, as well as the processing time of requests between processes. The problem of the mechanism of ensuring fault tolerance in microkernel and monolithic architecture is that the process of failure processing is reduced to a simple restart. As a rule, a failed process loses its current state and current data [9]. Formally, the problem of increasing fault tolerance can be expressed as an increase in the dimension of the sets of localized failures L and recoverable failures R, which are included in the set of all possible failures E.

The calculation of system fault tolerance indicators can be performed in two stages:

At the first stage, the calculation of the set of probable errors E is carried out with the dynamics of the growth of system volumes [10]. The increase in errors N is explained by the assumption that with an increase in the size of the system, the number of errors increases, D is the number of isolation areas of system failures [11]:

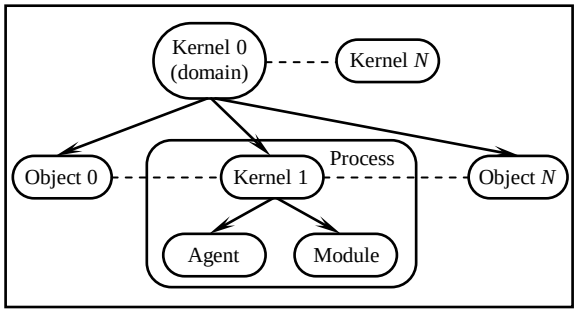

Fig. 2. Domain model of fault tolerant operating system

$$E=\frac{N}{D},$$

$$R=\frac{L}{R_{\text{recovered}}},$$

$$L=\frac{E}{L_{\text{localized}}},$$

$$F=\frac{R}{L}.$$

At the second stage, the fault tolerance index F is calculated. To calculate the set of localizable failures L, the coefficient of the degree of localization of errors was used $L_{\text{localized}}$. To calculate the set of recoverable faults R, the coefficient of the degree of recoverability of localized faults $R_{\text{recovered}}$ was used. The indicators $L_{\text{localized}}$ and $R_{\text{recovered}}$ tend to unity [10], depending on the volume of the isolated area. In other words, the indicators correspond to the approximate volume of the system and the degree of ability to determine and localize possible errors and determine the recoverable. That is, the simpler and smaller the functional area, the less errors should be contained in it. The main difference between domain architecture from monolithic and microkernel is that the traditional organization of the OS kernel is significantly modified in favor of a deeper decomposition. At the same time, the analogy of the system core as an application request handler is transferred to the so-called domain objects [12]. Any set of domain objects can be defined in the system, each of which is responsible for processing requests to child objects, including domain ones. In accordance with the need to increase the failure rate OS stability and based on the analysis and identified shortcomings, as well as received by us previous experience in the design and implementation of OS, was developed the "domain" architecture of the OS has been worked out (see Fig. 2).

For a preliminary assessment of indicators fault tolerance of system architectures was carried out experimental modeling. Initial data and the simulation results are presented in Table 1.

Table 1

The source data and the results of the simulation of fault tolerance

System	Domain levels	Error rate, E	Failure Localization Ratio, $L_{localized}$	Recovery rate of localized failures, $R_{recoverad}$	L	R	F
monolithic		10000	24	1.55	357.14	106.60	0.29
micronuclear		2500	7	1.35	357.14	264.55	0.74
domain	1	2500	7	1.35	357.14	264.55	0.74
domain	2	1250	6	1.3	208.33	160.25	0.76
domain	4	625	5	1.25	125	100	0.8
domain	8	312.50	4	1.2	78.12	65.10	0.83
domain	16	156.25	3	1.15	52.08	45.28	0.86
domain	32	78.12	2	1.1	39.06	35.50	0.9
domain	64	39.06	1	1.05	39.06	37.20	0.95

Assessment of the reliability characteristics of fail-safe OS

The chosen concept of constructing a specialized distributed real-time operating system will allow a homogeneous system to function when there is an N–1 failure of processor elements in the system. If the probability of disconnecting workable processor modules is not taken into account, an optimistic estimate of the probability of failure of the entire system for a certain period of operation and average operating time system failure [11] can be conducted. Suggesting that the failure flow at each

ISSN 2076-2429 (print)
ISSN 2223-3814 (online)

node of the system is the simplest, i.e. stationary, ordinary and without consequences, with the exponential law of the distribution of the length of the interval between adjacent events (failures) [13]:

$$P_K(t)=\frac{(t)^K}{K!}e^{(-t)},$$
$$T_0=\frac{1}{\lambda},\quad P_0=e^{(-t)}, \tag{1}$$

where $P_K(t)$ – the probability that exactly "K" events (failures) will occur during time t;

λ – flow parameter, failure flow rate;

T_0 – the mathematical expectation of the length of the interval between adjacent events – the mean time between failures;

$P_0(t)$ – the probability that not a single event (failure) will occur during time t, the probability of failure-free operation.

Denote by T_0^y – mean time between failures of one system node. For fault-tolerant systems, the failure state will be understood as the state of fatal failure, i.e. for OS – $N(m)$, this is the state in which more than "m" system nodes (m+1, m+2, ...) failed. At an arbitrary instant of time t, we can find the system in one of two states:

– workable with probability $R(t)$,

– in a state of fatal failure, with probability $P(t)$.

If we look at the system, taking into account the health states of each of its N elements (nodes), then at an arbitrary instant of time t we can find the system in one of $2N$ states (see Fig. 3).

If we match each node of the system with a discharge of a binary N bit number (0 – the node is working, 1 – the node has failed), then each such state of the system can be associated with its own number equal to the value of the entered binary N bit number. To each such state, there is some probability the system is at time t in this state [13].

All $2N$ states of the system can be divided into several groups of states, each of which differs from the others in the number of failed nodes. The zero group (group number 0) contains one state ($C_N^0=1$), in which all nodes of the system are in a healthy state, i.e. There are exactly 0 failed elements. The first group includes all states that exactly one node failed (the binary numbers of these states contain only one unit in the N bit binary code). The number of states included in the first group is $C_N^1=N$ – the number of combinations of N by 1 (C_n^m).

The second group consists of states in which there are two failed elements in the system. These states are exactly C_N^0 etc.

The i-th group includes all states in which exactly i nodes failed in the system, such states C_N^i.

The penultimate (N–1) – th group includes C_N^{N-1} states, i.e. N states.

The last Nth group contains one state ($C_N^N=1$), in which all N nodes of the system have failed.

Because at any time, the system can only be in one of all $2N$ states, then these events are incompatible [14]. Therefore, the probability of finding a system in any of the states belonging to one of the groups mentioned above can be obtained as the sum of the probabilities of finding the system in all states of this group. If we take into account that inside each i-th group all states are characterized by the presence of exactly i failed nodes, then the probabilities for all states of one group are equal, therefore:

$$P_i=C_N^i\cdot P_i^1\ , \tag{2}$$

where P_i – the probability of finding the system (at an arbitrary time t) in any of the states assigned to the i-th group;

P_i^1 – the probability of the system being in one specific state, assigned to the i-th group.

All states assigned to the i-th group are characterized by the presence in the system (at an arbitrary instant of time t) of exactly i failed nodes and exactly (N–i) serviceable nodes.

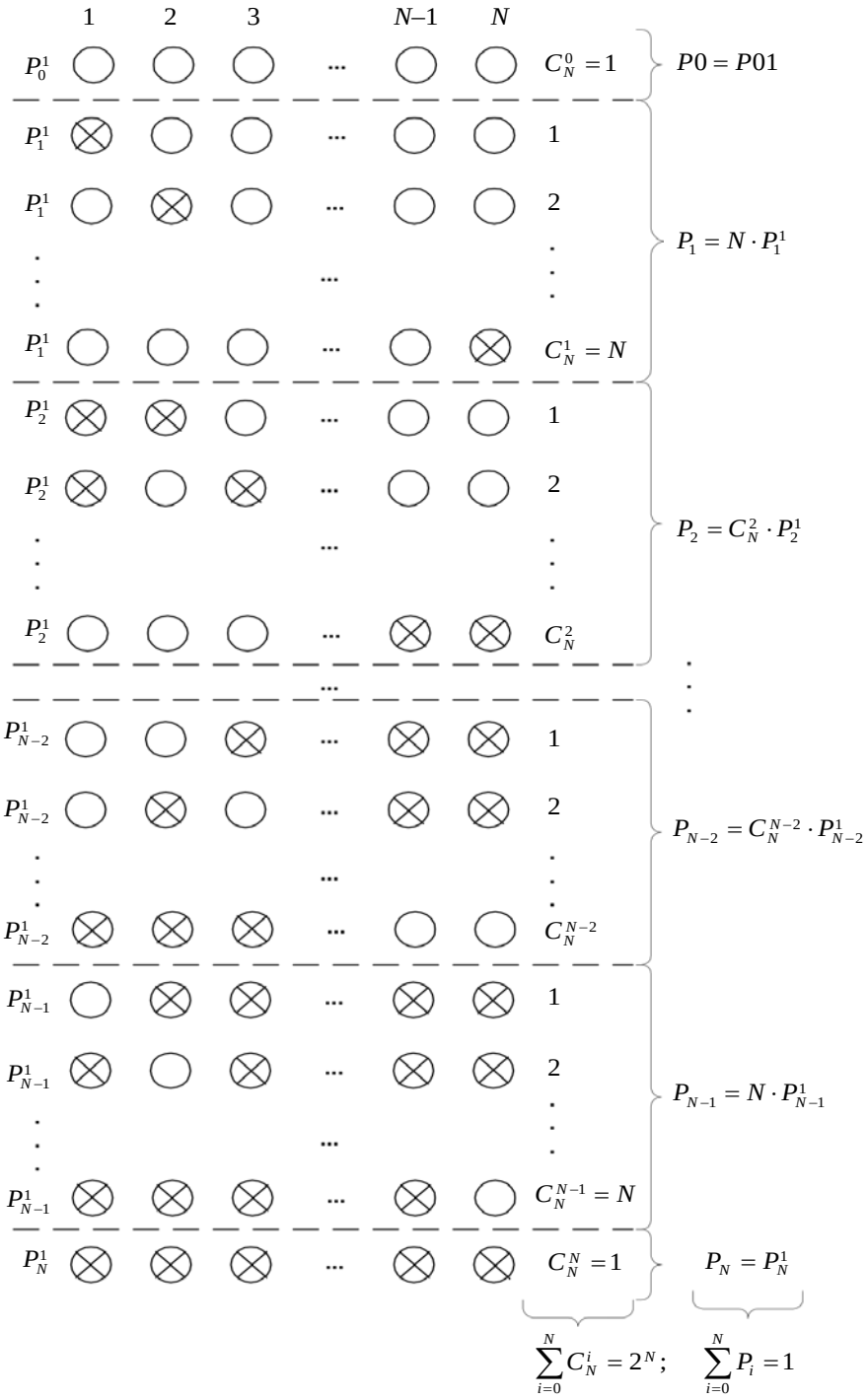

Fig. 3. N-node system states

In accordance with the assumption introduced above about the simplest flow of failures (1), the probability P_i^1 can be estimated as follows [14]:

$$P_i^1(t) = \left(e^{-\frac{t}{T_0^y}} \right)^{(N-i)} \cdot \left(1 - e^{-\frac{t}{T_0^y}} \right)^i, \quad (3)$$

ISSN 2076-2429 (print)
ISSN 2223-3814 (online)

where the first bracket corresponds to the fact that (N–i) elements are in a healthy state, and the second to the fact that i elements have failed. Substituting (3) into (2), we can obtain an expression for calculating the probabilities P_i.

Obviously, for the OS – N (m) system (N node system with a fault tolerance rank m, all system states included in the groups 0, 1, 2, ..., m belong to those states in which the system functions normally [15]. In this regard, the probability $R(t)$ can be estimated as follows [15]:

$$R(t)=\sum_{i=0}^{m} P_i \ . \tag{4}$$

The probability of a fatal failure of the OS – N (m) system can be estimated as the sum of the probabilities of the system being in the states assigned to the groups m+1, m+2, ..., N–1, N [16]:

$$R(t)=\sum_{i=m+1}^{N} P_i \ . \tag{5}$$

The criterion for the correctness of the proposed method [17] is the fulfillment of the condition $R(t)+P(t)=1$ for any systems and any values of t.

Combining expressions (2), (3), (4) and (5), we obtain the final formulas for calculating the probabilities of failure-free operation – $R^{N(m)}(t)$ and fatal failure – $P^{N(m)}(t)$ OS – N (m) systems for an arbitrary time t:

$$\begin{aligned} R^{N(m)}(t) &= \sum_{i=0}^{m} C_N^i \left(e^{-\frac{t}{T_0^y}} \right)^{N-i} \cdot \left(1-e^{-\frac{t}{T_0^y}} \right)^{i}, \\ R^{N(m)}(t) &= \sum_{i=m+1}^{N} C_N^i \left(e^{-\frac{t}{T_0^y}} \right)^{N-i} \cdot \left(1-e^{-\frac{t}{T_0^y}} \right). \end{aligned} \tag{6}$$

For practical calculations [18], it is advisable to use one of these formulas. Namely, one that has (depending on the values of N and m) fewer summable terms, i.e. at:

$$m \geq \frac{N-1}{2},$$

it is advisable to use the formula $P^{N(m)}(t)$ otherwise, the formula $R^{N(m)}(t)$. In this case, the second parameter is obtained from the relation $R^{N(m)}(t)+P^{N(m)}(t)=1$.

Thus, for systems of type N (N–1), expressions (6) take the form:

$$\begin{aligned} R^{N(m)}(t) &= 1-P^{N(m)}(t), \\ P^{N(m)}(t) &= \left(1-\mathrm{e}^{-\frac{t}{T_0^y}} \right)^{N}. \end{aligned} \tag{7}$$

Let us now consider the determination of the mean time between failures $T_0^{N(m)}$ of fault-tolerant OS – N(m) systems.

The non-recoverable N-node fault-tolerant system of the m-th rank (OS – N(m)) can be represented by a Markov model with the number of states (N+1) (see Fig. 4).

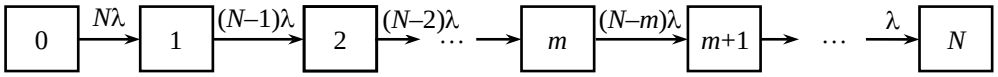

Fig. 4. Unrecoverable N-node fault-tolerant system of rank m (OS-N (m)): 0 – a state in which no node in the system has failed; 1 – condition (uniting a group of system states C_N^1 – , in which exactly 1 node failed; 2 – a state (uniting a group of system states C_N^2), in which exactly 2 nodes failed; m – a state (uniting a group of system states C_N^m), in which exactly m nodes failed, etc

The transition from one state to another (with the gradual degradation of the system) is determined by the intensity of the flow of failures affecting the system in the corresponding state 8. The intensity of the flow of failures affecting the system in the *i*-th state is determined by the number of workable nodes (*N*–*i*). The average time the system is in the *i*-th state is determined as follows:

$$T_0^i = \frac{1}{(N-1)} = \frac{T_0^y}{(N-1)}, \tag{8}$$

where $\lambda = \frac{1}{T_0^y}$ – failure rate of one system node [19].

A fatal failure of the OS – *N*(*m*) system will occur only when the system transitions from state m to state *m*+1, therefore, the average time between failures of the OS – *N*(*m*) system is equal to the average time the system was sequentially in states 0, 1, 2 ..., *m*:

$$T_0^{N(m)} = \sum_{i=0}^{m} T_0^i = T_0^y \cdot T_0^y \cdot \sum_{i=0}^{m} \frac{1}{N-i}. \tag{9}$$

Expression (8) was obtained on the basis of following fundamental property of the exponential distribution law: “if the period of time distributed according to the exponential law has already lasted some time t, then this does not affect the distribution law of the remaining part of the interval: it will be the same as the law distribution of the entire gap” [20]. This property of the exponential law is essentially one of the formulations for the “absence of aftereffect”, which is the main property of the simplest flow that we have adopted as a model of the failure flow.

If you enter the designation:

$$K_{N(m)} = \sum_{i=0}^{m} \frac{1}{N-1}, \tag{10}$$

then this “reliability coefficient” in accordance with (8) is the ratio of $T_0^{N(m)}$ to T_0^y:

$$K_{N(m)} = \frac{T_0^{N(m)}}{T_0^y},$$

and shows how many times compared to T_0^y – the average time between failures of one node, the average time between failures of the OS – *N*(*m*) system as a whole has changed [20].

Use formulas (7) and (10), it is possible to evaluate the reliability characteristics of fault-tolerant systems of type *N* (*N*–1).

We take the average time between failures of the node $T_0^y = 10^5$ hours. We calculate the values of the characteristics according to the formulas (7) and (10).

To simplify the data analysis, we will construct two graphs reflecting the increase in the system reliability with the increase in the hardware part (see Fig. 5 and Fig. 6).

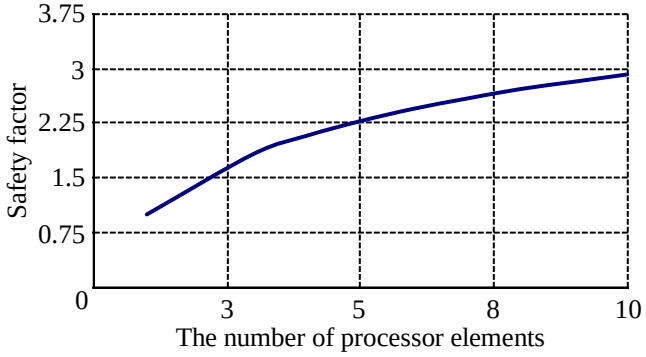

Fig. 5. Reliability coefficient

ISSN 2076-2429 (print)
ISSN 2223-3814 (online)

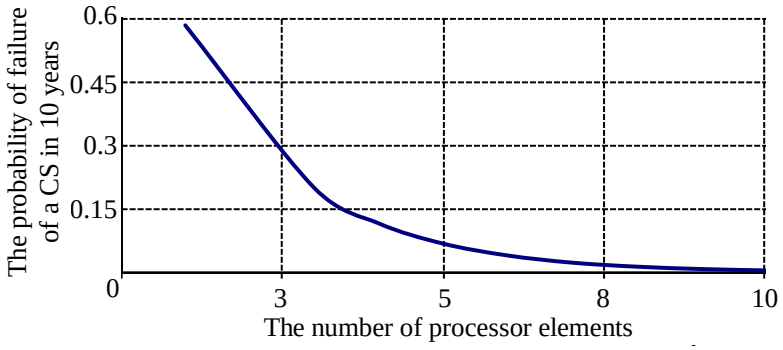

Fig. 6. The probability of failure of an aircraft of type N (N–1) for 10 years

Results of methodology for calculating reliability characteristics

Domain system when number of domain levels:

1. $D = 1$, corresponds to the fail-safe micronuclear system.

2. $D > 1$ is able to reduce by about N times the same probable errors in comparison with micro kernel system and, accordingly, get the value of the resiliency body that will strive for unit (to full coverage of failures). What makes it possible ness significantly increase fault tolerance in comparison with other architectural approaches.

The analysis of the curves shows that the average uptime is 2…3 times higher than the average uptime of one processor element while building up computing resources 5…7 times and then stabilizes and increases slightly. The probability of failure of systems with a fault tolerance rank of N (N–1) decreases sharply when considering aircraft, and then its decrease is insignificant. The probability of a system failure over 10 years with the uptime of one processor element (PE) of 10,000 hours was 0.068, which is 8.5 times less than the probability of a single PE failure over the same period.

Conclusions

A full-scale experiment to determine the reliability of operating systems of various architectures is practically impossible due to unacceptable time and financial costs. Simulation does not simplify the task also. Therefore, the only way to solve this problem is to develop simple and adequate models.

We propose approach to the construction of models of operating system architectures under the adoption of restrictions on the absolute reliability of the equipment, the random nature and independence of failures in programs without taking into account their recovery and the absence of parallel processes in the operation of OS modules. Approach can be used for an initial study of the reliability of the operation of specific operating system architectures.

Improving the accuracy of modeling results in such models is related to the refinement of the initial data, which determine the mean time between failures of the modules of the operating system under study and the intensity of transitions of the system under study from one state to another. To clarify the reliability characteristics of system modules, it is necessary to build models of reliability changes in identifying and eliminating software errors and the corresponding statistics of the operating system developer.

The work considered the so-called “domain” approach, which is designed to increase the fault tolerance of execution systems. The essence of the approach is a mechanism that allows you to organize reflexive processing of system requests for the controlled space of delegated objects. Based on the experience gained, the concept of a single fault-tolerant runtime was developed.

A probabilistic model of fault-tolerant aircraft was proposed in the article, its reliability characteristics were calculated, which showed an increase in the average time between failures of the aircraft by 2.5…3.5 times with expansion of the aircraft to 5…7 nodes, recommendations are given for choosing the type of aircraft during its design.

Література

1. Swift M., Bershad B., Levy H. Improving the Reliability of Commodity Operating Systems. *ACM Transactions on Computer Systems*. 2005. Vol. 23, No. 1. P. 77–110.

2. Cordy J., Shukla M. Practical metaprogramming. *IBM Centre for Advanced Studies*. 1992. P. 2–9.
3. Reekie H., Hylands C., Lee E. Tcl and Java Performance. University of California at Berkley. 1998.
4. Krall A. Efficient JavaVM Just-in-Time Compilation. PACT, 1998, 205 p.
5. Muller G., Moura B., Bellard F., Consel C. JIT vs Offline Compilers: Limits and Benefits of Bytecode Compilation. *IRISA*. 1997. PI 1063.
6. Leroy X. Java Bytecode Verification: Algorithms and Formalizations. *JOAR*. 2005. Vol. 30. P. 3–9.
7. An experimental Study of Soft Errors in Microprocessors / G.P. Saggese, N.J. Wang, Z.T. Kalbarczyk et al. *IEEE Micro*. 2005. V. 25. №6. P.30–39. DOI: 10.1109/MM.2005.104.
8. Basili V.R., Perricone B.T. Software errors and complexity: an empirical investigation. *ACM 27*. 1984. P. 42–52.
9. Suri N., Valter C.J., Hugu M.M. Advances in Ultra Dependable Distributed Systems. *Computer Society Press*. 1995. P. 56–61.
10. Таненбаум Э., Вудхалл А. Операционные системы. Разработка и реализация. 3-е изд. СПб : Питер, 2007. С. 254–256.
11. Таненбаум Э., Бос Э. Современные операционные системы. 4-е изд. СПб : Питер, 2015. 457 с.
12. Назаров С.В., Широков А.И. Технологии многопользовательских операционных систем. М. : Изд. Дом МИСиС, 2012. С. 98–101.
13. Назаров С.В., Вилкова Н.Н. Структурный рефакторинг многослойных программных систем. *Информационные технологии и вычислительные системы.* 2016. № 4. С. 13–23. URL: https://elibrary.ru/item.asp?id=27656660 (дата звернення: 05.01.2020).
14. Кельберт М.Я., Сухов Ю.М. Вероятность и статистика в примерах и задачах. Том 2. Марковские цепи как отправная точка теории случайных процессов и их приложения. М. : МЦНМО. 2009. С. 145–147.
15. Назаров С.В. Эффективность современных операционных систем. *Современные информационные технологиии ИТ-образование.* 2017. Т.13, №2. С. 9–24. DOI: https://doi.org/10.25559/SITITO.2017.2.229
16. Василенко Н.В., Макаров В.А. Модели оценки надежности программного обеспечения. *Вестник Новгородского государственного университета. Серия: Технические науки.* 2004. № 28. С. 126–132. URL: https://elibrary.ru/item.asp?id=18184720 (дата звернення: 05.01.2020).
17. Таха Х. Введение в исследование операций. 7-еизд. М. : Вильямс, 2005. С. 34–67.
18. Мартышкин А.И. Математическая модель диспетчера задач с общей очередью для систем параллельной обработки. *Современные методы и средства обработки пространственно-временных сигналов*: сборник статей XI Всероссийской научно-технической конференции. Пенза : ПДЗ, 2013. С. 87–91.
19. Андреев А.М., Можаров Г.П., Сюзев В.В. Многопроцессорные вычислительные системы: теоретический анализ, математические модели и применение. Москва, Изд-во МГТУ им. Н.Э. Баумана, 2011, С. 124–156.
20. Воеводин В.В. Параллельные вычисления. СПб. : БХВ – Петербург, 2002. С. 145–152.

References

1. Swift, M., Bershad, B., & Levy, H. (2005). Improving the Reliability of Commodity Operating Systems. *ACM Transactions on Computer Systems, 23*, 1, 77–110.
2. Cordy, J., & Shukla, M. (1992). Practical metaprogramming. *IBM Centre for Advanced Studies*, 2–9.
3. Reekie, H., Hylands, C., & Lee, E. (1998). *Tcl and Java Performance.* University of California at Berkley.
4. Krall, A. (1998). *Efficient JavaVM Just-in-Time Compilation.* PACT.
5. Muller, G., Moura, B., Bellard, F., & Consel, C. (1997). JIT vs Offline Compilers: Limits and Benefits of Bytecode Compilation. *IRISA*, PI 1063.
6. Leroy, X. (2005). Java Bytecode Verification: Algorithms and Formalizations. *JOAR, 30*, 3–9.
7. Saggese, G.P., Wang, N.J. & Kalbarczyk, Z.T. et al. (2005). An experimental Study of Soft Errors in Microprocessors. *IEEE Micro, 25*, 6, 30–39.
8. Basili, V.R., & Perricone, B.T. (1984). Software errors and complexity: an empirical investigation. *ACM 27*, 42–52.
9. Suri, N., Valter, C.J., & Hugu, M.M., (1995). Advances in Ultra Dependable Distributed Systems. *Computer Society Press*, 56–61.

ISSN 2076-2429 (print)
ISSN 2223-3814 (online)

10. Tanenbaum, E., & Vudkhall, A. (2007) *Operating systems. Development and implementation*. 3rd ed. SPb: Piter, 254–256.
11. Tanenbaum, E., & Bos, E. (2015). *Modern operating systems.* 4th ed. SPb: Piter.
12. Nazarov, S.V., & Shirokov, A.I. (2012). *Multiuser operating system technologies*. Moscow: Ed. House of MISIS, 98–101.
13. Nazarov, S.V., & Vilkova, N.N. (2016). Structural refactoring of multilayer software systems. *Information technologies and computing systems, 4,* 13–23. Retrieved from: https://elibrary.ru/item.asp?id=27656660. (Last access: 5.01.2020).
14. Kelbert, M.Ya., & Sukhov, Yu.M. (2009). Probability and statistics in examples and problems. Vol. 2. *Markov chains as a starting point of the theory of random processes and their applications.* M.: MCNMO, 145–147.
15. Nazarov, S.V. (2017). The effectiveness of modern operating systems. *Modern information technologies and IT education, 13*, 2, 9–24. DOI: https://doi.org/10.25559/SITITO.2017.2.229.
16. Vasilenko, N.V., & Makarov, V.A. (2004). Models for assessing the reliability of software. *Bulletin of the Novgorod State University. Series: Engineering Sciences, 28*, 126–132. Retrieved from: https://elibrary.ru/item.asp?id=18184720. ((Last access: 5.01.2020).
17. Taha, H. (2005). *Introduction to Operations Research.* 7th edition, M.: Williams, 34–67.
18. Martyshkin, A.I. (2013). Mathematical model of a task manager with a common queue for parallel processing systems. *Modern methods and means of processing space-time signals: collection of articles of the XI All-Russian scientific and technical conference*. Penza: PDZ, 87–91.
19. Andreev, A.M., Mozharov, G.P., & Syuzev, V.V. (2011). *Multiprocessor computing systems: theoretical analysis, mathematical models and applications.* Moscow, Publishing house of MSTU im. N.E. Baumana, 124–156.
20. Voevodin, V.V. (2002). *Parallel computing.* SPb.: BHV Petersburg, 145–152.

Швагірев Павло Анатолійович; Shvahirev Pavlo, ORCID: https://orcid.org/0000-0003-3913-4412
Лопаков Олексій Сергійович; Lopakov Oleksii, ORCID: https://orcid.org/0000-0001-6307-8946
Космачевський Володимир Володимирович; Kosmachevskiy Volodymir, ORCID: https://orcid.org/0000-0002-3234-2297
Салій Віра Іванівна; Salii Vira, ORCID: https://orcid.org/0000-0003-2426-5241

Received May 21, 2020
Accepted July 12, 2020