Indonesian Journal of Electrical Engineering and Computer Science
Vol. 14, No. 3, June 2019, pp. 1484~1492
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v14.i3.pp1484-1492

Prediction of entering processes into the deadlock state

Andrii Nicheporuk[1], Yuriy Klots[2], Oksana Yashyna[3], Sergiy Mostovyi[4], Yuriy Nicheporuk[5]
[1,2,4,5]Department of Computer Engineering and System Programming, Khmelnytskyi National University, Ukraine
[3]Department of Software Engineering, Khmelnytskyi National University, Ukraine

Article Info

Article history:

Received Jan 8, 2018
Revised Jan 21, 2018
Accepted Feb 23, 2019

Keywords:

Boundary state
Deadlock state
Fuzzy logic
Prediction algorithm
Process

ABSTRACT

The work is devoted to the problem of processes' deadlock in the multi-tasking system. On the basis of the study of the life cycle of the process, the boundary state of the process was distinguished, which will precede the deadlock state. Proposed the method for prediction of entering processes into the deadlock state, which consists of two algorithms: algorithm of detection of potential processes that may fall into the deadlock and algorithm of processes detection that falling into the state of deadlock. An evaluation of time complexity of proposed method is conducted. Unlike the known methods and algorithms, proposed method use fuzzy logic components to detect two or more processes that fall into the state of deadlock, and thus do not make the algorithm cumbersome, which allows it to be used in modern operating systems.

Corresponding Author:

Andrii Nicheporuk,
Department of Computer Engineering and System Programming,
Khmelnytskyi National University,
Khmelnytskyi, Ukraine.
Email: andrey.nicheporuk@gmail.com

1. INTRODUCTION

The current state of computer technology requires maintaining a high level of parallelism while ensuring the simultaneous use of as many system components and resources as possible. During the operation of suches multi-tasking system quite often there is a situation of blocking processes that are performed on them [1].

The partial case of blocking processes is the possibility of their mutual blocking. Mutual blocking (or deadlock) is a state in multi-tasking environment or database management system, in which several processes are in a state of unexpected waiting for the resources employed by these processes. The emergence of processes deadlock leads to an increase in the time of their execution (can grow to infinity) and to the inefficient use of system resources (empty waiting cycles).

Today the problem of deadlock is still common actually. According to Sun's bug database at http://bugs.sun.com/ near two thousand bug reports out of 215 000 contain the keyword "deadlock". Moreover most of the modern operating systems including Windows and the UNIX family ignore deadlock [2].

To solving the problem of deadlock, different approaches have been developed to handle deadlocks in multi-tasking systems, including both dynamic [3]–[5] and static analyses [6]–[8]. Although a number of methods and tools have been developed to identify and eliminate the processes deadlock in computer systems, they are not always appropriate to use for solving the problem of deadlock, since some of them are only theoretical and can not be implemented in modern operating systems [9], [10]. The other part in the implementation becomes rather cumbersome and resource-intensive. Therefore, developers of modern operation systems, as well as developers of modern database management system, do not include known algorithms to avoid deadlock of processes.

Under conditions of a small number of processes in the system, the absence of such means was permissible. However, the rapid development of hardware, the growth of volume and content of software,

which solves large-scale and responsible tasks, should not allow the emergence of deadlock, which in turn requires the development of new approaches to solving this problem.

2. PREDICTION OF ENTERING PROCESSES INTO DEADLOCK STATE

Process is a system of actions that implements an internal function in a computer system and is designed in such a way that the control program of the computer system can redistribute the resources of this system in order to provide multitasking [2].

Let us denote the set of executable processes as $A=\{a_i\}_{i=1}^{y}$, where y – amount of processes. A resource of the computer system is component of the computer system, that which can be allocated to the process of data processing for a certain amount of time. Let us denote the set of available resources of the computer system (CS) as $RE=\{re_j\}_{j=1}^{x}$, where x – number of resources types.

To the resources of the CS we will take present memory, processors, input / output devices, mechanisms for mutual exclusion (semaphore, mutex, etc), and also data, that needed for work of processes (files in memory and on disk drives, results of calculations of other processes, etc). Each process from the moment of creation to the moment of termination passes through a number of states as shown in Figure 1.

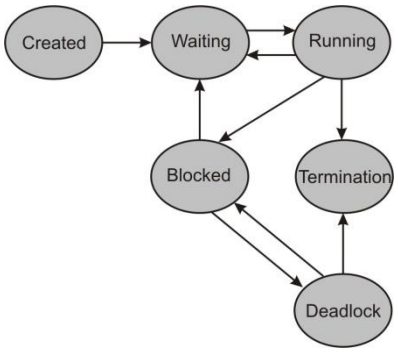


Figure 1. State diagram for process, which includes deadlock stage

Under the signature of the process, we will understand the set of its characteristics, which uniquely identifies the state of the process in the CS at a certain time t:

$$a_i(t)\rightarrow(a_{i_1}^t,a_{i_2}^t,...,a_{i_z}^t) \quad (1)$$

where $a_i(t)\in A$ – current process, $a_{i_1}^t,a_{i_2}^t,...,a_{i_z}^t$ are the process characteristics at the current time (the parameters and resources that use the process at the moment).

To the characteristics of the process that generates a signature, let's take the following: the process ID, the parent process identifier, the user ID that belongs to the process, the process priority, the process quotas (the amount of memory and processor time of the available processes) and the list of resources received. As the signature of the process includes its unique characteristics, then at the same time in the system there are not two absolutely identical signatures.

The life cycle of the process can be presented as a sequence of states through which the process passes through. The transition from state to state occurs due to a change in certain parameters that characterize the process. Changing the parameters of the process occurs for a number of reasons: opearatiob system (OS) actions, the actions of other processes, the execution of its own program code. The state of each individual process will affect the state of the CS in general.

Let denote the state of the process through w, reason for changing parameters through r and the transition from state to state through $w_i \xrightarrow{changing\ parameters\ through\ r_j} w_{i+1}$. Then the life cycle of the process can be presented (2):

$$a_i: w_0 \xrightarrow{r_j} w_1 \xrightarrow{r_j} ... \xrightarrow{r_j} w_{k-1} \xrightarrow{r_j} w_k \quad (2)$$

where $w_0 \in W$ – initial state of the process (state "created"), $w_k \in W$ – completion execution of the process (state "terminated"), W – set of program states of the process (Figure 1), $r_j \in R$ – a set of possible reasons for changing the process parameters.

Taking into account statements (1) and (2), the life cycle of each process can be presented as:

$$a_i : (a_{i_1}^{t_0}, a_{i_2}^{t_0}, ..., a_{i_z}^{t_0}) \xrightarrow{r_j} (a_{i_1}^{t_1}, a_{i_2}^{t_1}, ..., a_{i_z}^{t_1}) \xrightarrow{r_j} ... \xrightarrow{r_j} (a_{i_1}^{t_k}, a_{i_2}^{t_k}, ..., a_{i_z}^{t_k}) \quad (3)$$

In multitasking systems interacting processes can get into a state of deadlock at a certain point in time. According to the life cycle of processes, before entering the state of deadlocking, processes are in other states, such us created, waiting, running and blocked is shown in Figure 1. To the state of the deadlock, processes get, as a rule, from the blocked state. Consequently, among a set of processes, it is possible to allocate a subset of processes, which can at the next time get into the state of deadlock. Before entering the state of deadlock, the process will be in a certain "boundary" state, after which the probability of transition to the state of deadlock will be high. Processes deadlock leads to partial or complete loss of functionality of the CS. Therefore, let us consider the state of deadlock processes in the non-working state of the CS, and other states of processes - the working state of the CS.

We will attribute to the working state the following process states: created, running, terminated and waiting. To the "boundary" state refer the state "blocked".

Let's present the process' way that enters the state of deadlock, in the form of the following scheme as shown in Figure 2.

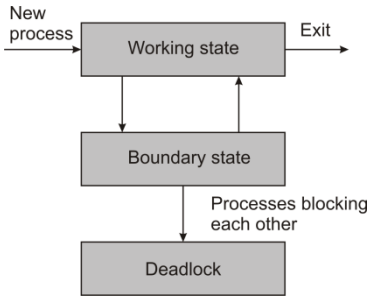


Figure 2. The scheme of processes transition to the state of deadlock

As can be seen from the scheme, the occurrence of processes deadlock is possible only for a part of the processes that are in the boundary state. In the transition of processes to the boundary state there is a change in their parameters.

Consider the life cycle of the process. At the time of creation (the state created) the process is in working state, it is provided with part of the system resources. Denote this state of the process as $s_{work}(t)$. At some point in time, the process for further execution requires additional system resources that are currently unavailable. There is a transition of the process to the state of blocked, that is, the process falls into the boundary state. Denote this state of the process as $s_{bound}(t)$ and the transition to this state, as $s_{work}(t) \xrightarrow{resource\ need\ re_i} s_{bound}(t)$. At satisfaction of necessities of process in resources it goes back into the working state, that is carries out the transition $s_{bound}(t) \xrightarrow{providing\ a\ resource\ re_i} s_{work}(t)$. When the necessary resource can not be obtained due to its use by another process, the process remains in the boundary state. If the second process, expects the resource occupied by the first process, then both of them fall into the deadlock state. Denote this state of the process as $s_{deadlock}(t)$ and the transition to this state, as $s_{bound}(t) \xrightarrow[capture\ the\ resource\ re_i]{waiting\ for\ resource\ re_i} s_{deadlock}(t)$.

Thus, taking into account (3), the process life cycle will have one of the following variants:

1) The process is created, it is provided with all necessary resources to complete the work, it is executed and terminated (the process is always in working state $s_{work}(t)$):

$$a_i : s_0 \xrightarrow{r_j} s_1 \xrightarrow{r_j} s_k \tag{4}$$

where $s_0 \in s_{work}$, $s_1 \in s_{work}$, $s_k \in s_{work}$.

2) The process is created, it has been provided with some of the resources required to complete the work, it needs additional resources, after a while he receives them, runs and terminates (the process is initially in working order $s_{work}(t)$, then transfers to $s_{work}(t) \xrightarrow{resource\ need\ re_i} s_{bound}(t)$ the boundary state $s_{bound}(t)$ and then $s_{bound}(t) \xrightarrow{providing\ a\ resource\ re_i} s_{work}(t)$ again to the working state $s_{work}(t)$:

$$a_i : s_0 \xrightarrow{r_j} ... \xrightarrow{r_j} s_l \xrightarrow{r_j} ... \xrightarrow{r_j} s_k \tag{5}$$

where $s_0 \in s_{work}$, $s_l \in s_{bound}$, $s_k \in s_{work}$.

3) The process is created, it has been provided with some of the resources required to complete the work, it needs additional resources, that holds another process which in turn requires the resources of the first process (the process is initially in working order $s_{work}(t)$, then transfers to $s_{work}(t) \xrightarrow{resource\ need\ re_i} s_{bound}(t)$ the boundary state $s_{bound}(t)$, from which the transition occurs $s_{bound}(t) \xrightarrow[capture\ the\ resource\ re_i]{waiting\ for\ resource\ re_i} s_{deadlock}(t)$ to the deadlock state).

$$\left\{ \begin{array}{l} a_i : s_0 \xrightarrow{r_j} ... \xrightarrow{r_j} s_l \xrightarrow{r_j} s_x \\ a_m : s_0 \xrightarrow{r_j} ... \xrightarrow{r_j} s_n \xrightarrow{r_j} s_x \end{array} \right\} \tag{6}$$

where $s_0 \in s_{work}$, $s_l \in s_{bound}$, $s_n \in s_{bound}$, $s_x \in s_{deadlock}$.

From the above considered possible variants of the behavior of processes critical for the CS is the last, in which processes fall into the deadlock state.

As can be seen from Figure 2, the forecasting of the process' state includes two stages: definition a set of processes that can fall into the state of deadlock and allocation a processes' set of a group of processes that fall into the deadlock state. Thus, the method for prediction of entering processes into the deadlock state will contain two parts (algorithms).

According to [11], the model for forecasting the processes' state includes the following values:

$$M = \langle A, S, D, P, R \rangle \tag{7}$$

where A is the set of processes' signature performed in the CS at the moment; S – ordered sequence of characteristics of the computer system (total amount of RAM, external memory, the amount of free memory at the moment, the number of peripheral devices); D – subset of processes' signatures that are in a state similar to the deadlock state (in the boundary state); P – a set of rules on the basis of which a group of processes that fall into the deadlock state is determined; R – probabilities vector of transition to the deadlock state of processes from a subset D.

Algorithm 1. Detection of potential processes that may fall into the deadlock (detection of processes in the boundary state):

1) If $a_k \in A$ needs a resource $r_i \in R$, then go to 2;
2) If $a_k \in A$ is in a state $s_{work}(t)$, then go to 3; else go to 4;
3) Perform for $a_k \in A$ transition from working state to boundary $s_{work}(t) \xrightarrow{resource\ need\ re_i} s_{bound}(t)$ and include $a_k \in A$ to $D \subset A$. Go to 4.
4) If $a_k \in A$ obtain the resource $re_i \in RE$ then go to 5; else go to 6;

5) Complete for $a_k \in A$ transition from the boundary state to the working $s_{bound}(t) \xrightarrow{providing\ a\ resource\ \ re_i} s_{work}(t)$ and exclude $a_k \in A$ from $D \subset A$. Go to 4;
6) If the whole set A is checked then go to 7; else go to 1;
7) End of the algorithm.

To determine the processes that fall into the deadlock state, the fuzzy inference system (FIS) of forecasting the state of processes was used [12].

The subsystem of fuzzification is intended to determine the degree of belonging of the input values $x_g \in X$, $X = D \cup S$, $g = \overline{1,h}$ to fuzzy sets, which are linguistic variables from the corresponding linguistic scale $T_{x_g} = \{T_{x_g}^1, T_{x_g}^2, ..., T_{x_g}^{m_{x_g}}\}$, where m_{x_g} – the number of linguistic variables in the g-th scale. The need for fuzzification is due to the fact that FIS uses linguistic rules.

The rule base containing linguistic rules is the basis of the mechanism of fuzzy conclusion. The mechanism of fuzzy conclusion carries out the mapping of input fuzzy sets T_{x_g} by using each rule, in the output T_y from a set of output linguistic variables $T_y = \{T_y^1, T_y^2, ..., T_y^{m_y}\}$. Each rule in the rule base $P = \{P_j\}$, $j = \overline{1,n}$ presents as follows:

$$P_j = if\ x_1 \in T_{x_1} \vee x_2 \in T_{x_2} \vee ... \vee x_h \in T_{x_h}, then\ y_j \in T_y \tag{8}$$

Output fuzzy sets y_i of each rule are combined into one fuzzy set of conclusions $\tilde{y}$.

After that, the subsystem of defuzzification maps a fuzzy set of conclusions $\tilde{y}$ in a clear number $\bar{y}$, which will be the result of FIS for given input values x_g.

Algorithm 2. Processes detection that falling into the state of deadlock:

1. Normalization of the proces' characteristics of the CS using the following formula:

$$P_n = 1 - \frac{P_d + P_m}{P_d \cdot P_m}, P_d \neq 0 \tag{9}$$

where, P_n, P_d, P_m – normalized, current and maximal value of process' characteristic, respectively;

2. For each 1 $x_g \in X$, $X = D \cup S$, $g = \overline{1,h}$ d determination the degree of belonging to the fuzzy sets of input (degrees of truth $\mu_g^i(x_g)$).

3. For each rule p, the the firing strength a_j is thus computed, as:

$$a_j = \min(\mu_1^j(x_1), \mu_2^j(x_2), ..., \mu_h^j(x_h)) \tag{10}$$

4. Based on the firing strength a_j s define an output fuzzy set with a truncated membership function $\ddot{\mu}^j$:

$$\ddot{\mu}^j(y) = \min(a_j, \mu^j(y)) \tag{11}$$

5. Combine the outputs obtained for each rule in step 4 (obtain conclusion) into a single fuzzy set, using a fuzzy aggregation operator:

$$\tilde{y} = \max(\mu^j(y)),\ j = \overline{1,r} \tag{12}$$

6. Maps a fuzzy set $\tilde{y}$ to a crisp set using centroid method:

$$\bar{y} = \frac{\int_{C_1}^{C_t} x \cdot f_{\tilde{y}}(x)dx}{\int_{C_1}^{C_t} f_{\tilde{y}}(x)dx} \quad (13)$$

7. Repeat the steps 1-6 k times for all processes in the boundary state.
8. End of the algorithm.

3. TIME COMPLEXITY OF PROPOSED METHOD

For evaluate time complexity of the method for prediction of entering processes into the deadlock state, the web server Apache2 with PHP 5, MySQL sql server and virtual environment Qemu [13] were used. To verify the proposed method the simple php script was launced on server, which in the case of parallel execution leads to the occurrence of deadlock. An example of a php script for testing is given below:

```
$sql = "SELECT COUNT(*) FROM t "; $x = mysql_query($sql);
$r = mysql_fetch_row($x);$max_id = $r[0];
$id1 = rand(0,$max_id); do { $id2 = rand(0,$max_id); } while ($id1==$id2);
mysql_query("START TRANSACTION;");
$sql1 = "select a from t where id = $id1 for update"; mysql_query($sql1);
// Choosing a tuple for calculation
usleep(100); // Data processing simulation
$sql2 = "select b from t where id = $id2 for update"; mysql_query($sql2);
mysql_query("COMMIT;"))
```

In the presented code, business logic is deleted, but the sequence of queries is completely saved, which, when executed in parallel, leads to the appearance of deadlock.

Figure 3 shows the level of deadlock in the computer system. The first curve shows the level of deadlocking when using standard MySQL tools to detect blocked transactions. The second curve shows the level of deadlocking that arose when using the proposed method for prediction of entering processes into the deadlock state.

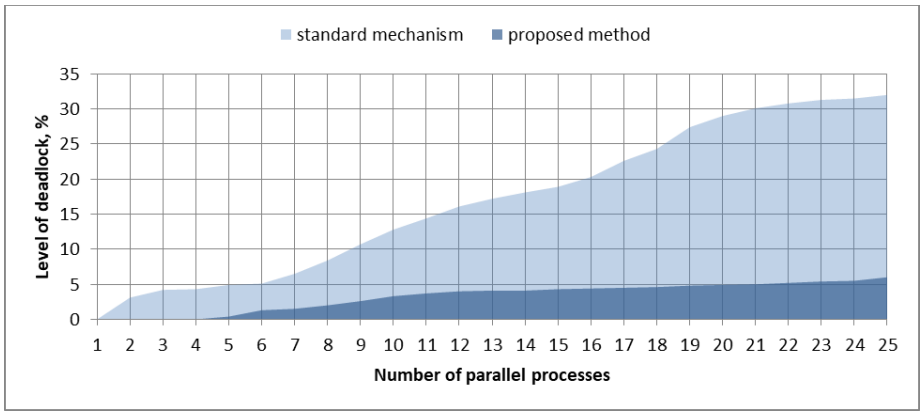


Figure 3. Level of deadlock in computer system

The time of execution of processes using different mechanisms for solving deadlocking is shown in Figure 4. The first curve shows the average process execution time using a standard mechanism for solving deadlocking. The second curve - the average time of execution of processes using the proposed method for prediction of entering processes into the deadlock state. The third - the average time for execution of processes without of deadlocking.

In the case of the use of the standard mechanism for detecting deadlocking, a significant increase in execution time is observed even with a small number of parallel processes. During this period, there is no significant load on the processor (no more than 20%), since processes await access to blocked resources for most of the time.

In the case when there are no deadlocking processes, the time of their execution increases slightly (by 3% for the number of parallel processes 25 and the level of loading of the processor is not more than 20%).

The use of the proposed method for prediction of entering processes into the deadlock stateshows a much slower growth of the average process execution time, under the same conditions. The reduction in the execution time of processes in the execution of a single stream is due to the lack of delays associated with the expectation of the release of the resource.

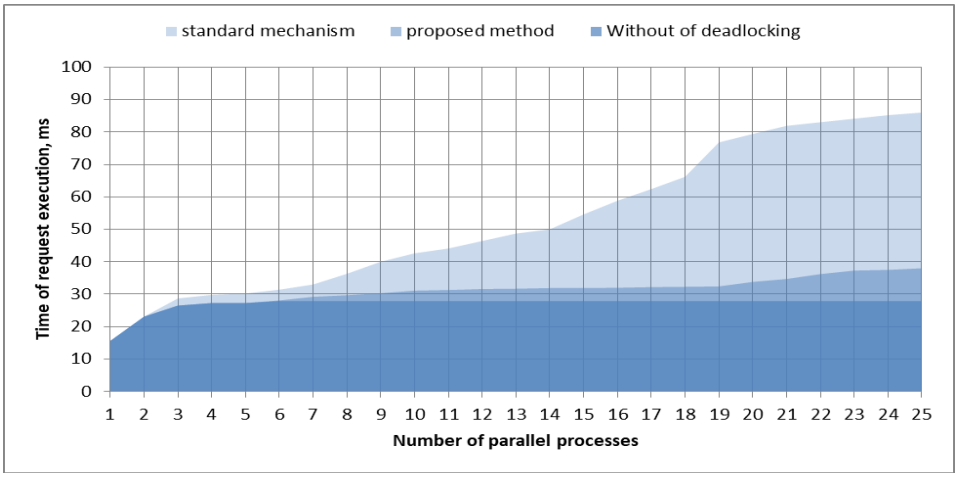


Figure 4. Dependence of the processing time of requests from the number of parallel processes

Additional, in the course of the study, a comparison was made between the execution time of the transaction phases at the maximum load of the system and the different operating modes, as shown in Figure 5.

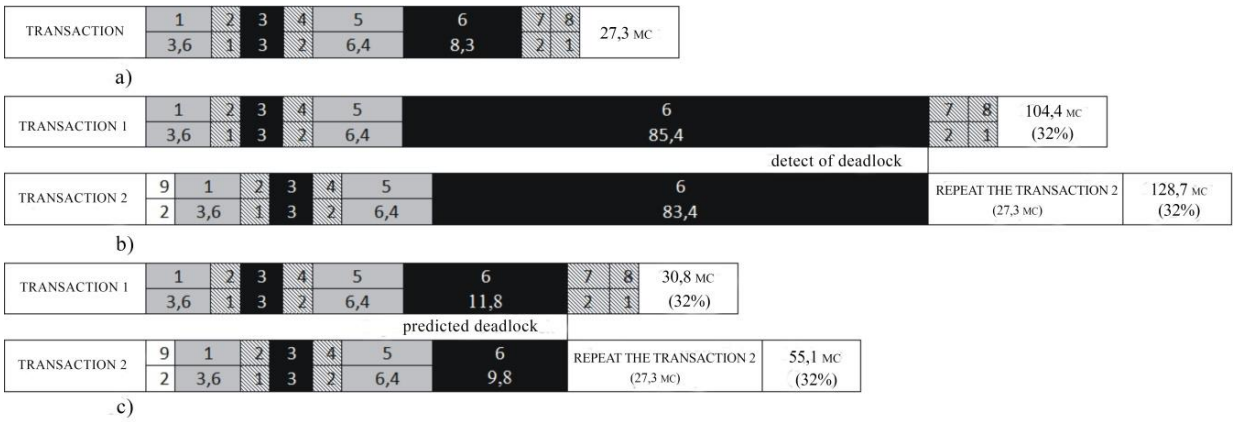


Figure 5.Timelines for completion of transaction phases

a) without deadlocking; b) deadlock with standard detection; c) deadlock with prediction. (1 - Preparation; 2 - Running a transaction; 3 - Expecting the release of the tuple id1; 4 - Running the 1st SELECT; 5 - Processing the data; 6 - Expecting the release of the tuple id2; 7 - Running the 2nd SELECT; 8 - Completing the transaction ; 9 - Delay from startup.)

In Figure 5.a presents a timeline for the execution of the phases of the transaction, provided that when it is executed there there was no deadlocking. The average transaction time was 27.3 ms, of which 11.3 ms (41%) took the expectation of blocked resources. In the course of the study, 36% of these transactions were detected.

In Figure 5.b presents a timeline for deadlocking two transactions with a standard mechanism for solving them. *Transaction 1* at first access (4) to the database table selects and blocks the tuple with *id1* key. *Transaction 2* begins to run 2 ms later than *Transaction 1* and at first access (4) selects and blocks the entry with *id2* key. After processing the received data (5), *transaction 1* refers to the tuple with *id2*. Since it is locked, the transaction passes to waiting for the release of the resource (6). In turn, *transaction 2* attempt to access to the tuple with *id1*. It also had locked and *transaction 2* enters standby mode for the release of the resource (6). The processes fall into the deadlock and can not independently exit the cycle of endless expectations. To decid deadlock the computer system periodically analyzing the graph of processes and resources. Since the task is algorithmically complex, it is executed at intervals of time in 150-200ms. After detecting deadlock, one of the processes that later logged in is forcibly terminated and restarted. Another of the blocked processes continues to run. Large time intervals between repeated analyzes of the graph lead to significant delays in deadlock detecting. The average process time that had been performed repeatedly is 128.7 ms, the process that continue running after the deadlocking- 104.4 ms. During the study of such transactions, it turned out to be 32%, as the processes in the deadlock fall in pairs.

In Figure 5. a timeline for deadlocking of two transactions with prediction of deadblocking is presented. By the time of the request in *transaction 2* tuple with the key *id2*, timelines are identical. At the moment of this request (6), *transaction 2* falls into the boundary state and for it the prediction is carried out. The prediction time is about 10ms. As a result of the prediction, the probability of deadblocking of processes is determined, after which one of them is selected. This process is terminated and is restarted. The average execution time of the process that was performed repeatedly is 55.1 ms. The process that continued after deadlock- 30.8 ms. As a result of the study, 32% of these transactions were detected. Thus, from the obtained studies, the average transaction time with the standard mechanism for detecting deadlocking was 84,4 ms, and with prediction of interlocking 37,3 ms, which is 2.3 times less.

4. CONCLUSION

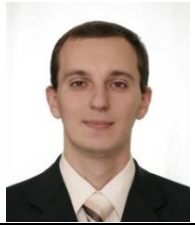
In this paper, we analyzed the problem of processes' deadlock in the multi-tasking system. On the basis of the study of the life cycle of the process, the boundary state of the process was distinguished, which will precede the deadlock state. Proposed the method for prediction of entering processes into the deadlock state, which consists of two algorithms: algorithm of detection of potential processes that may fall into the deadlock and algorithm of processes detection that falling into the state of deadlock. An evaluation of time complexity of proposed method is conducted. Application of the method for prediction of entering processes into the deadlock state has reduced the time of execution of processes in 2,3 times, that allows more efficient use of computer system resources. Unlike the known methods and algorithms, proposed method use fuzzy logic components to detect two or more processes that fall into the state of deadlock, and thus do not make the algorithm cumbersome, which allows it to be used in modern operating systems.

REFERENCES

[1] M. Samak, M.K. Ramanathan, "Trace driven dynamic deadlock detection and reproduction", *In Proceedings of the 19th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp. 29-42, 2014.
[2] A.S. Tanenbaum, H. Bos, "Modern Operating Systems", Pearson PLS. p. 1136, 2014.
[3] P. Joshi, M. Naik, K. Sen, D. Gay, "An effective dynamic analysis for detecting generalized deadlocks", *In Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 327-336, 2010.
[4] P. Joshi, C.S. Park, K. Sen, M. Naik, "A randomized dynamic program analysis technique for detecting real deadlocks", *In Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 110-120, 2009.
[5] Y. Cai, W.K. Chan, "MagicFuzzer: Scalable deadlock detection for large-scale applications", *In Proceedings of 34th International Conference on Software Engineering*, 2012.
[6] A. Soleimany, Z. Giyahi, "An Efficient Distributed Deadlock Detection and Prevention Algorithm by Daemons", *International Journal of Computer Science and Network Security*, vol. 12, no. 4, pp. 150-155, 2012.
[7] S. Hu, et al., "Tagger: Practical PFC Deadlock Prevention in Data Center Networks", *In Proceedings of the 13th International Conference on emerging Networking Experiments and Technologies*, pp. 451-463, 2017.
[8] S. Hu, et al., "Deadlocks in datacenter networks: Why do they form, and how to avoid them", *In Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pp. 92–98, 2016.

[9] L. Boussaid, S. Zouaoui, M. Abdellatif, "Priority based round robin (PBRR) CPU scheduling algorithm", *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 1, pp. 190-202, 2019.
[10] N. Srilatha, M. Sravani, Y. Divya, "Optimal Round Robin CPU Scheduling Algorithm Using Manhattan Distance", *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 6, pp. 3664-3668, 2017.
[11] F. Schmidt, M. Niepert, F. Huici, "Representation Learning for Resource Usage Prediction", *In Proceedings of 18th SysML Conference*, pp. 1-3, 2018.
[12] A. Moffaq, et al., "Fuzzy Logic based Edge Detection Method for Image Processing", *International Journal of Electrical and Computer Engineering (IJECE)*, vol.8, no.3, pp.1863-1869, 2018.
[13] O. Pomorova, O. Savenko, S. Lysenko, A. Nicheporuk, "Metamorphic Viruses Detection Technique based on the the Modified Emulators", CEUR-WS, vol. 1614, pp. 375-383, 2016.

BIOGRAPHIES OF AUTHORS

	Andrii Nicheporuk obtained his PhD degree in 2018. He has been appointed as a Senior Lecturer in Department of Computer Engineering and System Programming, Khmelnytskyi National University, Ukraine. His research interest includes detection of malware, reverse engineering, robotics and software programing.
	Yuriy Klots obtained his PhD degree in 2007. He is Assistant Proffesor in Department of Computer Engineering and System Programming, Khmelnytskyi National University. His area of research interest includes pattern recognition, software programing, diagnosing computer systems and networks.
	Oksana Yashyna obtained his PhD degree in 2015. She is Assistant Proffesor in Department of Software Engineering, Khmelnytskyi National University. Her area of research interest includes software programing, problem of deadlock processes, research and introduction of smart technologies in various spheres of human life.
	Sergiy Mostovyi received the master's degree in Computer Engineering from the Khmelnytskyi National University, in 2008. His research interest includes interaction between processes in Windows and Linux operating systems, software programing and system administration.
	Yuriy Nicheporuk received the master's degree in Computer Engineering from the Khmelnytskyi National University, in 2017. Currently hi is postgraduate student. His research interest includes software programing, cryptography and network security,