УДК 004.451

О. С. Савенко, канд. техн. наук, доц.;
С. В. Мостовий

ПРОГНОЗУВАННЯ ПОТРАПЛЯННЯ ПРОЦЕСІВ У СТАН ВЗАЄМОБЛОКУВАННЯ

Проаналізовано життєвий цикл процесу, в комп'ютерній системі виділено граничний стан, що передує стану взаємоблокування. Розроблено алгоритм прогнозування потрапляння процесів у стан взаємоблокування.

Вступ та постановка задачі

Вагома частка взаємоблокувань процесів припадає на взаємоблокування процесів, що виконуються в комп'ютерних системах (КС).

Під комп'ютерною системою будемо розуміти сукупність програмно-апартних засобів, призначених для розв'язання поставленої задачі, а саме: персональні комп'ютери (ПК) та відповідне програмне забезпечення (як системне, так і прикладне).

В результаті дослідження відомих методів та алгоритмів уникнення взаємоблокувань процесів в операційних системах (ОС) [1, 2] виявлено, що вони не в повному обсязі розв'язують поставлену задачу. Не всі алгоритми придатні до реалізації у сучасних операційних системах [3, 4], оскільки мають низку недоліків (блокування роботи ОС, наявність циклів активного очікування, складність програмної реалізації для багатьох процесів, необхідність використання спеціалізованої команди процесора) і є складними для реалізації. Тому більшість сучасних ОС не містять ефективних засобів для вирішення проблеми взаємоблокування процесів [5].

За умови виконання в системі невеликої кількості процесів відсутність таких засобів була допустима. Проте масштабність задач, які розв'язуються за допомогою ПК, зростає, і це вимагає розв'язання задачі уникнення взаємоблокування процесів. Тому задача розробки нових методів та засобів, які б дозволили прогнозувати входження процесів у стан взаємоблокування, є актуальною.

Під час розробки нових методів та засобів розв'язання задачі взаємоблокування процесів необхідно усунути основні недоліки відомих методів та засобів. Тому виникає потреба у дослідженні умов, за яких виникає взаємоблокування процесів, протягом всього часу перебігу процесів (протягом їхнього життєвого циклу).

Життєвий цикл процесів комп'ютерної системи

Багатозадачність (англ. multitasking) — властивість операційної системи, або середовища програмування, забезпечувати можливість паралельної (або псевдопаралельної) обробки декількох процесів [6]. Дійсна багатозадачність операційної системи можлива тільки в розподілених обчислювальних системах.

Процес — це система дій, що реалізує певну функцію в обчислювальній системі й оформлена так, що керуюча програма обчислювальної системи може перерозподіляти ресурси цієї системи з метою забезпечення багатозадачності [6]. Позначимо множину виконуваних процесів як $A = \{a_i\}_{i=1}^{y}$, де y — кількість процесів.

Ресурс обчислювальної системи — засіб обчислювальної системи, що може бути виділений процесу обробки даних на певний інтервал часу [6]. Позначимо множину наявних ресурсів ОС як $RE = \{re_j\}_{j=1}^{x}$, де x — кількість видів ресурсів.

До ресурсів ОС віднесемо наявну пам'ять, процесори, пристрої вводу/виводу, а також дані, необхідні для роботи процесів (файли в пам'яті та на зовнішніх носіях, результати обчислень інших процесів).

Кожен процес від моменту створення до моменту завершення проходить через низку станів (рис. 1).

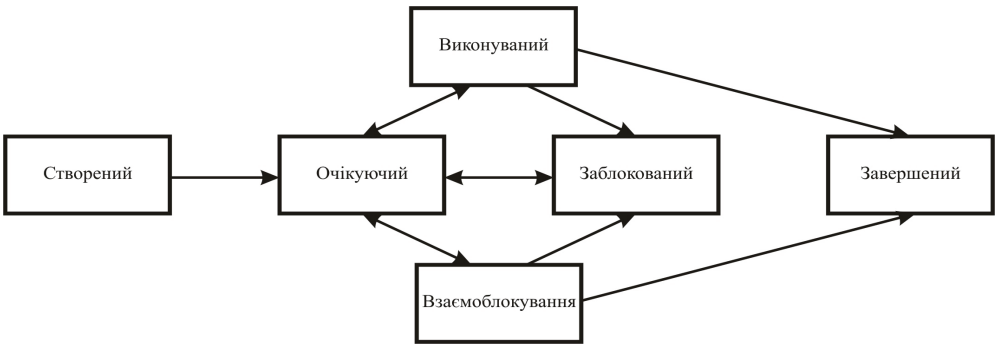

Рис. 1. Діаграма станів процесу, що включає стан взаємоблокування

Під сигнатурою процесу будемо розуміти сукупність його характеристик, яка однозначно ідентифікує стан процесу в ОС в певний момент часу t:

$$a_i(t) \to \left(a_{i_1}^t, a_{i_2}^t, ..., a_{i_z}^t \right), \tag{1}$$

де $a_i(t) \in A$ — поточний процес; $a_{i_1}^t, \ldots, a_{i_z}^t$ — характеристики процесу в поточний момент часу (параметри та ресурси, які використовуються у процесі в цей момент).

До характеристик процесу, що формують сигнатуру, віднесемо такі: ідентифікатор процесу, ідентифікатор батьківського процесу, ідентифікатор користувача, якому належить процес, пріоритет процесу, квоти процесу (кількість пам'яті і процесорний час доступні процесу), дескриптори відкритих процесом файлів [6].

Оскільки до складу сигнатури процесу входять унікальні характеристики, то одночасно в системі не існує двох абсолютно однакових сигнатур.

Отже, життєвий цикл процесу можна подати у вигляді послідовності станів, через які проходить цей процес. Перехід зі стану в стан відбувається через зміну певних параметрів, якими характеризується процес. Зміна параметрів процесу відбувається з низки причин: дії ОС, дії інших процесів, виконання власного програмного коду. Стан кожного окремого процесу буде впливати на стан КС в цілому.

Позначимо стан процесу через w, причину зміни параметрів через r, а перехід із стану в стан через $w_i \xrightarrow{\text{зміна параметрів з причини } r_j} w_{i+1}$. Тоді вираз для життєвого циклу процесу буде мати вигляд (2)

$$a_i : w_0 \xrightarrow{r_j} w_1 \xrightarrow{r_j} \ldots \xrightarrow{r_j} w_{k-1} \xrightarrow{r_j} w_k, \tag{2}$$

де $w_0 \in W$ — початковий стан процесу (стан «створений»); $w_k \in W$ — кінцевий стан процесу (стан «завершений»); W — множина програмних станів процесу (див. рис. 1); $r_j \in R$ — множина можливих причин зміни параметрів процесу $\left(j = 1, 2, 3 \ldots \right)$.

Враховуючи визначення сигнатури та (1) і (2), життєвий цикл кожного процесу можна подати у вигляді:

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

У стан взаємоблокування можуть потрапляти процеси, що взаємодіють між собою у багатозадачних ОС в певний момент часу. До потрапляння у стан взаємоблокування процеси протягом свого життєвого циклу знаходяться в інших станах, а саме стан «створений», стан «очікуючий», стан «виконуваний», стан «заблокований», стан «завершений» (див. рис. 1). У стан взаємоблокування процеси потрапляють, як правило, із стану «заблокований» або стану «очікуючий». Отже, у певний момент часу серед множини процесів можна виділити підмножину процесів, які можуть в наступний момент часу потрапити до стану взаємоблокування. Перед входженням у стан взаємоблокування процес буде знаходитись у певному «граничному» стані [7], після якого ймовірність переходу у стан взаємоблокування буде високою. Взаємоблокування процесів призводить до часткової або повної втрати функційної здатності ОС. Тому вважатимемо стан взаємоблокування процесів неробочим станом ОС, а інші стани процесів — робочим станом ОС.

Віднесемо до робочого стану такі стани процесу: стан «створений», стан «виконуваний», стан

«завершений». До «граничного» стану віднесемо такі стани процесу: стан «заблокований», стан «очікуючий».

Покажемо шлях, яким процес потрапляє у стан взаємоблокування, у вигляді такої схеми (рис. 2).

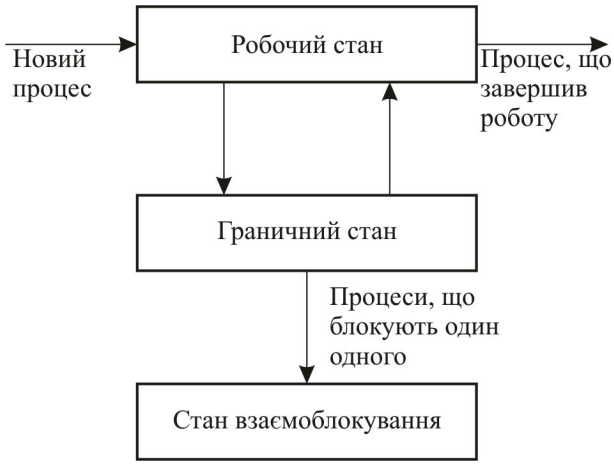

Рис. 2. Схема переходу процесів у стан взаємоблокування

Як видно зі схеми (див. рис. 2), виникнення взаємоблокування процесів можливе лише для частини процесів, що знаходяться у граничному стані. Під час переходу процесів до граничного стану відбувається зміна їхніх параметрів.

Розглянемо життєвий цикл процесу. В момент створення (стан «створений») процес знаходиться у робочому стані, йому надана частина ресурсів системи. Позначимо цей стан процесу як $s_{\text{роб}}(t)$. В певний момент часу процесу для подальшого виконання необхідний додатковий ресурс системи, який на цей час є недоступний. Відбувається перехід процесу до стану «заблокований», тобто процес потрапляє у граничний стан. Позначимо цей стан процесу як $s_{\text{гран}}(t)$, а перехід до цього стану як $s_{\text{роб}}(t) \xrightarrow{\text{потреба ресурсу } re_i} s_{\text{гран}}(t)$. У разі задоволення потреб процесу у ресурсах він повертається у робочий стан, тобто здійснює перехід $s_{\text{гран}}(t) \xrightarrow{\text{надання ресурсу } re_i} s_{\text{роб}}(t)$. Коли необхідний ресурс отримати неможливо з причини його використання іншим процесом, то процес залишається у граничному стані. Якщо ж другий процес в свою чергу очікує ресурс, що зайнятий першим процесом, то вони обидва потрапляють у стан взаємоблокування. Позначимо цей стан процесу як $s_{\text{взаємоблокування}}(t)$, а перехід до цього стану — як $s_{\text{гран}}(t) \xrightarrow[\text{утримування ресурсу } re_j]{\text{очікування ресурсу } re_i} s_{\text{взаємоблокування}}(t)$.

Таким чином, враховуючи (3), життєвий цикл процесу буде мати один із таких виглядів:

1. Процес створений, йому надано всі необхідні для завершення роботи ресурси, він виконується і завершується (процес весь час знаходиться в робочому стані $s_{\text{роб}}(t)$), тобто

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

де $s_0 \in s_{\text{роб}}$; $s_1 \in s_{\text{роб}}$; $s_k \in s_{\text{роб}}$.

2. Процес створений, йому надано частину ресурсів, необхідних для завершення роботи, він потребує додаткових ресурсів, через деякий час отримує їх, виконується і завершується (процес спочатку знаходиться в робочому стані $s_{\text{роб}}(t)$, потім переходить $s_{\text{роб}}(t) \xrightarrow{\text{потреба ресурсу } re_i} s_{\text{гран}}(t)$ в граничний стан $s_{\text{гран}}(t)$, потім $s_{\text{гран}}(t) \xrightarrow{\text{надання ресурсу } re_i} s_{\text{роб}}(t)$ знову в робочий стан $s_{\text{роб}}(t)$), тобто

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

де $s_0 \in s_{\text{роб}}$; $s_l \in s_{\text{гран}}$; $s_k \in s_{\text{роб}}$.

3. Процес створений, йому надано частину ресурсів, необхідних для завершення роботи, він потребує додаткових ресурсів, які утримує інший процес, який в свою чергу потребує ресурсів першого процесу (процес спочатку знаходиться в робочому стані $s_{\text{роб}}(t)$, потім переходить $s_{\text{роб}}(t) \xrightarrow{\text{потреба ресурсу } re_i} s_{\text{гран}}(t)$ в граничний стан $s_{\text{гран}}(t)$, із якого відбувається перехід $s_{\text{гран}}(t) \xrightarrow[\text{утримування ресурсу } re_j]{\text{очікування ресурсу } re_i} s_{\text{взаємоблокування}}(t)$ до стану взаємоблокування).

$$\begin{cases} a_i : s_0 \xrightarrow{r_j} \ldots \xrightarrow{r_j} s_l \xrightarrow{r_j} s_x; \\ a_m : s_0 \xrightarrow{r_j} \ldots \xrightarrow{r_j} s_n \xrightarrow{r_j} s_x, \end{cases} \tag{6}$$

де $s_0 \in s_{\text{роб}}$; $s_l \in s_{\text{гран}}$; $s_n \in s_{\text{гран}}$; $s_x \in s_{\text{взаємоблокування}}$.

Із розглянутих вище можливих варіантів поведінки процесів критичним для ОС є останній, за якого процеси потрапляють у стан взаємоблокування.

Алгоритм прогнозування потрапляння процесів у стан взаємоблокування

Як видно із рис. 2, прогнозування стану процесу включає два етапи: визначення множини процесів, що можуть потрапити у стан взаємоблокування; виділення із множини потенційних процесів групи процесів, що потраплять у стан взаємоблокування. Таким чином, алгоритм прогнозування стану процесу буде містити дві частини.

Згідно з [8] до моделі прогнозування стану процесів входять такі величини:

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

де A — множина сигнатур процесів, що виконуються на ПК у певний момент; S — впорядкована послідовність характеристик комп'ютерної системи (загальний обсяг оперативної пам'яті, зовнішньої пам'яті, обсяг вільної в цей момент пам'яті, кількість периферійних пристроїв); D — підмножина сигнатур процесів, що знаходяться у стані, наближеному до стану взаємоблокування (у граничному стані); P — множина правил, на основі яких визначається група процесів, що потраплять у стан взаємоблокування; R — вектор ймовірностей переходу у стан взаємоблокування процесів із підмножини D.

Алгоритм 1. Виявлення потенційних процесів, що можуть потрапити у стан взаємоблокування (виявлення процесів, що знаходяться у граничному стані).

1. Якщо $a_k \in A$ потребує ресурс $r_i \in R$, то перехід до 2.
2. Якщо $a_k \in A$ знаходиться в стані $s_{\text{роб}}(t)$, то перехід до 3, інакше — перехід до 4.
3. Виконати для $a_k \in A$ перехід із робочого стану до граничного $s_{\text{роб}}(t) \xrightarrow{\text{потреба ресурсу } re_i} s_{\text{гран}}(t)$ та включити $a_k \in A$ до $D \subset A$. Перехід до 4.
4. Якщо $a_k \in A$ надано у використання ресурс $re_i \in RE$, то перехід до 5, інакше — перехід до 6.
5. Виконати для $a_k \in A$ перехід із граничного стану до робочого $s_{\text{гран}}(t) \xrightarrow{\text{надання ресурсу } re_i} s_{\text{роб}}(t)$ та виключити $a_k \in A$ із $D \subset A$. Перехід до 6.
6. Якщо перевірено всю множину А, то перехід до 7, інакше — перехід до 1.
7. Кінець алгоритму.

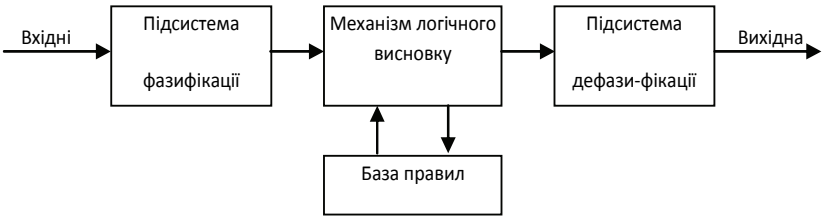

Рис. 3. Загальна схема СНЛВ

Для визначення процесів, що потраплять у стан взаємоблокування, використовується система прогнозування стану процесів, в основі якої лежить система нечіткого логічного висновку (СНЛВ). На рис. 3 показана структура СНЛВ.

Підсистема фазифікації призначена для визначення ступеня належності вхідних значень $x_g \in X$, $X = D \cup S$, $g = \overline{1,h}$ до нечітких множин входу, що є лінгвістичними змінними з відповідної лінгвістичної шкали $T_{x_g} = \left\{ T_{x_g}^1, T_{x_g}^2, ..., T_{x_g}^{m_{x_g}} \right\}$, де m_{x_g} — кількість лінгвістичних змінних у шкалі для g-го входу. Необхідність у фазифікації зумовлена тим, що у СНЛВ використовуються лінгвістичні правила.

База правил, що містить лінгвістичні правила, є основою механізму логічного висновку. Механізм логічного висновку здійснює відображення вхідних нечітких множин T_{x_g} за допомогою кожного правила у вихідну T_y з набору вихідних лінгвістичних змінних $T_y = \left\{ T_y^1, T_y^2, ..., T_y^{m_y} \right\}$. Прави-

ла із множини правил $P=\left\{P_j\right\}$, $j=\overline{1,n}$, що міститься в базі правил, подані у такому форматі:

$$P_j = \text{"якщо } x_1 \in T_{x_1} \text{ і } x_2 \in T_{x_2} \ldots \text{і } x_h \in T_{x_h}, \text{ то } y_j \in T_y\text{"}. \quad (8)$$

Вихідні нечіткі множини y_i кожного правила об'єднуються в одну нечітку множину висновку $\tilde{y}$. Після цього підсистема дефазифікації здійснить відображення нечіткої множини висновку $\tilde{y}$ у чітке число $\overline{y}$, яке буде результатом СНЛВ для заданих вхідних значень x_g.

Алгоритм 2. Виявлення процесів, що потраплять у стан взаємоблокування:

1. Проводимо нормування показників за такою формулою:

$$P_n = 1 - \frac{P_d + P_m}{P_d P_m}, \quad P_d \neq 0, \quad (9)$$

де P_n — черговий нормований показник; P_d — поточне значення показника в конкретній системі; P_m — максимальне значення показника в конкретній системі.

2. Для кожного $x_g \in X$, $X = D \cup S$, $g=\overline{1,h}$ визначаємо ступінь належності до нечітких множин входу (ступені істинності $\mu_g^j\left(x_g\right)$.

3. На основі ступенів істинності передумов $\mu_g^j\left(x_g\right)$ для кожного правила P_j, $j=\overline{1,n}$ розраховуємо ступінь його виконання α_j за формулою

$$\alpha_j = \min\left(\mu_1^j\left(x_1\right), \mu_2^j\left(x_2\right), \ldots, \mu_h^j\left(x_h\right)\right). \quad (10)$$

4. На основі ступеня виконання α_j для кожного правила P_j, $j=\overline{1,n}$ розраховуємо результат його виконання.

5. На основі результату виконання кожного правила P_j, $j=\overline{1,n}$ визначаємо вихідну нечітку множину з усіченою функцією належності $\ddot{\mu}^j\left(y\right)$ за формулою

$$\ddot{\mu}^j\left(y\right) = \min\left(\alpha_j, \mu^j\left(y\right)\right). \quad (11)$$

6. Вихідні нечіткі множини $\ddot{\mu}^j\left(y\right)$ згідно з (12) агрегуємо в нечітку множину висновку $\tilde{y}$, що має функцію належності (13).

$$\tilde{y} = \max\left(\mu^j\left(y\right)\right), \quad j=\overline{1,r}; \quad (12)$$

$$\mu_{\tilde{y}} = \max\left(\ddot{\mu}^1\left(y\right), \ddot{\mu}^2\left(y\right), \ldots, \ddot{\mu}^r\left(y\right)\right). \quad (13)$$

7. Приводимо до чіткості нечітку множину $\tilde{y}$ за допомогою процедури дефазифікації за центроїдним методом:

$$\overline{y} = \int_{C_1}^{C_t} x \cdot f_{\tilde{y}}\left(x\right) dx \Big/ \int_{C_1}^{C_t} f_{\tilde{y}}\left(x\right) dx. \quad (14)$$

8. Встановлюємо для R_k значення $\overline{y}$.

9. Повторюємо кроки 1—8 *k* разів.

10. Кінець алгоритму.

Ефективність алгоритму прогнозування стану процесу

Оцінка ефективності алгоритму включає як якісний, так і кількісні показники. Якісним показником є те, чи розв'язує алгоритм поставлену задачу, чи ні. Кількісними показниками є час виконання та ємність алгоритму. В цьому випадку критичними показниками є час, за який буде виконуватись прогнозування настання ситуації взаємоблокування, та задіяні при цьому ресурси системи. Отже, критерієм для оцінки ефективності алгоритму буде час.

Згідно із загальноприйнятими підходами основним показником часової ефективності є часова складність (ЧС) — порядок складності алгоритму $O(f)$.

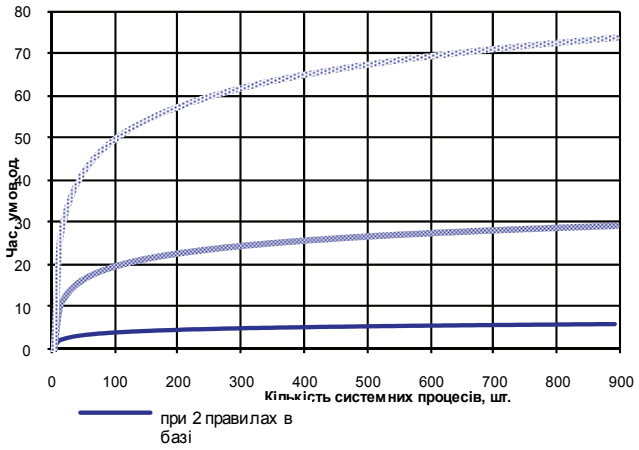

Рис. 4. Часова складність алгоритму прогнозування стану процесу

Для алгоритму 1 ЧС становить $O(k)$, де k — кількість наявних в системі процесів, оскільки k разів повторюються однакові лінійні дії.

Для алгоритму 2 ЧС становить $O\left(j \cdot \log\left(n\right)\right)$, де n — кількість процесів, що знаходяться у граничному стані, j — кількість правил, що містяться у базі правил.

На рис. 4 показана ЧС алгоритму прогнозування стану процесу для різної кількості процесів та правил. Як випливає з рис. 4, складність алгоритму зростає нелінійно зі збільшенням кількості процесів у системі, оскільки, на відміну від відомих алгоритмів розв'язання задачі взаємоблокування, розроблений алгоритм прогнозування стану процесів використовує компоненти нечіткої логіки, що робить його гнучким у використанні.

Висновки

В роботі проведено аналіз життєвого циклу процесу в комп'ютерній системі. В результаті дослідження виявлено «граничний» стан, який передує стану взаємоблокування процесів. Це дозволяє в поточний момент часу визначити множину процесів, які можуть потрапити в стан взаємоблокування в наступний момент часу. В подальшому необхідно визначити та дослідити умови та параметри КС, за яких процеси із виділеної множини потраплять у взаємоблокування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Coffman E. G. System deadlocks / E. G. Coffman, M. J. Elphick, A. Shoshani // Computing Surveys. — June 1971. — Vol. 3, No. 2. — P. 67—78.

2. Isloor S. S. The Deadlock Problem: An Overview / S. S. Isloor, T. A. Marsland // Computer. — 1980. — Vol. 13, No. 9. — P. 58—78.

3. Kaveh N. Deadlock detection in distribution object systems / Nima Kaveh, Wolfgang Emmerich // Software Engineering Notes. — September 2001. — Vol. 26, No. 5. — P 44—51.

4. Confirmation of deadlock potentials detected by runtime analysis / [Saddek Bensalem, Jean-Claude Fernandez, Klaus Havelund, Laurent Mounier] // International Symposium on Software Testing and Analysis — 2006. — P. 41—50.

5. Савенко О. С. Дослідження та аналіз блокування процесів в комп'ютерній системі / О. С. Савенко, Ю. П. Кльоц, С. В. Мостовий // Вісник ХНУ. — 2007. — Т. 1. — № 3. — С. 248—251.

6. Таненбаум Э. Операционные системы: разработка и реализация. Класика CS / Таненбаум Э, Вудхалл А. — СПб : Питер, 2006. — 576 с.

7. Поморова О. В. Теоретичні основи, метод та засоби інтелектуального діагностування комп'ютерних систем : моногр. / О. В. Поморова. — Хмельницький : ТОВ «Тріада-М», 2006. — 253 с.

8. Савенко О. С. Модель прогнозування стану процесів в комп'ютерній системі / О. С. Савенко, С. В. Мостовий // Радіоелектронні і комп'ютерні системи. — 2008. — № 5(32). — С. 109—115.

Рекомендована кафедрою захисту інформації

Стаття надійшла до редакції 2.01.2013
Рекомендована до друку 6.03.2013

Савенко Олег Станіславович — доцент; ***Мостовий Сергій Володимирович*** — асистент.
Кафедра системного програмування, Хмельницький національний університет, Хмельницький