FRI-2G.303-1-CCT1-09

COMPARATIVE ANALYSIS OF DEKKER'S AND PETERSON'S ALGORITHMS IN TIME-SHARING SYSTEMS [9]

Tzanko Golemanov, PhD
Department of Computer Systems and Technologies,
"Angel Kanchev" University of Ruse
Tel.: +359 82 888 681
E-mail: TGolemanov@uni-ruse.bg

Emilia Golemanova, PhD
Department of Computer Systems and Technologies,
"Angel Kanchev" University of Ruse
Tel.: +359 82 888 681
E-mail: EGolemanova@uni-ruse.bg

Abstract: *This report explores the "frozen quantum" issue in mutual exclusion algorithms used in time-sliced operating systems. It compares Peterson's and Dekker's approaches, particularly under preemptive scheduling. The analysis highlights a key drawback in Peterson's algorithm, where a process interrupted after raising its flag but before entering the critical section causes complete CPU quantum wastage and reduced system performance. In contrast, Dekker's method shows improved efficiency by allowing temporary flag release during contention, which enhances CPU usage. Simulations indicate that Dekker's algorithm performs notably better in high-contention environments. The findings challenge conventional assumptions about algorithm efficiency and emphasize the critical importance of context-aware synchronization mechanism selection in modern operating systems.*

Keywords: *Operating Systems, Mutual Exclusion Algorithms, Teaching Tools*

ВЪВЕДЕНИЕ

В областта на паралелното програмиране и операционните системи (ОС), проблемът с критичната секция (*Critical Section, CS*) представлява фундаментално предизвикателство. Критичната секция (Tanenbaum and Bos, 2016) (Silberschatz, 2018) е част от програмния код, където процесите осъществяват достъп до споделени ресурси, като променливи, бази данни или хардуерни устройства. Когато множество конкурентни процеси се опитват да използват тези ресурси едновременно, възниква състояние на надпревара (*race condition*), което може да доведе до непредсказуеми и неправилни резултати. Класическите алгоритми на Dekker (Dijkstra, 1965) и Peterson (Peterson, 1981), разработени преди повече от четири десетилетия, представляват крайъгълни камъни в развитието на софтуерно базирани подходи за **Взаимно изключване** (*Mutual Exclusion*) и продължават да бъдат базови в обучението на компютърните специалисти. Анализът на тези класически решения е от съществено значение за разбирането на основните принципи на синхронизацията, но поведение им в реални системи с времеделение не винаги е добре разбрано.

ВЗАИМНО ИЗКЛЮЧВАНЕ, АКТИВНО ИЗЧАКВАНЕ, ЗАМРАЗЕН КВАНТ

Съвременните операционни системи използват планиране с превключване на процесора (*Preemptive Scheduling*), за да поддържат многозадачност. При този модел, планировчикът на ОС реализира принципа на Времеделение (*time-sharing*) чрез принудително прекъсване (*preempt*) на изпълнението на даден процес след изтичане на предварително зададен времеви интервал (*time quantum*).

[9] The paper was presented on 24 October 2025 in section "Communication and Computer Technologies" with original title in Bulgarian: СРАВНИТЕЛЕН АНАЛИЗ НА АЛГОРИТМИТЕ НА DEKKER И PETERSON В СИСТЕМИ С ВРЕМЕДЕЛЕНИЕ

За разлика от други механизми за **Взаимно изключване** (като семафори), които прехвърлят чакащия процес в състояние Блокиран (*waiting/sleeping*), алгоритмите на Dekker и Peterson разчитат на **Активно Изчакване** (*Busy-waiting*). Busy-waiting е техника, при която процесът циклично изпълнява празен цикъл (*spin*), непрекъснато проверявайки стойността на споделени променливи, докато условието за влизане в CS не бъде удовлетворено.

Тази комбинация - *Busy-waiting* в среда с *Preemptive Scheduling* – води до проблема дефиниран тук като **„Замразен квант“** (*frozen quantum*). Той се изразява в неефективно „изгаряне“ на CPU-време, значителна деградация на системния пропускателен капацитет (*throughput*), увеличена латентност и големи системни разходи (*overhead*) от ненужни контекстни превключвания.

Настоящото проучване има за цел да:

- Идентифицира и анализира проблема на "замразения квант" при алгоритъма на Peterson
- Сравни поведението на алгоритмите на Dekker и Peterson в условия на интензивна конкуренция
- Предложи количествени мерки за ефективността на двата алгоритъма
- Направи изводи за практическата приложимост на класическите алгоритми в съвременни системи

Анализът се основава на теоретично изследване на алгоритмите, симулиране на тяхното поведение при различни сценарии на натоварване и оценка на производителността им по ключови метрики.

АЛГОРИТЪМЪТ НА DEKKER: КОРЕКТНОСТ И СЛОЖНОСТ

Разработен от холандския математик Theodorus Dekker през 1965 година, алгоритъмът представлява първото коректно решение на проблема за **Взаимно изключване** за два процеса (P_0 и P_1), използващо само споделена памет.

Детайли на имплементацията за процес P_0:

```
flag_0 = true;              // заявка за влизане в CS
while ( flag_1 )            // проверка на заявката на конкурента
 if  ( turn  == 1 )         // ако е ред на конкурента
  {
    flag_0 = false;         // оттегляне на заявката
      while (turn  == 1)  ; // Busy-waiting:  чака CS да се освободи от конкурента
    flag_0 = true;          // подновяване на заявката за вкизане в CS
  };

        // Critical Section

flag_0 = false;             // отмяна на заявката
turn   = 1;                 // предоставяне на ред на конкурента
```

Сложността на Dekker произтича от неговия механизъм за разрешаване на конфликти, който е проектиран да гарантира прогрес и да избегне взаимна блокировка (*deadlock*).

Когато процес P_i (например P_0) иска да влезе, той първо задава flag_0 = true. Ако открие, че и P_1 също има намерение да влезе (flag_1 е вдигнат), възниква конфликт. Разрешаването на конфликта зависи от променливата turn:

- Ако P_0 има приоритет (turn == 0), той преминава към критичната секция
- Ако P_0 няма приоритет (turn == 1), се изпълнява блокът за отстъпване:
    - P_0 временно оттегля своето намерение, като задава flag_0 = false
    - След това P_0 влиза в цикъл на busy-waiting: while (turn == 1**) ;**

  - P_0 чака P_1 да влезе, да завърши критичната секция и да промени turn обратно на 0 при излизане
- Едва след като turn се промени, P_0 подновява заявката си (flag_0 = true) и се опитва отново да влезе.

Това временно оттегляне (свалянето на флага) е решаващо: то позволява на приоритетния процес (P_1) да види, че другият процес (P_0) вече не се интересува (въпреки че P_0 само изчаква), като по този начин P_1 може да продължи без риск от *deadlock*.

Въпреки, че този механизъм е коректен, логическата му сложност е висока. По-големият брой инструкции, необходими за отстъпване (флаг-запис, *busy-wait*, флаг-запис), представлява по-широк прозорец на изпълнение в *entry section*, който може да бъде прекъснат от планировчика на ОС. Това потенциално въвежда по-висок дисперсен *overhead* при *busy-waiting* в сравнение с по-простата логика на Peterson.

АЛГОРИТЪМЪТ НА PETERSON: ЕЛЕГАНТНОСТ И СТРОГОСТ

Алгоритъмът на Peterson е публикуван през 1981 г. много по-късно от Dekker. Основната мотивация на Peterson е да предложи значително по-прост и по-елегантен софтуерен алгоритъм за два процеса, чиято коректност е по-лесна за доказване.

Peterson използва същите разделяеми променливи флагове за индикация на намерение и turn за приоритет. Стъпките на алгоритъма за процес P_0 са:

```
flag_0 = true;              // заявка за влизане в CS
turn  = 1;                  // предоставяне на ред на конкурента
while ( (flag_1) and  // проверка на заявката на конкурента и
        (turn == 1) )   ;   // ако е ред на конкурента: Busy-waiting

      // Critical Section

flag_0 = false;             // отмяна на заявката
```

Ключовият елемент, който отличава Peterson, е механизмът за делегиране на приоритета. Чрез установяването на turn = 1 (давайки приоритет на конкурента), P_0 се съгласява да изчака, ако P_1 е в конфликт и P_1 все още поддържа своя флаг активен. Взаимното изключване се гарантира, тъй като в случай на състезание (и двата процеса са задали флага си на true), само този, който не е получил приоритет чрез последното записване в turn, ще бъде принуден да чака.

Простотата на Peterson е структурно предимство. Секцията за вход е по-кратка и съдържа по-малко логически стъпки в сравнение със сложния механизъм за оттегляне на Dekker. Тази относителна простота прави алгоритъма теоретично по-ефективен, тъй като минимизира броя на сложните операции по споделена памет, които могат да бъдат прекъснати, въпреки че основният проблем с *busy-waiting* остава непроменен.

ФУНДАМЕНТАЛЕН ДЕФЕКТ НА PETERSON ПРИ ВРЕМЕДЕЛЕНИЕ: АНАЛИЗ НА "ЗАМРАЗЕНИЯ КВАНТ"

"Замразения квант" е ситуация, при която CPU-време се губи поради блокиращо чакане на процеса с приоритет, причинено от другия процес, който не може да напредне, но не може и да освободи заетия ресурс.

Алгоритъмът на Peterson не предвижда механизми за адаптация към прекъсвания от планиращата стратегия. Агресивно-вдигнатият флаг остава в такова състояние до достигане на критичната секция, независимо от времевите ограничения. За разлика от Dekker тук липса "отстъпване" и не се предоставя възможност за временно освобождаване на ресурса. Така задържаният на границата на заета CS процес (напр. P_1), чрез вдигнатия си флаг

(flag_1) блокира възможността на другия процес за достъп до CS, което води до „Замразен квант“ и неефективно използване на предоставеното му CPU-време.

В таблицата е представен сценарий на изпълнението на двата процеса при Peterson.

Инициализираща процедура: flag_0 = false; flag_1 = false;

t	P_0	P_1	*Забележка*
1	flag_0 = true		*P_0 заявка за CS*
2	turn = 1		*ред на конкурента*
3	(flag_1 == true) and (turn==1)		*flag_1 е false*
4	CS		*P_0 влиза в CS*
5	CS		*Прекъсване от ОС*
6		flag_1 = true	*P_1 заявка за CS*
7		turn = 0	*ред на конкурента*
8		(flag_0 == true) and (turn==0)	*блокиране*
9		busy-waiting	*Блокиран **КВАНТ***
10		busy-waiting	*Прекъсване от ОС*
11	CS		*P0 излиза от CS*
12	flag_0 = true		*P_0 заявка за CS*
13	turn = 1		*ред на конкурента*
14	(flag_1 == true) and (turn==1)		*блокиране **flag_1***
15	**frozen quantum** waiting		***Замразен КВАНТ***
16	…		

СРАВНИТЕЛЕН АНАЛИЗ. МЕТРИКИ ЗА ОЦЕНКА

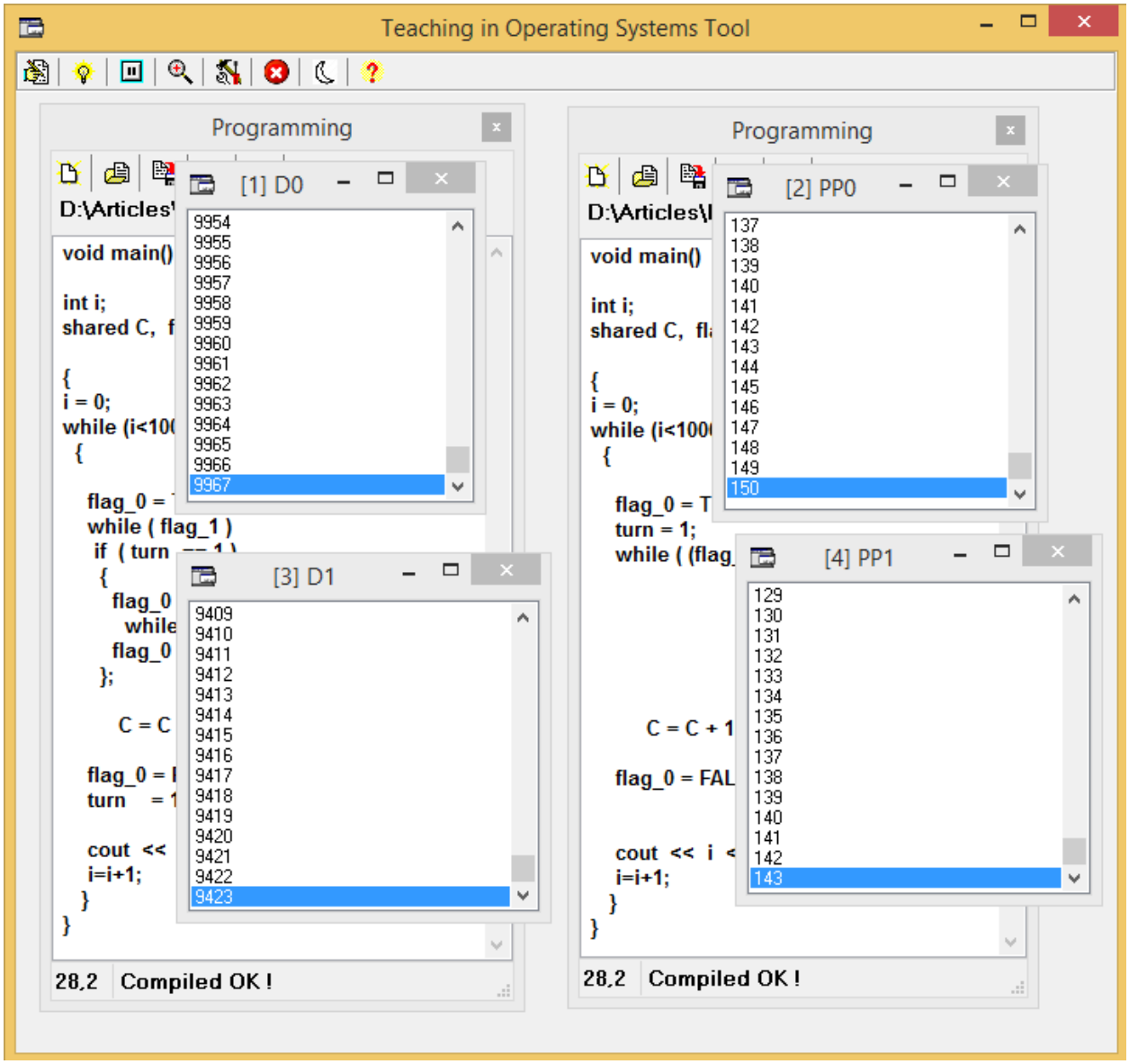
Дефинирани метрики:

- Пропускателна способност - брой успешни влизания в критичната секция за единица време
- Ефективност на кванта - процент използване на процесорно време
- Латентност - средно време за изчакване за достъп
- Справедливост - равномерност на разпределение на достъпа

Конфигурация на симулацията:

- Квант от време: 50ms и 100ms
- Брой процеси: серии от 2-4 конкурентни процеса
- Интензивност на конкуренция: висока (непрекъснати опити за достъп до CS)
- Продължителност: 10000 цикъла на влизане в CS за всеки процес

Въпреки че и двата алгоритъма страдат от проблема *busy-waiting*, анализът при висока конкуренция в среда с времеделение показва значителни разлики. На фиг.1 е показана експерименталната постановка във виртуалната машина TOST (Golemanov and Golemanova, 2020). В отделни редакторски прозорци са заредени програми с алгоритмите на нулевите процеси D0 (на Dekker) и PP0 (на Peterson). Моментът на конкретното наблюдение е малко преди завършване на 10000-те цикъла на двата процеса на Dekker (D0 и D1) и двата процеса на Peterson (PP0 и PP1) при 100 ms квант.



Фиг. 1 Програмиране и изпълнение на процеси на Dekker и Peterson в TOST

Резултати и интерпретация:

В таблиците по-долу са представени резултатите от експериментите по отношение на показателите Производителност и Оползотворяване на процесорното време.

Таблица 3 Производителност

Критерий	Peterson	Dekker
50 ms	~1 влизания в CS	~2 влизания в CS
100 ms	~1 влизания в CS	~10 влизания в CS
Средно време на чакане при 100 ms квант	70-80 ms	20-30 ms
Използване на кванта	~10 %	~90 %

Интерпретация: Алгоритъмът на Dekker демонстрира значително по-добра производителност благодарение на механизма за отстъпване, който позволява по-ефективно използване на процесорното време. При висока конкуренция се проявяват значителните различия в дизайна, като Dekker поддържа стабилна производителност, докато тази на Peterson рязко спада.

Таблица 4 Оползотворяване на CPU

Критерий	Peterson	Dekker
% “Пропилени“ квантове	50-60%	10-20%
Средна продължителност на “пропиляното“ CPU време	2.8 кванта	1.2 кванта

Интерпретация: Peterson страда от не само по-чести, но и по-продължителни периоди на "замразяване", което обяснява значително по-ниската му ефективност.

ЗАКЛЮЧЕНИЕ

Това изследване демонстрира, че алгоритъмът на Peterson, въпреки простотата и елегантността си, страда от сериозен недостатък в системи с времеделение - проблема на "замразения квант". Алгоритъмът на Dekker, чрез своя механизъм за временно отстъпване, предлага значително по-добра производителност в условия на интензивна конкуренция. Подробният анализ на сценариите на изпълнението, както и на експерименталните данни показва, че:

- При голям квант в алгоритъма на Peterson реално възниква проблем с „еднократното излизане“ в CS и загуба на голяма част от CPU-времето поради механизма за редуване, основан на променливата за ред (turn) и статусите на флаговете. Това води до голяма неефективност - до 99% от времето може да се „прахосва“ в чакане и синхронизация.
- При алгоритъма на Dekker поведението е по-различно: задържаният процес сваля своя флаг, позволявайки на другия процес да използва пълноценно своя квант без да губи време в зацикляния. Това осигурява много по-ефективно усвояване на времето в сравнение с Peterson при такива условия.

Въпреки че и двата алгоритъма имат предимно историческо и образователно значение в днешни дни, тяхният анализ остава важен за разбиране на фундаменталните принципи на междупроцесната синхронизация. Проблемът на "замразения квант" служи като добър пример за това как теоретично коректни решения могат да имат незадоволително практическо поведение в определени контексти.

БЛАГОДАРНОСТИ

Този доклад се публикува с подкрепата на проект 2025-ФЕЕА-02 „Приложение на AI в различни професионални сфери: възможности, предизвикателства и етични въпроси“, финансиран от фонд „Научни изследвания“ на Русенския университет „Ангел Кънчев“.

REFERENCES

Dijkstra, E. W. (1965) ‘Solution of a problem in concurrent programming control’, *Communications of the ACM*.

Golemanov, T. and Golemanova, E. (2020) ‘Using TOST in Teaching Operating Systems and Concurrent Programming Concepts’, *Advances in Science, Technology and Engineering Systems*, 5(6). doi: 10.25046/aj050610.

Peterson, G. L. (1981) ‘Myths About the Mutual Exclusion Problem’, *Information Processing*.

Silberschatz, A. (2018) *Operating System Concpets*, *Wiley*.

Tanenbaum, A. S. and Bos, H. (2016) *Modern Operating Systems*, *Education*. Pearson India; 4th edition.