**Доклад**

По

Конкурентно програмиране

Какво представлява състезателното състояние

(Race Condition)?

1. Въведение
2. Състезателно състояние *(Race Condition)*
3. Проверете-след-действайте *(Check-then-act)*
4. Четене-Промяна-Писане *(Read-Modify-Write)*
5. Откриване
6. Елиминиране

* Избягване на Споделено състояние
* Използване на синхронизации и атомарни операции

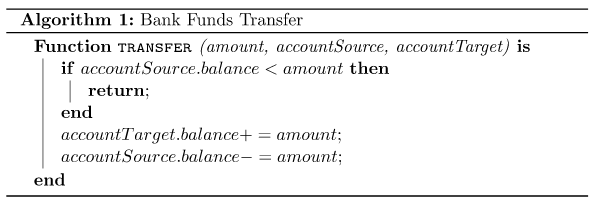
1. Надпревара с данни *(Data Race)*
2. Източници

Изготвил: Ваня Ванева 11а

**1-Състезателно състояние *(Race Condition)***

По дефиниция условието за състезание е състояние на програма, при което нейното поведение зависи от относителното синхронизиране или преплитане на множество нишки или процеси. Един или повече възможни резултати може да са нежелани, което да доведе до грешка. Ние наричаме този тип поведение недетерминистично.

Проста функция за извършване на превод на средства между две банкови сметки:

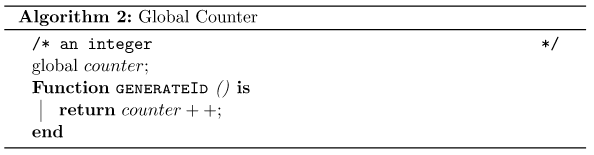


**2-Проверете-след-действайте *(Check-then-act)***

Това е най-често срещаният тип състезателно състояние. Дефинира се от програмен поток, където потенциално остаряло наблюдение се използва, за да се реши какво да се прави по-нататък. Мързеливата инициализация е пример за модел на проверка и действие.

**3-Четене-Промяна-Писане *(Read-Modify-Write)***

Въпреки че типът „проверете след това действайте“ състезателно състояние наистина е най-често срещаният тип, който можем да срещнем в многонишкови приложения, има друг, по-лесен за разбиране тип.



**4-Откриване**

Едно състезателно състояние обикновено е трудно за възпроизвеждане, отстраняване на грешки и елиминиране. Ние описваме бъговете, въведени от условията на състезанието, като хейзенбъгове. Тъй като условията на състезание са свързани със семантиката на приложението, няма общ начин за откриването им. Многонишковите единични тестове с фокус върху стабилността на резултатите от теста ще помогнат, но е малко вероятно да предоставят 100% гаранция.

**5-Елиминиране**

**Избягване на Споделено състояние:**

Неизменните обекти, чието състояние не може да бъде променено след конструиране, са по своята същност безопасни за нишки. Използването на неизменни обекти, доколкото е възможно, винаги е препоръчително.

Локалните за нишката променливи, локализирани по такъв начин, че всяка нишка има свое лично копие, също са безопасни за нишката, тъй като са локални за всяка нишка.

**Използване на синхронизации и атомарни операции:**

Примитивите за синхронизиране, като критични секции, се използват, за да се гарантира, че определена част от програмата не може да бъде изпълнена от повече от една нишка едновременно. Заключването е механизъм за синхронизиране за налагане на поведение на критична секция на ниво нишка. Мутексът е една и съща абстракция, съществуваща в множество системни процеси.

**6-Надпревара с данни *(Data Race)***

Състезание за данни възниква, когато две нишки имат достъп до една и съща променлива едновременно и поне един от достъпите е запис. Концепцията за надпревара на данни е по-специфична за достъпа до паметта в конкретен модел на паралелност и следователно варира в различните платформи.

**7-Източници**

<https://www.baeldung.com/cs/race-conditions>