*Изпитна тема № 12:* **Конкурентно програмиране**

*Конкурентност. Изпълнение на програма. Процес. Блокиращи операции. Видове блокиращи операции. Нишка. Връзка между процес и нишка. Създаване на нишки. Управление на нишки. Синхронизация между нишки. Проблеми при работа с нишки и асинхронни операции. Работа с асинхронни операции. Обещания (Promise/Task) и обратни извиквания (Callback). Работа с асинхронни операции чрез async/await и други механизми за реализиране на асинхронни операции.*

**Конкурентност**-Извършване на няколко действия едновременно. Конкурентоспособността е полезна във всяка ситуация, в която дадено приложение трябва да прави няколко неща по едно и също време.

**Изпълнение**

Има различни методи за осъществяване на конкурентно програмиране. Например изпълнение на всяко изчислително изпълнение като процес на операционната система или осъществяване на изчислителен процес като набор от нишки в рамките на един процес на операционната система.

Конкурентно взаимодействие и комуникация

--Споделена памет: конкурентните компоненти си комуникират чрез промени по съдържанието на обща памет (примерно Java и C#). Този вид конкурентно програмиране обикновено изисква някаква форма на заключване за да може да се координират отделните нишки. Програма, която изпълнява всички от тях, се нарича thread-safe.

--Предаване на съобщения: компонентите си общуват чрез размяна на съобщения. Обменът на съобщения може да се извършва асинхронно или синхронно, в който изпращачът се блокира, докато не изпрати съобщението. Пращането на асинхронните съобщения може да е ненадеждно. Предаването на съобщения тенденциозно е по-лесно от Споделената памет и се смята за по стабилна форма на конкурентно програмиране. Предаването на съобщения може ефективно да се осъществи върху симетрични микропроцесори, със или без споделена памет.

**Блокираща операция**

Операция, която блокира продължението на изпълнение на нишката до настъпване на дадено условие.

Примери:

● прочитане на файл

● достъп до отдалечен ресурс

● извършване на тежък изчислителен процес

● Изпълнение на изчакващ метод (wait)

**Процесът** и нишката са свързани по същество. Процесът е изпълнение на програма, докато нишката е изпълнение на програма, управлявана от средата на процеса.

Друг основен момент, който разграничава процеса и нишката, е, че процесите са изолирани помежду си, докато нишките споделят памет или ресурси помежду си.

Процесът е изпълнение на програма и изпълнява съответните действия, посочени в програмата, или е единица за изпълнение, където се изпълнява програма. Операционната система създава, планира и прекратява процесите за използване на процесора. Останалите процеси, създадени от основния процес, са известни като детски процес.

Процесът е съвкупността от памет, стек и код на приложението. Операционната система работи с процеси, които потребителите възприемат като приложения - това са две имена за едно и също понятие. Един процес може да изисква паралелно изпълнение на повече от една задача. Затова процесите са съставени от една или повече нишки, които се изпълняват едновременно от гледна точка на потребителя. Нишката е основната единица, за която се заделя процесорно време.

**Нишката** е изпълнение на програма, която използва ресурсите на процеса за изпълнение на задачата. Всички нишки в рамките на една програма логично се съдържат в процеса. Ядрото разпределя стека и блока за управление на нишката (TCB) на всяка нишка. Операционната система записва само показалеца на стека и състоянието на процесора в момента на превключване между нишките на същия процес.

Нишките се изпълняват по три различни начина; това са нишки на ниво ядро, нишки на потребителско ниво, хибридни нишки. Нишките могат да имат три състояния, готови и блокирани; тя включва само изчислително състояние, а не разпределение на ресурсите и състояние на комуникация, което намалява режима на превключване. Увеличава паралелността (паралелизъм), следователно скоростта също се увеличава.

Нишката трябва да има свойство за приоритет, когато са активни няколко нишки. Времето, което получава за изпълнение съответно на други активни нишки в същия процес, се определя от приоритета на нишката.

**Свойства на нишката:**

•Само едно системно обаждане може да създаде повече от една нишка (лек процес).

•Нишките споделят данни и информация.

•Threads споделя инструкции, глобални и heap региони, но има собствен индивидуален стек и регистри.

•Управлението на нишки консумира не или по-малко системни обаждания, тъй като комуникацията между нишките може да бъде постигната с помощта на споделена памет.

•Свойството на изолация на процеса увеличава неговите разходи по отношение на потреблението на ресурси.

1.Всички нишки на програмата логично се съдържат в процеса.

2.Процесът е тежко претеглян, но нишката е леко претеглена.

3.Програмата е изолирана единица за изпълнение, докато нишката не е изолирана и споделя памет.

4.Нишката не може да има индивидуално съществуване; той е прикачен към процес. От друга страна, процес може да съществува индивидуално.

5.В момента на изтичане на нишката, свързаният с нея стек може да бъде възстановен, тъй като всяка нишка има свой стек. За разлика от това, ако даден процес умре, всички нишки умират, включително процеса.

Процесите се използват за постигане на изпълнение на програмите едновременно и последователно. Докато нишката е единица за изпълнение на програмата, която използва средата на процеса, когато много нишки използват средата на същия процес, те трябва да споделят кода, данните и ресурсите му. Операционната система използва този факт, за да намали режийните разходи и да подобри изчисленията.

**Нишките се създават** чрез разширяване на класа Thread. След това разширеният клас Thread извиква метода Start () , за да започне изпълнението на дъщерната нишка.

*Видове нишки в C#*

● Нишка на главния метод (main thread) - нишката, с която се стартира

и изпълнява стандартно програмата

● Нишка на преден план (foreground thread) - нишка, която остава активна до приключване на работата си, дори когато главната нишка е приключила вече с изпълнението си. Thread обектът по подразбиране е foreground thread.

● Нишка на зададен план (нишка във фонов режим, background thread) - нишка, която прекратява работата си най-късно, когато и главната нишка я прекрати. За да направите една нишка във фонов режим, трябва да зададете свойство на обекта на нишката: thread.IsBackground = true

*Проблеми със стандартното създаване на нишки:*

● Нишките, които можем да създадем сами заемат доста памет

● Нишките, които създаваме сами отнемат допълнително време на програмата за изпълнение

● Когато нишката приключи работа тя се унищожава (това също отнема време)

*Алтернатива: ThreadPool (пул от нишки)*

● Пул, който съдържа определено количество създадени от .NET нишки

● Когато трябва да се изпълни дадена операция се взема нишка от този пул (пести се време и памет от създаване)

● При приключване на операцията нишката се връща в пула, но не се изтрива (пести се време и памет от изтриване)

● Полезно при голямо текучество на нишки

● ВАЖНО: Добре е да се използва за нишки, работещи във фонов режим

*Проблеми, свързани с нужда от синхронизация*

● Понякога трябва да използваме нишки, но те трябва да се изчакват (синхронизират)

● Понякога различни нишки боравят с общ ресурс

● Понякога една нишка е зависима от друга нишка

*Използване на lock*

● lock създава ключалка, която заключва даден обект, докато трае

изпълнението на определен код. Когато изпълнението приключи, обектът се отключва.

● По този начин можем да заключим обекта от текущата нишка и когато тя се изпълни да го освободим

● Другите нишки не могат да достъпят обекта през това време

Видове проблеми, които могат да възникнат при употреба на нишки

● Race condition

● Deadlock

● Livelock

● Starvation

*Race condition*

● Race condition - състояние на гонка

● Две или повече нишки се опитват едновременно да променят общ ресурс

● Може да доведе до некоректни стойности

*Deadlock*

● Deadlock - мъртва хватка, взаимно заключване

○ Нишка 1 заема ресурс P

○ Нишка 2 заема ресурс Q

○ Нишка 1 заявява ресурс Q

○ Нишка 2 заявява ресурс P

● Т.е. Мъртвата хватка се случва, когато една нишка заема ресурс и заявява друг ресурс, който се използва от друга нишка в същото време.

● Мъртва хватка може да възникне и между по-голям брой нишки.

*Livelock*

● Livelock - самоизключващо се блокиране

● Нишка А вижда, че нишка B изисква ресурс и се опитва да го отстъпи

● От своя страна B също се опитва да го отстъпи

● Това може да продължава безкрайно.

*Starvation*

● Starvation - гладуване

● Нишка А използва общ ресурс, но го задържа често за дълго

● Нишка B получава ресурса, но на цената на често дълго чакане

● Нишка А е лакома, а Нишка B – гладуваща

**Асинхронно програмиране**

● Вид конкурентно програмиране

● Всяка група от команди се изпълняват на отделна нишка

● Създава се обект, който информира главната нишка при завършване или провал.

● Този обект се използва и за следене на прогреса

Обещания (Task)

● Task = Обект, който позволява асинхронното изпълнение на команди.

● Task = обект, представящ команди, които трябва да се изпълнят

● Task позволява да разберем дали работата е приключена и спомага за получаване на резултат от работата

Кога да използваме Task :

● Когато желаем да изпълняваме няколко действия паралелно

● Когато действията са свързани с нещо външно

○ Например: Достъпване на страница от уеб сървър

● Ако трябва да получим резултат от изпълнението

● Ако искаме да имаме възможност да отменим изпълнението

● Предпочитан избор в C#

● В други програмни езици има аналогични обекти - обещаниe

(Promise)

*Разлики м/у Thread и Task*

● Thread използва абстракциите на ОС за работа с нишки

● Task е начинът за изпълнение на асинхронен код и е част от Task

Parallel Library библиотеката на C#

● Task може да върне резултат. Няма директен начин да изкараме резултат от нишка.

● Task поддържа отмяна (cancellation) чрез cancellation tokens. Нишките нямат такава възможност

● Новата нишка е самостоятелен обект, а Task се базира на ThreadPool.

***Ключова дума async***

● Маркира метод като асинхронен

● Необходима, ако някъде в методите ще имаме действие, което ще изчакваме

● Един от начините за реализиране на асинхронно програмиране в C#

● Използва се заедно с ключовата дума await

● В други програмни езици (JavaScript, например) също има async/await концепция

***Ключова дума await***

● await поставя на пауза изпълнението на метода до получаването на резултат, но без да блокира викащата нишка

● await позволява изпълнението на задача без то да води до блокиране на нишката, в която е извикана

● Може да използвате await в рамките на метод отбелязан като async