

Конкурентност. Изпълнение на програма. Процес

Симеон Марков. 11А

1. **Конкурентността (concurrency)**

* Се явява важен инструмент в създаването на качествени и добри програми. В днешно време потребителите не обичат да чакат. Ако един потребител не успее да зареди уеб сайта, който иска да достъпи, в рамките на 2-3 секунди, то той често ще се откаже и ще посети конкурентен уебсайт. Въпросът стои не по-различно с приложенията – ако накараме потребителя да чака, то той често ще се откаже от употребата на нашето приложение, а ние бихме могли да го загубим като потребител и клиент. Оказва се, възможно е една и съща програма да прави няколко неща едновременно или поне привидно едновременно. Това се случва благодарение на конкурентността. Въпреки че конкурентността в програмирането съществува от дълго време, тя постоянно продължава да създава неприятности на програмистите. В днешно време все пак е възможно лесно да се пишат програми със средствата на конкурентното програмиране благодарение на библиотеки и съвременни средства в езиците за програмиране

|  |
| --- |
| Computer script on a screen  int number = int.Parse(Console.ReadLine());  if(number> 0) { Console.WriteLine(„Positive”); } else { Console.WriteLine(„Negative or zero”); }   * В нормалния случай програмите се изпълняват операция след операция. Това означава, че ако например имаме следния програмен фрагмент:   Изпълнение на програма |
|  |

|  |  |
| --- | --- |
| Какви форми на конкурентност съществуват?  * Оказва се, че съществуват различни форми на конкурентност. Една популярна такава е многопоточността (многонишковост). Това е форма на конкурентност, при която програмата се изпълнява чрез няколко потока, които могат да извършват различни действия. Употребата на тази форма е с цел оползотворяване на наличните ядра на процесора.  Паралелна обработка (паралелно програмиране)изпълнение на голям обем задачи чрез разпределение върху няколко потока (нишки), които се изпълняват едновременно.Многопоточност (Многонишковост)  * форма на конкурентност, която използва няколко програмни потока на изпълнение. |  |
| Laptop with phone and calculator |  |
| Асинхронно програмиране   * Друга съвременна форма на конкурентността е асинхронното прграмиране. При него действията, които трябва да се извършат се представят като операции, които предстои да бъдат завършени в бъдещ момент и се представят като специални обекти (обещания). В миналото този вид конкурентност е бил труден за реализиране чрез код. В последните години употребата му е изключително улеснена чрез подходящ синтаксис на езиците за програмиране. В C# се използват ключовите думи async и await.  1. **Асинхронно програмиране** - разновидност на конкурентността, която използва обещания (promises) или обратни извиквания (callbacks) с цел предотвратяване на създаването на излишни потоци. 2. **Обещание (promise)** -тип, който представлява операция, която ще бъде завършена в бъдещето. В .NET това са Task и Task. Аналози има и в други програмни езици. 3. **Обратно извикване (callback)** – функция/метод, който се изпълнява при приключването на дадена операция. Характерни са за някои по-стари имплементации на асинхронно програмиране и/или за някои технологии. 4. **Асинхронна операция (asynchronous operation) –** стартирана операция, която ще приключи след някакво време.        * **Стартиране на програмата -** При стартиране програмата се зарежда в оперативната памет (ОП). Тогава операционната система (ОС) създава абстракция за изпълнението на тази програма, която наричаме процес. Процесът изпълнява всички инструкции, зададени в програмата. Всеки процес има 4 части: стекова памет, динамична памет (heap memory), данни, текст. | |

## Процеси, Нишки, Time-slicing

* **Процесът** е инстанция на изпълнявана програма, която е изолирана от други процеси на същия компютър. По-специално, той има своя собствена частна част от паметта на машината. Абстракция на процеса е виртуален компютър. Това кара програмата да се чувства така, сякаш има цялата машина за себе си - сякаш е създаден нов компютър със свежа памет, само за да стартира тази програма. Точно както компютрите, свързани в мрежа, процесите обикновено не споделят памет. Един процес изобщо няма достъп до паметта или обектите на друг процес. Споделянето на памет между процесите е възможно на повечето операционни системи, но изисква специални усилия. За разлика от това, новият процес автоматично е готов за предаване на съобщения, тъй като се създава със стандартни входни & изходни потоци.
* **Нишка(Thread) -** Нишката е локус на контрола вътре в изпълнявана програма. Мислете за това като място в програмата, която се изпълнява, плюс стека от извиквания на методи, които са довели до това място, до което ще е необходимо да се върнете. Точно както процесът представлява виртуален компютър, абстракцията на нишките представлява виртуален процесор. Създаването на нова нишка симулира създаването на нов процесор във виртуалния компютър, представен от процеса. Този нов виртуален процесор изпълнява същата програма и споделя същата памет като другите нишки в процеса. Нишките автоматично са готови за споделена памет, защото нишките споделят цялата памет в процеса. Тя се нуждае от специални усилия, за да получите " **thread-local** " памет, която е частна за една нишка. Също така е необходимо да настроите предаването на съобщения**(message-passing )** изрично, като създавате и използвате структури от данни на опашка. Когато има повече нишки, отколкото процесори, конкурентността се симулира чрез **time slicing**, което означава, че процесорът превключва между нишките. На повечето системи **time slicing** се случва непредсказуемо и недетерминистично, което означава, че нишката може да бъде спряна или възобновена по всяко време.

Обобщение

1. **Конкурентност:** множество изчисления, изпълнявани едновременно
2. **Парадигми** за предаване на съобщения със споделена памет
3. **Процеси & нишки**

* Процесът е като виртуален компютър; нишка е като виртуален процесор.

1. **Race conditions**

* Когато коректността на резултата (постусловия и инварианти) зависи от относителното време на събитията.

5. **Конкурентните** бъгове са едни от най-трудните за намиране и поправяне бъгове и изискват внимателен дизайн, за да се избегнат.

6. Прогнозирането на това как конкурентният код може да се свърже с друг конкурентен код е много трудно за програмистите. Най-добре е да се проектира по такъв начин, че програмистите да не трябва да мислят за това