Изпълнение на програма. Процес. Конкурентност


CONCURRENT PROGRAMMING
Let's go. In and out. 20 minute adventure.
imgflip.com

Как се изпълнява програмата?

Нека имаме следното парче код:

```
int Sum(int a, int b) {
        return a + b;
}
int Multiply(int a, int b) {
        return a * b;
}
```

Как се изпълнява програмата?

И имаме следния код за Main():

```
void Main() {
        int a = 5, b = 10;
        Console.WriteLine(Sum(a, b));
        Console.WriteLine(Multiply(a, b));
}
```

Отговорът

1. Изпълнението започва от Main()
2. Инициализираме променливите a и b със стойностите 5 и 10
3. Извикваме метод Sum.
4. Контролът върху програмата се поема от него
5. Той пресмята сумата и връща стойността към метода, който го е повикал (Main)
6. Main методът отново държи контрола върху програмата. Върната от Sum стойност се извежда на екрана.
7. Повиква се Multiply метода
8. Контролът върху програмата се поема от Multiply
9. Той пресмята сумата и връща стойността към метода, който го е повикал (Main)
10. Main методът отново държи контрола върху програмата. Върната от Multiply стойност се извежда на екрана.

Въпросът

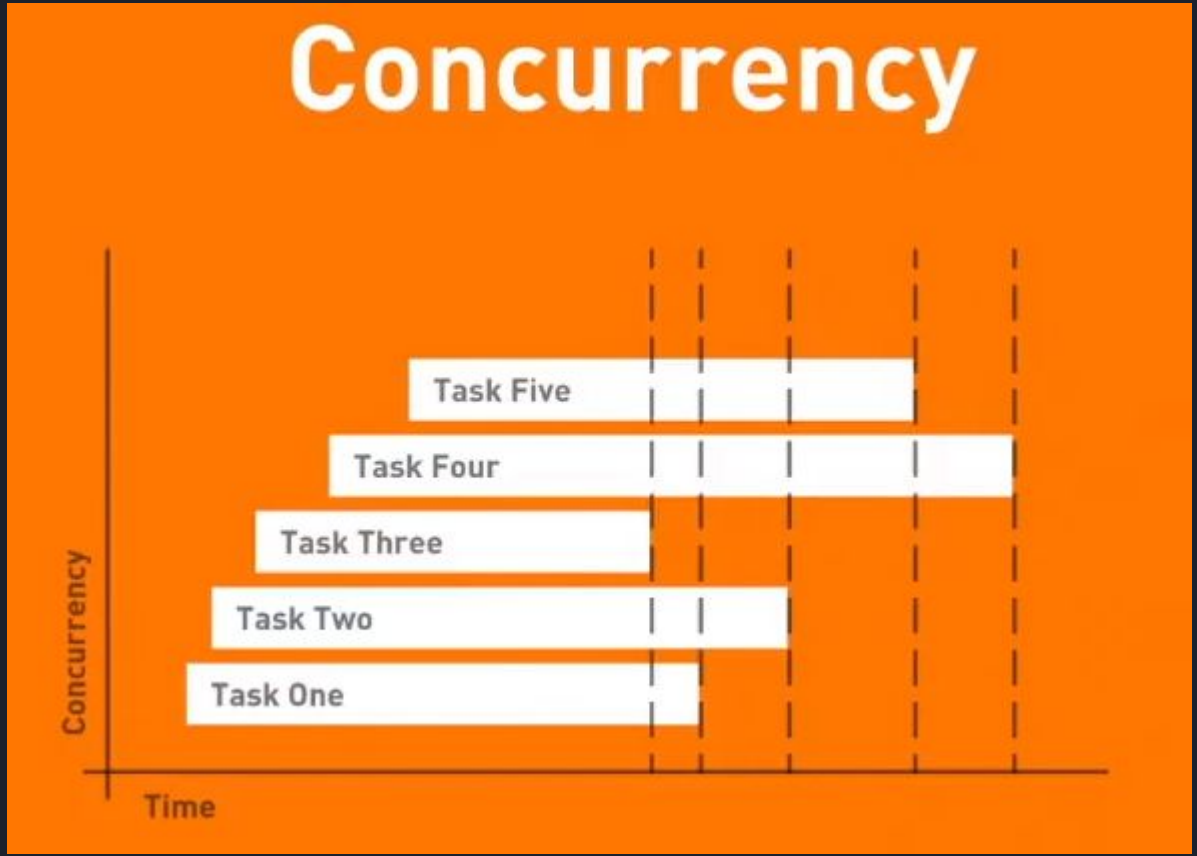
Какво ако Sum и Multiply трябва да извършат изчисления върху голям обем от данни? Трябва ли да чакаме единия да приключи, за да започне другия?

Ако **не знаем** за конкурентност и нишки - да!

Конкурентност

Изпълнението на няколко инструкции едновременно.

- При едно условие: инструкциите са независими една от друга:
  - Редът на изпълнение не е от значение
  - Да споделят възможно най-малко общи ресурси


Concurrency
Task Five
Task Four
Task Three
Task Two
Task One
Concurrency
Time

Кое е конкурентно и кое не?

Видове конкурентност

- Паралелно програмиране
  - Разпределяне на задача на подзадачи, които се разпределят чрез нишки за изпълнение от различни ядра на процесора паралелно; подходящо за тежи изчислителни задачи (CPU intensive)

Видове конкурентност

- Асинхронно програмиране
  - Разпределяне на задачи към друг извършител (нишка, устройство и др.), продължавайки да извършва друга работа, вместо да чака за отговор. Когато задачата е готова се изпълнява т.нар. функция на обратно извикване (callback), която информира главната нишка за завършването на задачата. Вместо функция за обратно извикване в някои програмни езици се среща обект, който описва задача, която не е завършила. Този обект се нарича future/promise/task.
    Асинхронното програмиране е добро за входно-изходни операции
- Разпределени изчисления (между отделни машини)

Процес

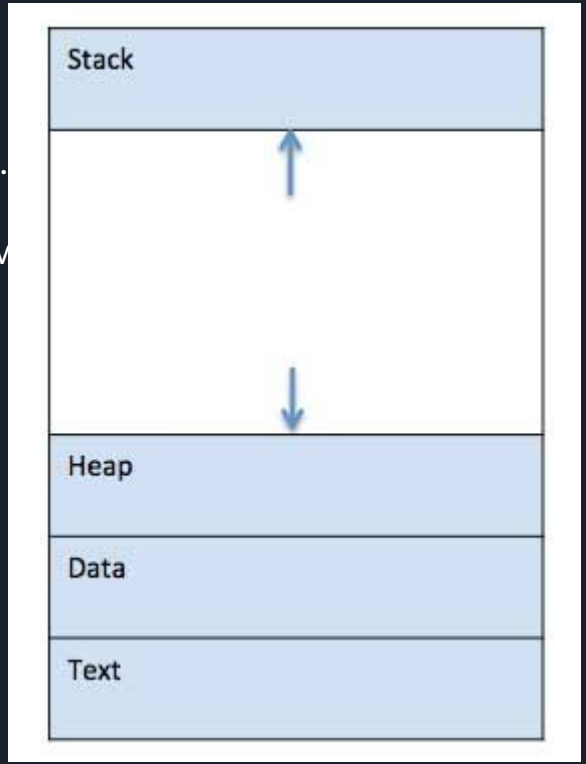
Процесът е програма, която е в състояние на изпълнение.

Програмата се превръща в процес зареждайки се в паметта.

Процесът изпълнява всички инструкции, зададени в програм

Зареденатa в паметта програма има 4 части:

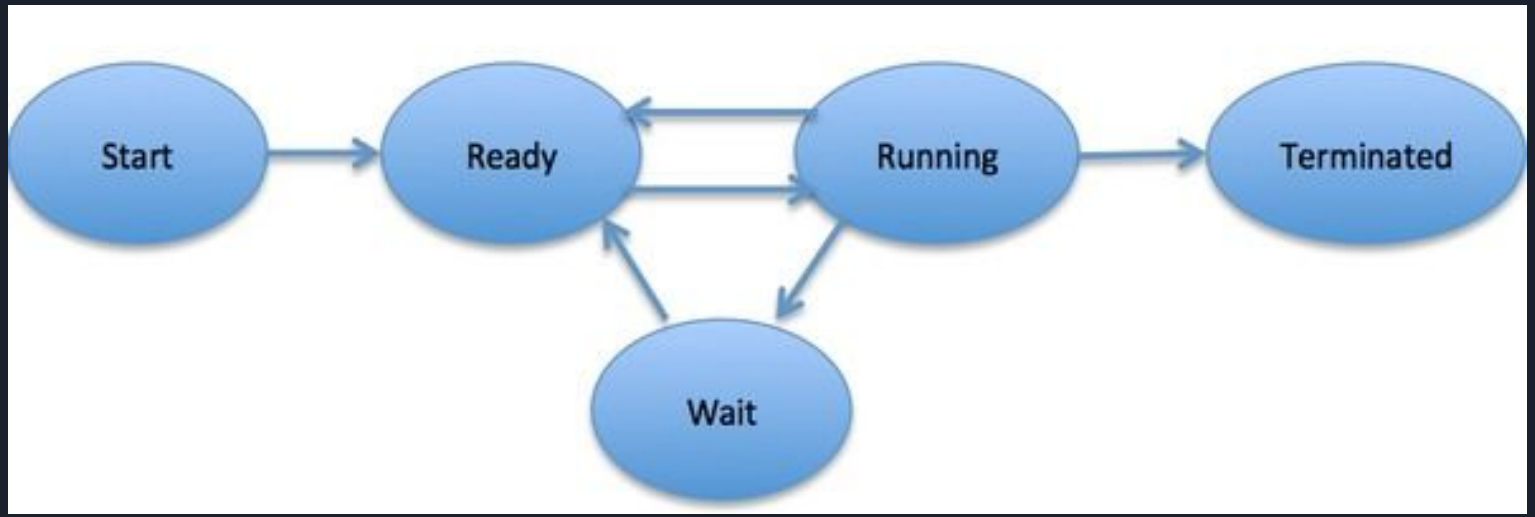
- Стекова памет
- Динамична памет (хийп, heap)
- Данни
- Тескт



Процес

- Стек - съдържа временни данни като данни за извикване на методи/функции и параметрите им, адрес на връщане на методите/функциите и локални променливи
- Хийп (Динамична памет) - съдържа динамично заделената памет за процеса - там се записват обекти (но не и обектните променливи), масиви и т.н.
- Текст - текущата дейност, представена от брояча на програмата и съдържанието на регистрите на процесора
- Данни - тук се пазят глобалните и статичните променливи

Жизнен цикъл на процеса



Благодаря за вниманието!
Автор: Петър Р. Петров,
ПГЕЕ „К. Фотинов”, гр. Бургас