Доклад

Тема: Нишки

1.Нишки

-В компютърните науки нишката **за** изпълнение е най-малката последователност от програмирани инструкции, които могат да се управляват независимо от [планировчик](https://en.wikipedia.org/wiki/Scheduling_(computing)) , който обикновено е част от операционната система . [1] Реализирането на нишки и [процеси](https://en.wikipedia.org/wiki/Process_(computing)) се различава в различните операционни системи. В [съвременните операционни системи](https://en.wikipedia.org/wiki/Modern_Operating_Systems)[Таненбаум](https://en.wikipedia.org/wiki/Andrew_S._Tanenbaum) показва , че са възможни много различни модели на организация на процеси. В много случаи нишката е компонент на процес. Множеството нишки на даден процес могат да се изпълняват [едновременно](https://en.wikipedia.org/wiki/Concurrent_computation)(чрез възможности за многопоточност), споделяне на ресурси като [памет](https://en.wikipedia.org/wiki/Shared_memory_(interprocess_communication)) , докато различните процеси не споделят тези ресурси. По-специално нишките на процес споделят неговия изпълним код и стойностите на неговите [динамично разпределени](https://en.wikipedia.org/wiki/Memory_management#HEAP) променливи и [глобални променливи, които не са](https://en.wikipedia.org/wiki/Global_variable)[локални за нишки](https://en.wikipedia.org/wiki/Thread-local_storage) във всеки даден момент.

2.Работа с нишки

-С Java създаването на многонишкови програми е изключително лесно. Достатъчно е да наследим класа **java.lang.Thread** и да припокрием метода **run()**, в който да напишем програмния код на нашата нишка. След това можем да създаваме обекти от нашия клас и с извикване на метода им **start()** да започваме паралелно изпълнение на написания в тях програмен код. Ето един пример, който илюстрира как чрез наследяване на класа **Thread** можем да създадем няколко нишки, които работят едновременно в рамките на нашето приложение:

Пример:

**class** MyThread **extends** Thread {

**private** String mName;

**private** **long** mTimeInterval;

**public** MyThread(String aName, **long** aTimeInterval) {

        mName = aName;

        mTimeInterval = aTimeInterval;

    }

**public** **void** run() {

**try** {

**while** (!isInterrupted()) {

                System.out.println(mName);

                sleep(mTimeInterval);

            }

        } **catch** (InterruptedException intEx) {

*// Current thread interrupted by another thread*

        }

    }

}

3.Създаване на нишки

-ThreadPool е клас, който дава възможност за асинхронното изпълнение на задачи като се грижи за управлението на стартираните нишки. Той може да се използва, когато не се налага създаването на методи за изпълнение, както и изричното създаване на нишка

Класът **java. lang. Thread позволява създаването на нови нишки**(threads)  
o Всяка нишка трябва задължително да наследи интерфейса  **Runnable**  
        - Изпълнявания код е разположен в нейния метод  run()  
o 2 метода за създаване на Thread :  
    - 1) производен клас на  java. lang. Thread  
            -- java. lang. Thread наследява Runnable  
            -- трябва да се предефинира метода run()  
    - 2) клас наследяващ интерфейса Runnable  
            -- трябва да се дефинира метода run()

<https://ff.tu-sofia.bg/AdvJava/Threads/Threads.html>

4.Управление на нишки

-Майкрософт въвежда и понятието нишка (на английски: *thread*) за всяка единица програмен код, състезаващ се за ресурсите на компютърната система. От един процес могат да бъдат генерирани („изтеглени“) една или повече нишки. От това следва и многонишковото изпълнение на програмния код, дори на една програма, като във връзка с това възниква въпросът за синхронизация на изпълнението на отделните нишки.

По принцип от една и съща програма могат да се генерират няколко едновременно изпълняващи се задачи – това трябва да бъде предвидено и програмирано още на ниво първичен код. В съвременните системи това се нарича многонишковост[[1]](https://bg.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%B8#cite_note-1).

Използването на нишки води до икономия на ресурси, тъй като създаването и унищожаването на нишка изисква по-малко време, а превключването на контекста между тях става по-бързо. Освен това нишките, които са част от един и същ процес, си поделят адресното пространство и файловете, и поради това те могат да комуникират без обръщане към ядрото, като така се подобрява ефективността при използване на многопроцесорни архитектури[[2]](https://bg.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%B8#cite_note-elsys-2)

<https://bg.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%B8>

5.Споделена памет между нишки

-Споделената памет между нишките се отнася до практиката да се позволи на множество нишки да имат достъп до един и същ блок памет едновременно. Това може да бъде полезна техника за подобряване на производителността на многонишкови програми, тъй като позволява на различни нишки да обменят данни и да комуникират по-ефективно.

Въпреки това, споделянето на памет между нишки може също да създаде редица потенциални проблеми и предизвикателства. Един често срещан проблем е известен като надпревара с данни, която възниква, когато две или повече нишки се опитват да осъществят достъп или да променят едно и също място в споделената памет едновременно, което води до непредсказуемо поведение и потенциални сривове на програмата.

За да се избегнат състезания с данни и други проблеми със синхронизирането, е важно да се използват подходящи техники за синхронизиране като ключалки, семафори или други механизми, които гарантират, че само една нишка може да има достъп до споделената памет в даден момент. Освен това може да се наложи използването на специализирани инструменти за програмиране или библиотеки, които предоставят допълнителна поддръжка за достъп до споделена памет и синхронизация в многонишкови програми.

6.Проблеми при работа с нишки

-Работата с нишки може да създаде няколко проблема, включително:

А) Състезателни условия: Когато множество нишки се опитват да получат достъп и да променят едни и същи данни едновременно, може да възникне състояние на състезание. Това може да доведе до противоречиви или неправилни резултати.

Б) Блокировки: Блокировки възникват, когато две или повече нишки чакат една друга да освободят ресурси, което води до ситуация, в която няма

В) Гладуване на нишки Липсата на нишка възниква, когато нишката не може да получи достъп до ресурсите на процесора, от които се нуждае, за да изпълни задачите си. Това може да се случи в ситуации, в които има твърде много нишки, конкуриращи се за ограничен брой ресурси на процесора.

-Има няколко често срещани причини за гладуване на конци:

-Прекомерна употреба на механизми за синхронизация: Ако твърде много нишки се борят за достъп до споделен ресурс, използвайки механизми за синхронизация като ключалки, семафори или монитори, тогава някои нишки може да бъдат блокирани за неопределено време в очакване на достъп.

-Лошо проектирани алгоритми: Някои алгоритми може да са лошо проектирани и да доведат до блокиране на нишки или изчакване за продължителни периоди от време. Това може да доведе до гладуване на нишка, ако алгоритъмът се използва в многонишкова среда.

-Конкуренция за ресурси: Ако нишките се конкурират за ограничени ресурси, като например памет или честотна лента на мрежата, тогава някои нишки може да нямат тези ресурси.

-Инверсия на приоритета: Ако нишка с нисък приоритет държи ресурс, който се нуждае от нишка с висок приоритет, тогава нишката с висок приоритет може да бъде блокирана за неопределено време в очакване на достъп до ресурса.

-За да се избегне гладуване на нишки, е важно внимателно да се проектират и тестват многонишкови приложения, за да се гарантира, че нишките не се конкурират за ограничени ресурси и че механизмите за синхронизиране се използват разумно. Освен това е важно да се наблюдава производителността на многонишковите приложения и да се настройва приложението според нуждите, за да се гарантира, че нишките нямат недостиг на CPU ресурси.

Г) Livelock е ситуация, която възниква, когато две или повече нишки активно се опитват да разрешат конфликт на ресурси, но не могат да постигнат напредък, защото многократно отговарят на действията на другия, без да постигнат реален напредък. С други думи, нишките са заети да правят нещо, но всъщност не напредват към изпълнение на задачите си.

Livelocks са подобни на задънените блокировки по това, че и двете включват ситуации, при които нишките не могат да постигнат напредък, но при livelock нишките активно се опитват да разрешат ситуацията, докато в задънена улица нишките са напълно блокирани и не могат да продължат.

Някои често срещани причини за брави включват:

# https://chat.openai.com/chat