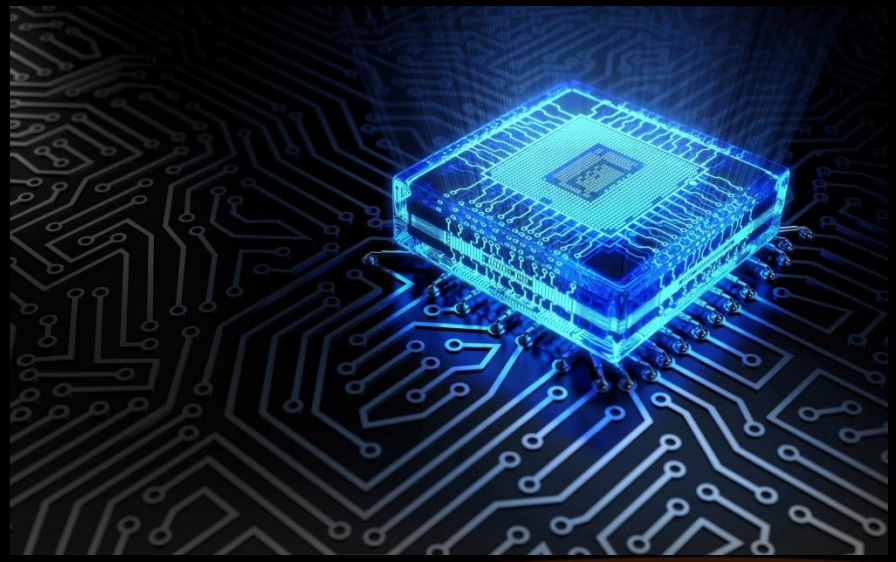
Нишки и Многонишково програмиране

Вградени системи



Инж. Венцеслав Кочанов

Какво е нишка?

Нишките са основните единици за изпълнение в програмата. Те позволяват едновременно изпълнение на код и могат да работят паралелно на множество процесорни ядра. Или казано иначе нишките са най-малките единици за изпълнение в рамките на процес и споделят едни и същи ресурси, като например пространство в паметта, докато работят независимо една от друга. Нишката не е просто, физическо нещо. Windows трябва да разпредели и поддържа обект на ядрото за всяка нишка, заедно с набор от спомагателни структури от данни. Но докато една нишка се изпълнява, част от нейното логическо състояние също се определя от хардуерното състояние, като например данни в регистрите на процесора.

От тук следва че състоянието на нишката се разпределя между софтуера и хардуера, докато работи. При дадена нишка, която работи, процесорът може да продължи да я изпълнява, а при дадена нишка, която не работи, ОС разполага с цялата необходима информация, за да може да планира нишката и да се изпълнява отново на хардуера.

Всяка нишка се прехвърля върху процесора от планировчика на нишки на Windows, което позволява реалното изпълнение на текущата работа.

Windows използва превантивно планиране на нишки, което му позволява принудително да спре изпълнението на нишка на определен процесор, за да изпълни друг код, когато е необходимо. Предимството предизвиква превключване на контекста, както беше обяснено по-рано. Това се случва, когато нишка с по-висок приоритет стане изпълнима или след изтичане на определен период от време (наречен квант или отрязък от време). И в двата случая, превключването се случва само ако няма достатъчно процесори, които да поемат двете въпросни нишки работещи едновременно; планировчикът винаги ще предпочита да използва напълно наличните процесори.

Нишките могат да блокират поради редица причини: явен I/O, трудна страница, повреда (т.е. причинена от четене или запис на виртуална памет, която е била изведена на диска от ОС), или чрез използване на един от многото примитиви за синхронизиране, на ядрото на Windows и синхронизиране на данни и управление. Докато дадена нишка блокира, тя не изразходва време или мощност на процесора, което позволява на други работещи нишки да напредват вместо нея. Актът на блокиране, както можете да си представите, променя структурата на данните на нишката, така че планировчикът на нишки на ОС да знае, че е станал недопустим за изпълнение и след това задейства превключване на контекста.

Когато възникне условието, което да деблокира нишката, тя отново става допустима за изпълнение, което я поставя обратно в опашката от изпълняеми нишки и планировчикът по-късно ще я насрочи за изпълнение, използвайки своите обикновени алгоритми за планиране на нишки. Понякога на събудените нишки се дава приоритет, за да се изпълняват отново, нещо, наречено повишаване на приоритета, особено ако нишката се е събудила в отговор на GUI събитие, като щракване на бутон. Има пет основни механизма в Windows, които рутинно причиняват нелокално прехвърляне на контрол. С други думи, IP адресът на процесора скача някъде много по-различно от това, което програмният код предполага, че трябва да се случи.

Първият е превключване на контекста, което вече сме виждали. ***Вторият*** е обработка на изключения. Изключение кара операционната система да изпълнява различни филтри и манипулатори за изключения в контекста на текущата изпълнявана нишка и, ако се намери манипулатор, IP адресът завършва вътре в него.

Третият механизъм, който причинява нелокално прехвърляне на контрол, е хардуерното прекъсване. Прекъсване възниква, когато възникне значимо хардуерно събитие, представляващо интерес, като завършване на I/O на някакво устройство, изтичане на таймера и т.н., и предоставя възможност за отговор на рутинна програма за изпращане на прекъсвания. Всъщност вече сме виждали пример за това: контекстните превключватели, базирани на изпреварване, се инициират от прекъсване, базирано на таймер.

Въпреки че прекъсването заема стека на режима на ядрото на текущо изпълняваната нишка, това обикновено не се забелязва: кодът, който се изпълнява, обикновено върши малко работа много бързо и изобщо няма да изпълнява код в потребителски режим.

Софтуерно базираните прекъсвания обикновено се използват в ядрото и системният код също, което ни води до ***четвъртия*** и ***петия*** метод: отложени извиквания на процедури (deferred procedure calls - DPC) и извиквания на асинхронни процедури (asynchronous procedure calls - APC). DPC е просто обратно извикване, което ядрото на ОС поставя на опашка, за да се изпълни по-късно. OPC работят на по-високо ниво на заявка за прекъсване (Interrupt Request Level - IRQL) от хардуерните прекъсвания, което просто означава, че те не задържат изпълнението на други хардуерно базирани прекъсвания с по-висок приоритет, ако такова се случи в средата на изпълнението на OPC.

Ако нещо трябва да се случи по време на хардуерно прекъсване, то обикновено се извършва от манипулатора на прекъсвания, поставящ DPC на опашка, за да изпълни тежката работа, която гарантирано ще се изпълни, преди нишката да се върне обратно в потребителски режим. Всъщност това е начинът, по който се извършват контекстни превключвания, базирани на изпреварване. APC е подобен, но може да изпълнява обратни извиквания в потребителски режим и да се изпълнява само когато нишката има никаква друга полезна работа за вършене, посочена от нишката, която въвежда нещо, наречено изчакване за предупреждение. Кога по-конкретно нишката ще извърши изчакване за предупреждение не е неизвестно и може никога да не се случи.

Следователно APC обикновено се използват за по-малко критична и по-малко чувствителна към времето работа или за случаи, в които извършването на предупредително изчакване е необходима част от модела на програмиране, спрямо който потребителите програмират. Както OPC, така и APC могат да бъдат планирани между процесорите да работят асинхронно и винаги да работят в контекста на нишката по време на изпълнение. Всяка нишка принадлежи към един процес, който има други интересни и подходящи данни, споделени между всичките му нишки - като таблица с манипулатори и таблица на страници с виртуална памет, но горното определение ни дава добра пътна карта за изследване на по-дълбоко ниво.

Преди всичко това, нека прегледаме какво прави една управлявана CLR нишка различна от собствената нишка. Това е въпрос, който възниква отново и отново.

Какво е CLR /Common Language Runtime/ и CLR нишка?
Common Language Runtime (CLR) е основен компонент на .NET Framework, който управлява изпълнението и жизнения цикъл на всички .NET приложения (код). Той предоставя различни услуги, включително автоматично управление на паметта, обработка на изключения, сигурност и безопасност. Когато .NET приложение се компилира, то генерира междинен езиков код, наречен Common Intermediate Language (CIL). CLR е отговорен за преобразуването на този CIL в машинен код и управлението на изпълнението на получената програма. Това е програмната среда, в която се изпълнява кодът на .NET приложенията (C#, F# и VB).

Представлява виртуална машина, която компилира междинния код - CIL (Common Intermediate Language) за конкретната хардуерна платформа и операционна система, с която работи потребителя. Използва се компилация по време на изпълнение или така наречения Just-In-Time /JIT/ compiler.

Функции на CLR

Функциите на CLR са няколко: - Изпълнението на IL кода и JIT компилацията; - Паметта и ресурсите на приложението; - Безопасността на типовете; - Сигурността; - Code access security; - Role-based security; - Изключенията; - Конкурентността; - Връзката с неуправляван код; - Процесите на debug и оптимизиране (profiling) при разработка на приложения.

CLR нишката е същата като Windows нишката . По известни са като „управлявани нишки“, което ги кара да звучат напълно различно от нишките на Windows. Но има разлика. На най-просто ниво, това ефективно не променя нищо за програмистите, които пишат паралелен софтуер, който ще работи на CLR. Можем да мислиме за нишка, изпълняваща управляван код, като точно същото нещо като нишка, изпълняваща собствен код, както е описано по-горе. Те наистина не са фундаментално различни, с изключение на някои езотерични и екзотични ситуации, които са повече теоретични, отколкото практически. Първо, прагматичната разлика: CLR трябва да проследява всяка нишка, която някога е изпълнявала управляван код, за да може CLR да изпълнява определени важни задачи. Състоянието, свързано с нишка на Windows, не е достатъчно.

Явна нишка и алтернативи

Това е начин на много ниско ниво за писане на паралелен софтуер. Понякога мисленето на това ниско ниво е неизбежно, особено за програмиране на системно ниво, а понякога и в приложения и библиотеки. Мисленето и управлението на нишките е трудно и може бързо да измести фокуса от решаването на реални алгоритмични домейни и бизнес проблеми. Ще откриеме, че изричните нишки бързо могат да станат натрапчиви и проникващи в архитектурата и изпълнението на вашата програма. Използването на пул от нишки вместо изрични нишки ни отдалечава от подробностите за управление на нишките и се връща към решаването на вашите проблеми с бизнеса или домейна.

Въпроси?