Моделиране на последователностна динамика чрез FSP нотация

Глава 1 – Резюме

Последователна програма: един поток на управлен. (един процес).

Конкурентна програма: множество потоци на управление (нишки):

- **Извършват множество изчисления паралелно.**
- **Контролират множество външни събития, случващи се едновременно.**

Локално взаимодействие: В рамките на една машина чрез споделена памет.

Мрежово взаимодействие: Между множество машини чрез мрежови протоколи.

Защо е нужно конкурентно програмиране?

- Увеличаване на производителността: Множество процеси могат да се изпълняват паралелно, използвайки многоядрени процесори.
- По-добра отзивчивост: Входно-изходните операции не блокират цялата програма.
- По-подходяща структура: За програми, които управляват множество устройства и събития.

Примери за грешки в конкурентното програмиране:

Therac-25: Компютъризирана машина за лъчелечение, при която грешки в конкурентното програмиране водят до сериозни наранявания.

Mars Rover: Проблеми в взаимодействието на задачите водят до рестартиране на софтуера и загуба на време за изследвания.

Рискове и предизвикателства: Проблеми с безопасността и надеждността на системите.

Моделиране на конкурентни системи: Използване на модели за представяне на реалния свят и проверка на адекватността на дизайна. Моделиране с помощта LTS/FSP

Моделите се описват с помощта на състояния и преходи, известни като **Означени преходни системи (Labelled Transition Systems, LTS)**. Това са формализирани представяния на динамични системи, при които всяко състояние на системата се свързва с възможни действия или събития, а преходите между състоянията са етикетирани с тези действия. Описанието на такива модели обикновено се извършва текстово с помощта на **Финитни състояния процеси (Finite State Processes, FSP)**, които представляват специализиран език за описание на динамично поведение на системи, като се дефинират състояния и преходи между тях. За визуализиране и анализиране на тези модели, обикновено се използва инструментът **LTSA (Labelled Transition System Analyzer.**

Програмиране на конкурентни системи с Java:

-Java като език за илюстративни примери и упражнения.

-Конструиране на Java програми за с-ми като Cruise Control System.

Процеси и нишки (Threads)

Процесите и **нишките** са основни концепции в многозадачността (concurrent execution) и многопоточността (multithreading).

- **Процес** е независим изпълняващ се модул, който има собствено адресно пространство и ресурси, като памет и регистри.

- **Нишка** (или поток) е по-малка единица изпълнение в рамките на даден процес. Нишките споделят ресурси и адресно пространство с останалите нишки в същия процес.

Конкурентно изпълнение (Concurrent Execution)

Конкурентното изпълнение означава, че множество процеси или нишки се изпълняват "паралелно" (въпреки че може да не са наистина едновременно, в зависимост от броя на процесорите) в една операционна система. Множество нишки или процеси могат да се изпълняват по ред, като операционната система редува между тях, като предоставя време на всяка нишка или процес.

Споделени обекти и интерференция (Shared Objects & Interference): Когато нишки или процеси имат достъп до **споделени обекти** (напр. глобални променливи или ресурси), може да възникне **интерференция**. Това е ситуация, при която едновременното четене и записване върху споделени обекти води до неочаквани или некоректни резултати. Тази интерференция може да доведе до трудни за откриване грешки (interference bugs).

Монитори и синхронизация на условия (Monitors & Condition Synchronization); Мониторите са синхронизационни механизми, които осигуряват безопасен достъп до споделени ресурси. Те предоставят структури за управление на достъпа до споделени обекти, като осигуряват **взаимно изключване** (**mutual exclusion**) и възможност за **синхронизация на условия** (condition synchronization).

- Мониторът осигурява, че само един поток може да изпълнява даден код в даден момент, когато се използва споделеният ресурс.
- **Синхронизация на условията** се използва, за да се осигури, че един поток ще изчака, докато не бъдат изпълнени определени условия (например, когато ресурсът стане наличен).

Задръжка (Deadlock): възниква, когато два или повече потока не могат да напредват поради взаимно изчакване на ресурси, които са задържани от други потоци. Например, ако поток А държи ресурс X и чака ресурс Y, докато поток Б държи ресурс Y и чака ресурс X, това води до задръжка, тъй като никой от потоковете не може да продължи.

Свойства на безопасността и живостта (Safety and Liveness Properties)

- **Свойства на безопасността** се отнасят до условия, които гарантират, че системата няма да достигне некоректни или нежелани състояния. Например, не трябва да се позволи на два потока да извършват операции върху един и същи ресурс по едно и също време, ако това води до грешки.
- **Свойства на живостта** се отнасят до условия, които гарантират, че системата няма да попадне в ситуация, в която не може да извършва напредък. Това включва избягване на задръжка и гарантиране, че всяка нишка или процес ще получи възможност за изпълнение след определено време.

Моделно-базирано проектиране (Model-based Design) е методология за разработка на софтуерни системи, при която процесите и системите се моделират предварително чрез

математически модели, преди да се реализират в код. Това помага за по-добро разбиране на взаимодействията между компонентите на системата, намалява възможността за грешки и улеснява анализата на безопасността, ефективността и други характеристики на системата преди нейното изграждане.

Динамични системи (Dynamic Systems) се отнасят до системи, чиято конфигурация и поведение се променят с времето, въз основа на външни или вътрешни фактори. В контекста на софтуерните системи, динамичността може да се отнася до промени в състоянието на системата или взаимодействията между различни компоненти, които се развиват по време на изпълнението. Тези системи могат да бъдат моделирани и анализирани чрез различни методи, включително симулации и математически модели.

Изпращане на съобщения (Message Passing) е техника, използвана за комуникация между процеси или нишки в конкуретни или паралелни системи. Вместо да споделят общо пространство в паметта, процесите или нишките обменят съобщения, което означава, че всяка страна изпраща информация към другата по определен начин, обикновено чрез канали за комуникация или съобщителни буфери. Тази техника е основна при моделирането на паралелни и разпределени системи.

Конкурентен софтуер и архитектури (Concurrent Software and Architectures) - софтуер, който включва паралелни процеси или нишки, които могат да се изпълняват едновременно, но в един и същ компютър или разпределена система. **Конкурентните архитектури** се отнасят до дизайна на хардуер и софтуер, които поддържат едновременни операции, като използват многопроцесорни системи

или многоядрени процесори. Това включва теми като синхронизация, съвместен достъп до ресурси и управление на паралелни операции.

Времеви системи (Timed Systems)- системи, в които времето играе важна роля за тяхното поведение. Те включват механизми за управление на времето, например, ограничаване на времето за изпълнение на операции или обработка на събития, които трябва да се случват в точно определен ред. Тези системи могат да бъдат използвани в области като вградените системи, мрежовите протоколи и управлението на процеси в реално време.

Програмно верифициране (Program Verification) - процесът на доказване на коректността на програми. Това може да се извършва чрез различни техники, като математическо доказателство, симулация или използване на верификационни инструменти. Целта е да се гарантира, че дадена програма изпълнява своите задачи правилно и съответства на зададените изисквания.

Логически свойства (Logical Properties) в контекста на софтуера се отнасят до математическите характеристики и условия, които трябва да бъдат изпълнени от програмата. Това включва логически изрази и правила, които дефинират вярността на програмата. Примерите за логически свойства могат да включват инварианти (непроменящи се условия по време на изпълнението на програмата), правилата за доказателства и типови системи. Логическите свойства играят централна роля в програмен верификационен процес, тъй като те помагат да се удостоверят правилността и стабилността на системата.