

## Глава 3: Конкурентно изпълнение

**Интерференция:** Проблем: Ако две нишки едновременно четат/пишат в споделен обект, могат да възникнат грешки.

**Пр.:** Една нишка чете стойност, друга я променя, преди първата да запише резултата.

**Пр.Орнаментална градина:** Две нишки (входове East и West) увеличават броя на посетителите. Проблем: Броячът не отчита правилно, защото двете нишки записват стойности едновременно.

**Решение:** Взаимно изключване

**Синхронизиране в Java:** Методите се маркират с ключовата дума **synchronized**.

```
synchronized void increment() {  
    value++;  
}
```

По този начин само една нишка може да достъпи метода.

FSP: Моделът включва заключване:

**LOCK = (acquire -> release -> LOCK).**

**TURNSTILE = (go -> RUN),**

**RUN = (arrive -> INCREMENT | end -> TURNSTILE),**

**INCREMENT = (value.acquire -> value.read[x:T] -> value.write[x+1]  
-> value.release -> RUN).**

**Конкурентното изпълнение** - няколко процеса могат да се изпълняват едновременно или чрез редуване на техните действия.

**Интеракция между процесите:** Процесите могат да си взаимодействат чрез **споделени действия**, като това взаимодействие трябва да бъде контролирано.

**Паралелна композиция на асинхронни процеси:** При паралелната композиция действията на различни процеси могат да се **редуват (interleaving)**, като всеки процес запазва своя ред на изпълнение.

**Интеракция чрез споделени действия:** Процесите могат да синхронизират действията си чрез **споделени събития**.

**Етикетиране и преименуване на действия:** Действията на процесите могат да бъдат **етикетирани** или **скрити**, за да се моделират различни аспекти на взаимодействието.

**Структурни диаграми:** Използват се за графично представяне на паралелната композиция и взаимодействията между процесите.

**Многонитови програми в Java:** В Java конкурентността се реализира чрез **нишки (threads)**, които могат да се изпълняват паралелно или чрез редуване на действията им.

**Едновременност (Concurrency)** представлява **логическа** обработка, която **протича сякаш се изпълнява едновременно**. Това не означава, че са необходими множество процесорни елементи (PEs). За да се реализира едновременност, е нужно редуващо се изпълнение на задачите върху един процесорен елемент (PE).

**Паралелност (Parallelism)** е физическа едновременна обработка. Тя включва множество процесорни елементи (PEs) и/или независими операции на устройства.

**И едновременността, и паралелността** изискват контролирано използване на споделени ресурси. Често използваме термините "паралелно" и "едновременно" взаимозаменяемо и като цяло не правим разлика между реално и псевдо-едновременно изпълнение.

Моделиране на едновременност като:

- моделираме **произволна скорост** на изпълнение на процесите – абстрахираме се от времето и не го вземаме предвид директно;
- **произволен относителен ред на действията** от различни процеси. Това означава, че използваме **редуване (interleaving)**, но съхраняваме реда на действията във всеки отделен процес
- Получаваме общ модел, който е независим от графика на изпълнение- **асинхронен модел на изпълнение**, който не предполага точно време за всяко действие.

**Паралелна композиция и редуване на действия**

Ако  $P$  и  $Q$  са процеси, тогава  $(P \parallel Q)$  представлява тяхното конкурентно изпълнение.  $\parallel$  е операторът за паралелна композиция

**ITCH = (scratch->STOP).**

**CONVERSE = (think->talk->STOP).**

**$\parallel$ CONVERSE\_ITCH = (ITCH  $\parallel$  CONVERSE).**

-Несвързани азбуки (Disjoint alphabets)

**Конкурентност:** редуването на действия в споделена среда.

**Паралелна композиция:** действията на два процеса могат да се редуват по произволен начин.

**Нишките в Java-**мощен инструмент за **конкурентно програмиране**.

**Композитни (Съставните) процеси** се дефинират като *Декартово произведение на примитивни процеси* т.е. те обединяват всички възм. състояния и действия на примит. процеси, които ги изграждат.

**Примитивните процеси** се дефинират с помощта на префиксиране на действия (**action prefix**) и избор (**choice**) - описване последователност от действия и възможности за разклоняване изпъл.

**Съставните процеси** се изграждат чрез **паралелна композиция на примитивни процеси** т.е. множество процеси могат да работят едновременно, като споделят ресурси и взаимодействат помежду си.

Моделите, описани чрез *FSP* са **крайни** - те имат ограничен брой състояния и преходи, което улеснява тяхното анализиране и проверка.

**Алгебрични закони за паралелната композиция:**

**Комуникативност (Commutative):**  $(P \parallel Q) = (Q \parallel P)$  - Редът на процесите в паралелната композиция **няма значение**.

**Асоциативност (Associative):**  $(P \parallel (Q \parallel R)) = ((P \parallel Q) \parallel R) = (P \parallel Q \parallel R)$

Ако процесите в една композиция имат общи действия, тези действия се наричат **споделени действия**. Те са начинът, по който се моделира взаимодействието между процесите.

$BILL = (play \rightarrow meet \rightarrow STOP).$

$BEN = (work \rightarrow meet \rightarrow STOP).$

$\parallel BILL\_BEN = (BILL \parallel BEN).$

**Несподелени действия** могат да бъдат изпълнявани произволно и независимо едно от друго (редуване или interleaving).

**Споделените действия**, от друга страна, **трябва да се изпълняват едновременно** от всички процеси, които участват в тях. Споделените действия **гарантират синхронизация между процесите**. Те изискват **всички участващи процеси да изпълнят конкретното действие едновременно**, което осигурява координация и предсказуемо поведение при взаимодействие. Non-disjoint alphabets **ръкостискане (handshake)** - действие, което се потвърждава или признава от друг процес. Това означава, че за дадено взаимодействие е необходимо и двата процеса (или повече) да участват активно в изпълнението на конкретното действие. **ограничава общото поведение** на системата, като изисква синхронизация между участващите процеси. Гарантира, че действието ще бъде извършено само ако всички участници го потвърдят и изпълнят едновременно.

Моделиране на взаимодействие – множество процеси (многостранна синхронизация) - взаимодействие, при което **няколко процеса участват едновременно в изпълнението на едно действие** - синхронизацията не се ограничава само до два процеса, а може да включва многостранно участие. Всички участващи процеси трябва да бъдат синхронизирани, за да изпълнят съвместното действие едновременно. Ако дори един от процесите не е готов за изпълнение, действието не може да се осъществи.

**||MAKERS = (MAKE\_A || MAKE\_B).**

**||FACTORY = (MAKERS || ASSEMBLE).**

Когато заменим дефиницията на един съставен процес (например MAKERS) в дефиницията на друг процес (например FACTORY), и приложим комутативните и асоциативните закони за паралелна композиция, получаваме оригиналната дефиниция на FACTORY, изразена в термини на примитивни процеси. Тази техника позволява:

**Процесни инстанции:** Когато говорим за процеси в контекста на моделиране, като например с помощта на **процесна алгебра** (Process Algebra), "процесни инстанции" са отделни екземпляри на даден процес. Всеки екземпляр е независим и може да следва своето собствено изпълнение според зададеното поведение.

**Етикетиране (Labeling):** Етикетирането е начин да разграничим действията на различни екземпляри на процеса. В случая с етикета a:P, всяко действие от азбуката на процеса P е предхождано от префикс a. Това е полезно, когато трябва да разграничим действията на различни процеси или инстанции.

Процесът **SWITCH** е дефиниран така:

$$\text{SWITCH} = (\text{on} \rightarrow \text{off} \rightarrow \text{SWITCH})$$

Това означава, че процесът **SWITCH** се състои от последователни действия: първо on, след това off, и после се връща към началното състояние, създавайки цикличен модел. Когато имаме две инстанции на този процес, те могат да бъдат обозначени с различни префикси:

$$\text{TWO\_SWITCH} = (\text{a:SWITCH} \parallel \text{b:SWITCH})$$

Тук  $\parallel$  означава **паралелна композиция**, което означава, че двата процеса (инстанции) работят независимо един от друг.

Можем да създадем масив от  $N$  инстанции на процеса **SWITCH**, като ги обозначим с индекси:

$\text{SWITCHES}(N=3) = (\text{forall}[i:1..N] \text{ } s[i]:\text{SWITCH})$  или по-кратко:

$\text{SWITCHES}(N=3) = (s[i:1..N]:\text{SWITCH})$

Това означава, че имаме  $N$  независими екземпляра на процеса **SWITCH**, всеки със собствен индекс.

The use of  $s$  in the expression  $s[i]:\text{SWITCH}$  or  $\text{SWITCHES}(N=3)$  is simply a **naming convention** to represent a **set of processes or instances** of the process **SWITCH**.

**Процесно етикетиране с множество от префикси** -предоставя начин за ясно разграничаване на действията на процеси, които споделят общ ресурс или работят заедно.

Когато използваме множество префикси  $\{a_1, \dots, a_x\}::P$ , всяко действие  $n$  в азбуката на процеса  $P$  се заменя с етикетите  $\{a_1.n, \dots, a_x.n\}$ . Това означава, че:

1. **Всеки етикет на действие  $n$**  се предшества от всички префикси в множеството  $\{a_1, \dots, a_x\}$ .
2. **Преходите в процеса  $P$**  се адаптират да отразяват новите етикети.

**Преходите на процеса:** Ако имаме преход в процеса  $P$  с вида:

$n \rightarrow X$ , този преход се заменя със:  $\{a_1.n, \dots, a_x.n\} \rightarrow X$ .

Това означава, че всяко действие  $n$  в процеса  $P$  може да бъде "виждано" с разл. етикети, в зависимост от префиксите, които са приложени.

**Преименуването на действия(Action Relabeling)** е техника, която се използва за промяна на имената на етикетите на действията в даден процес. Това е полезно при моделиране на системи, където е нужно процесите да **синхронизират** действията си върху определени етикети или за по-ясно разграничаване на техните роли.

$/\{\text{newlabel\_1}/\text{oldlabel\_1}, \dots, \text{newlabel\_n}/\text{oldlabel\_n}\}$

- `oldlabel` е оригиналният етикет на действието в процеса.
- `newlabel` е новият етикет, който ще замени стария.

Една от основните цели на преименуването е да се гарантира, че процесите **синхронизират върху определени действия**. Това означава, че действията, които преди са имали различни етикети, могат да бъдат унифицирани, за да позволят взаимодействие между процесите.

### **Преименуване на действия – Префиксни етикети**

**Преименуването на действия с префиксни етикети** позволява действията в един процес да бъдат пренасочени или унифицирани с действия в друг процес, като се използват **общи префикси**. Това е удобно за моделиране на системи, в които процеси като клиент и сървър трябва да синхронизират действията си, дори ако първоначално използват различни етикети.

`SERVERv2 = (accept.request ->service->accept.reply->SERVERv2).`

`CLIENTv2 = (call.request ->call.reply->continue->CLIENTv2).`

`||CLIENT_SERVERv2 = (CLIENTv2 || SERVERv2) /\{call/accept\}.`



## Скриване на действия – Абстракция за намаляване на сложността

**Скриването на действия** е техника, която се използва за намаляване на сложността при моделирането на процеси. Тя позволява определени действия да бъдат скрити, така че те да не са видими извън даден процес. Скрытите действия стават "тихи" и се обозначават със специалния етикет  $\tau$  (**тау**).

### Оператор за скриване $\{a1..ax\}$

Когато операторът за скриване  $\{a1..ax\}$  се приложи върху процес P:

- Етикетите на действията  $a1..ax$  се премахват от азбуката на процеса.
- Скрытите действия се обозначават като **тихи действия** ( $\tau$ ).
- Тихите действия **не се споделят** между различни процеси.

Нека имаме процес  $P=(a \rightarrow b \rightarrow P)$  Ако приложим скриване върху a:  $P \setminus \{a\}=(\tau \rightarrow b \rightarrow P)$  Действието a става скрито и се обозначава с  $\tau$ , докато b остава видимо. Еквивалентни са:

**USER = (acquire  $\rightarrow$  use  $\rightarrow$  release  $\rightarrow$  USER)**

**$\setminus \{use\}$ .     или**

**USER = (acquire  $\rightarrow$  use  $\rightarrow$  release  $\rightarrow$  USER)**

**@{acquire,release}.**

**Оператор за интерфейс @:** @{ acquire, release } ограничава наблюдаваните действия на процеса **USER**. Само действията с етикети acquire,release ще бъдат видими. Всички останали действия

в азбуката на USER ще бъдат скрити и ще се обозначават като тихи действия ( $\tau$ ).

Минимализация - Премахване на излишни скрити  $\tau$  действия- Минимализацията гарантира, че външен наблюдател няма да забележи разлика в поведението на с-мата след премахването на  $\tau$ .

RESOURCE = (acquire->release->RESOURCE).

USER = (printer.acquire->use ->printer.release->USER)\{use}.

||PRINTER\_SHARE= (a:USER||b:USER|| {a,b}::printer:RESOURCE).

{a,b}::printer:RESOURCE - дефинира споделен ресурс между множество процеси, като се уточнява как те взаимодействат с него.

| може да означава "ИЛИ" (логическа операция "или", дизюнкция)

### **Едновременни процеси и взаимодействие между процесите**

- **Едновременни процеси (Concurrent processes):**  
Това са процеси, които се изпълняват **едновременно**, като всеки от тях има собствено поведение, но може да взаимодейства с други процеси. Пример за това са паралелни нишки (threads) в програмирането.
- **Взаимодействие между процесите (Process interaction):**  
Това се осъществява чрез **споделени действия**, където два или повече процеса могат да изпълняват едно и също действие, което ги синхронизира.

**Асинхронен модел (Asynchronous Model):** Процесите се изпълняват с **произволна скорост**, без необходимост да бъдат синхронизирани по време. Асинхронното изпълнение означава,

че няма предварително определена последователност между действията на процесите.

**Интерливинг (Interleaving):** Процесите се разглеждат като последователно подреждане на техн. действия. Макар че процесите изглеждат като че ли се изпълняват паралелно, техните действия се **редуват в произволен ред**.

**Паралелна композиция (Parallel composition):** Два или повече процеса работят паралелно, като техн. действия могат да се **редуват (interleave)** или да се **синхронизират** чрез споделени действия. Паралелната композиция може да се разглежда като **крайно състояние (finite state process)**, което комбинира поведението на участващите процеси.

**Взаимодействие чрез споделени действия:** Процесите могат да комуникират и синхронизират чрез **споделени действия**. Споделените действия се изпълняват само когато всички участващи процеси са готови да ги извършат.

**Маркиране на процеси (Process labeling):** Всеки процес получава **префикс** (например а: или b:), за да се разграничават неговите действия.

**Преименуване на действия (Action relabeling):** Използва се функция за преименуване, за да се сменят етикетите на действията в процесите:  $/\{\text{нов\_етикет1}/\text{стар\_етикет1}\}$

**Скриване на действия (Action hiding):** Някои действия се премахват от видимия интерфейс на процеса и се обозначават като **"тихи действия" (tau)**.