Операционни системи за реално време

Автор: гл. ас. д-р инж. Любомир Богданов

Европейски съюз

ПРОЕКТ BG051PO001--4.3.04-0042
„Организационна и технологична инфраструктура за учене през целия живот и развитие на компетенции”
Проектът се осъществява с финансовата подкрепа на
Оперативна програма „Развитие на човешките ресурси”,
съфинансирана от Европейския социален фонд на Европейския съюз
Инвестира във вашето бъдеще!

ЕСФ България
Европейски социален фонд

Съдържание

1. Схеми за диспечериране на задачите (scheduling policies)
2. Комуникационни примитиви
3. Синхронизационни примитиви
4. Линукс за вградени системи
5. Дървесни двоични описания (device tree binaries)

Схеми за диспечериране на задачите

Операционна система за реално време (**R**eal-**T**ime **O**perating **S**ystem, RTOS) – фърмуер, с помощта на който може да се изпълняват повече от една “main” функции на един микропроцесор и който осигурява стандартни библиотеки, характерни за по-мощни системи (като персонални компютри).

Ако в микроконтролера е вграден един микропроцесор, изпълнението на “main” програмите е **псевдопаралелно**.

Процеси (или още задачи, process, task) - програми, които се изпълняват в паралел от RTOS.

Нишки (threads) – програми, които се стартират от един процес и работят (псевдо)паралелно във виртуалното адресно поле /ако има MMU/ на същия този процес.

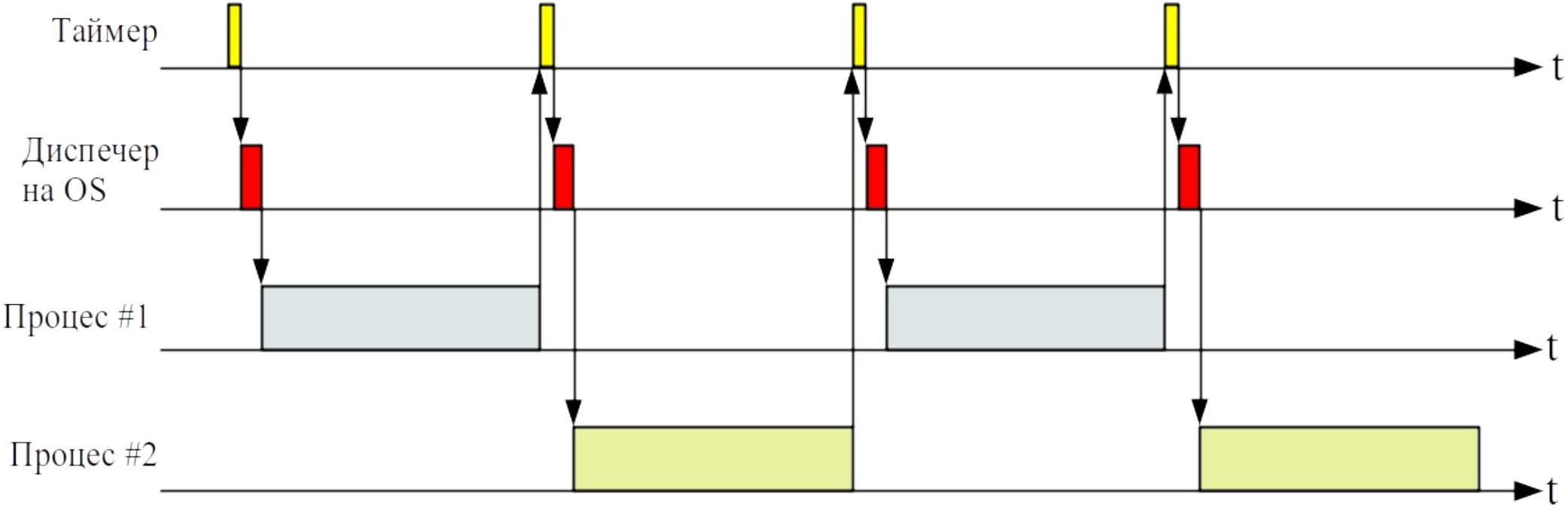
Схеми за диспечериране на задачите

В микроконтролери без MMU понятието процес и нишка съвпадат.

Диспечер на операционната система (scheduler) – част от кода на RTOS, която е отговорна за даване на процесорно време на всеки един процес. Действието на превключване от един процес към друг се нарича **контекстно превключване** (context switch). Кодът на диспечера се изпълнява от същия микропроцесор, който изпълнява процесите.
Да не се бърка с диспечер на инструкцията, който е хардуерен модул от μPU ядро.

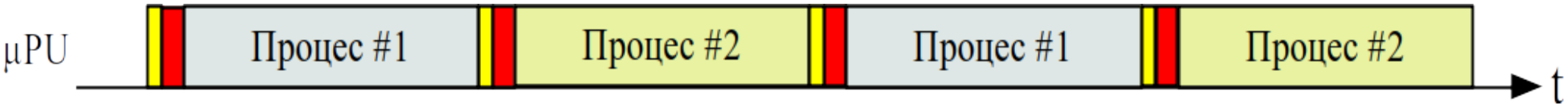
Квантовете от време се задават от един хардуерен таймер. Прекъсванията му прехвърлят процесорното време към диспечера.

Схеми за диспечериране на задачите

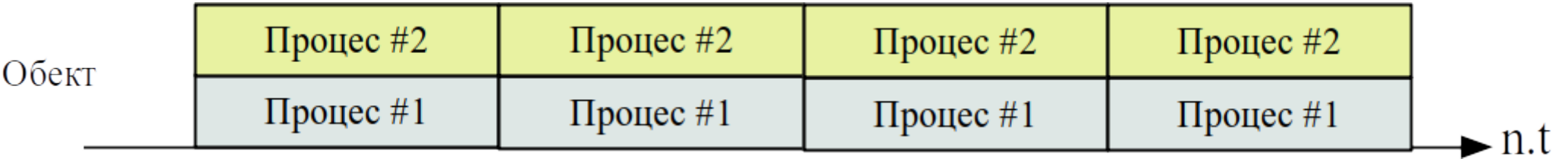


Схеми за диспечериране на задачите

Ето какво “вижда” μPU:



Ето какво “вижда” управляваният обект (вижте времевата база – n.t):



или с други думи, ако диспечерът превключва управлението на процесите много бързо, така че управляваният обект да не забележи, за обекта ще е еквивалентно все едно два микропроцесора изпълняват две програми – процес 1 и процес 2.

Схеми за диспечериране на задачите

Въпросът е – колко бързо трябва да става това превключване (time slice)?

Схеми за диспечериране на задачите

Отговорът е – зависи.

Зависи от управлявания обект. Може да е:
*x100 ns
*(x1 ÷ x100) μs
*(x1 ÷ x100) ms
*(x1 ÷ x100) s

За голяма част от некритичните приложения 10 ÷ 100 ms е достатъчно време.

Реално време – означава, че процесите трябва да завършат дадена операция в рамките на предварително известно време.

Схеми за диспечериране на задачите

Операционна система за критично реално време (hard real-time operating system) – ако дадената операция не завърши в даден отрязък от време, ще има катастрофални последници (влак ще спре да се движи, кола ще спре да завива, кран ще зависне и т.н.).

Операционна система за некритично реално време (soft real-time operating system) - ако дадената операция не завърши в даден отрязък от време, няма да има катастрофални последници (ще излезе син екран на кафе машината, ще се рестартира графичния интерфейс на пералнята, ще се забави изпращането на Интернет пакет и т.н.).

Схеми за диспечериране на задачите

Процесите минават през три етапа по време на изпълнението си:

***блокирана** (wait) – процесът не се изпълнява от диспечера, защото чака някакво условие да се изпълни;

***изпълнява се** (run) – процесът не блокирал и се изпълнява от диспечера в неговия си времеви квант;

***готова за изпълнение** (ready) – процесът не е блокирал, но чака разрешение от диспечера, за да продължи да се изпълнява.

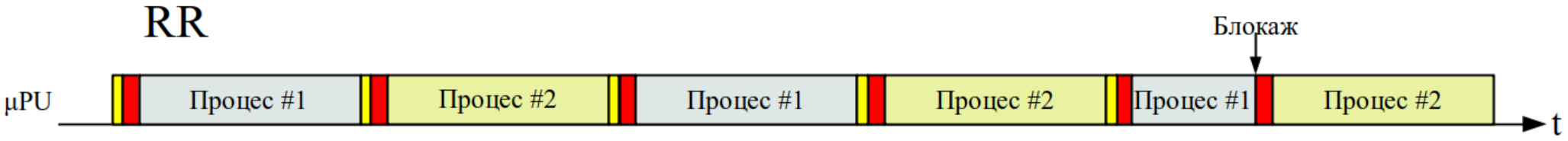
Схеми за диспечериране на задачите

Схема за диспечериране на задачите (scheduling policy) – политика, която указва кога диспечера да даде микропроцесорно време на даден процес.

Най-често използваните схеми са:
*round-robin (RR)
*preemptive (PRE)
*round-robin/preemptive (RR/PRE)
*cooperative (COOP)

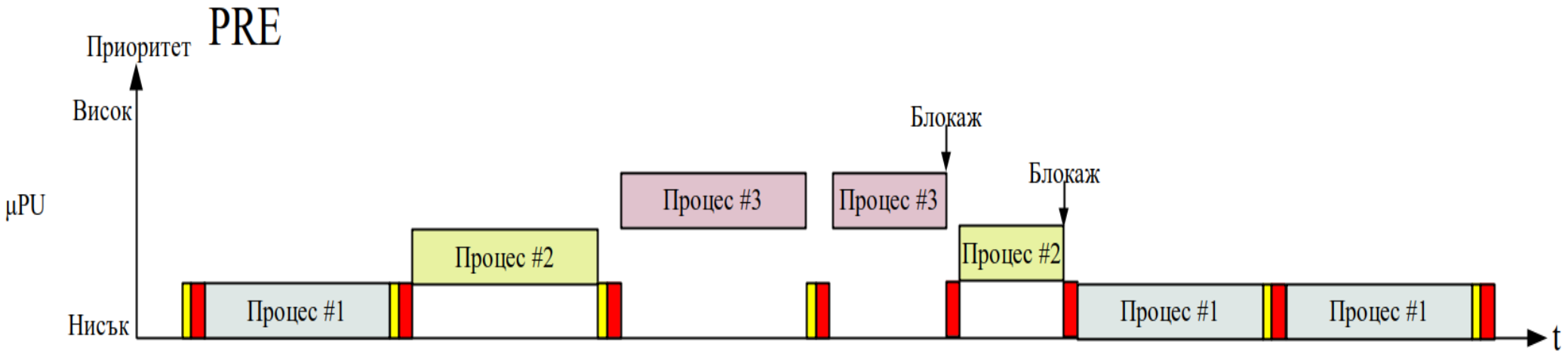
Схеми за диспечериране на задачите

Round-robin диспечериране (RR) – всички процеси са с един приоритет и получават равни квантове време за изпълнение. Ако някой от процесите блокира, изпълнението се предава на следващата поред.



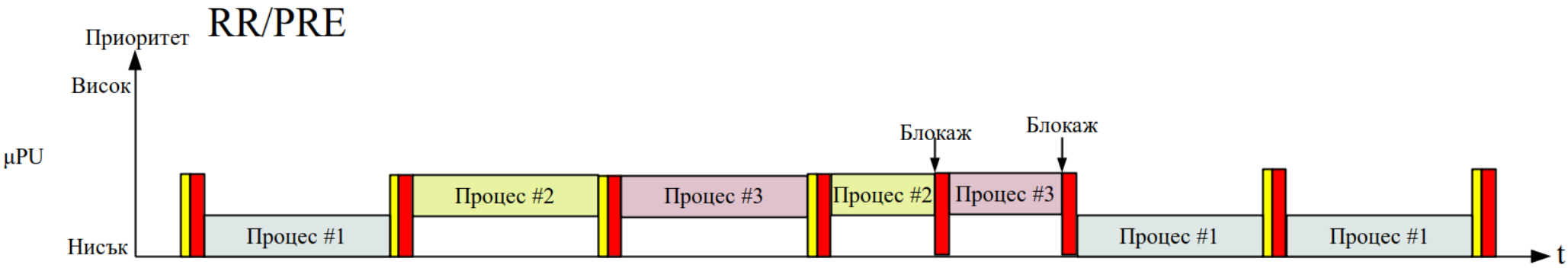
Схеми за диспечериране на задачите

Preemptive диспечериране (PRE) – на всеки процес се дава уникален приоритет. Не може да има два процеса с един и същи приоритет. Процеси с по-високи приоритети могат да прекъсват (preempt) процеси с по-ниски приоритети. Може да доведе до “приоритетен глад” (priority starvation), където процесите с ниски приоритети почти не се изпълняват заради процеси с по-високи приоритети.



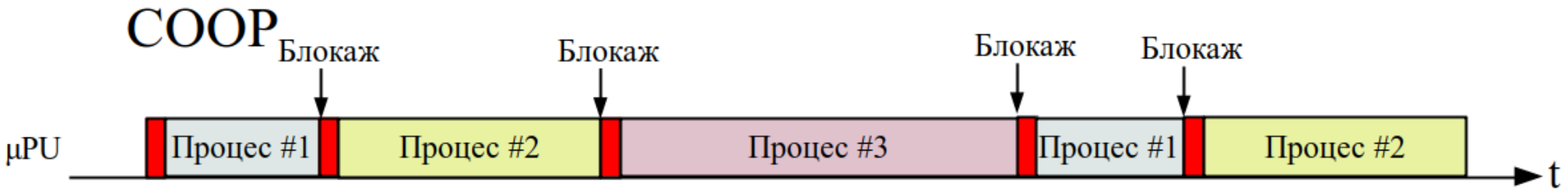
Схеми за диспечериране на задачите

Round-robin/Preemptive диспечериране (RR/PRE) – на всеки процес се дава приоритет. Може да има два процеса с един и същи приоритет. Процеси с различни приоритети се изпълняват по Preemptive схемата. Процесите с еднакви приоритети се изпълняват по RR схемата.



Схеми за диспечериране на задачите

Cooperative диспечериране (COOP) – всички процеси са с един и същ приоритет. Не се използва системен таймер, т.е. превключването от един процес в друг става без диспечер. Разчита се, че всеки един процес ще пуска следващия като се само блокира “доброволно”.



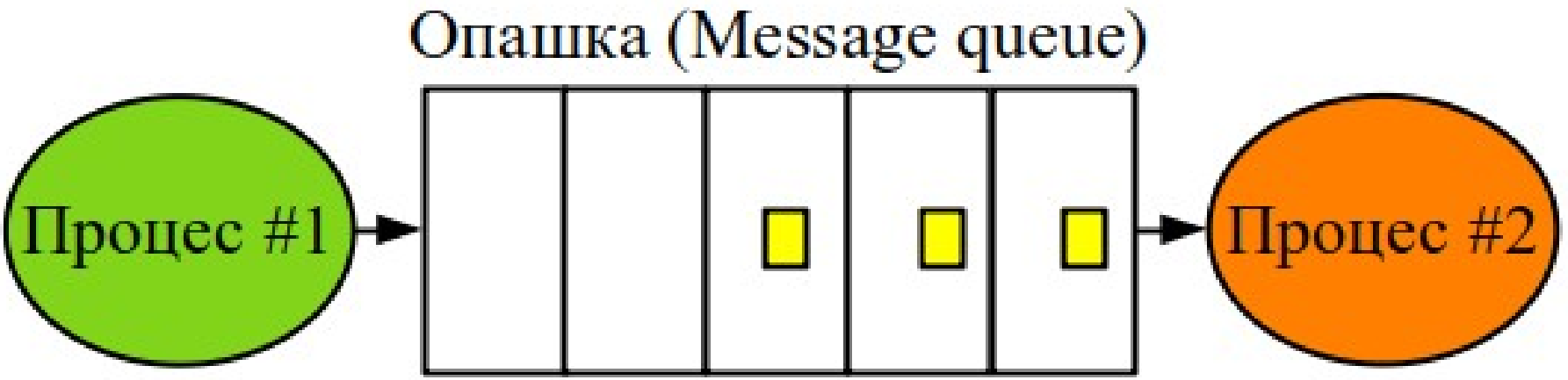
Комуникационни примитиви

Комуникационни примитиви (interthread communication) – променливи, които се използват за предаване на данни от един процес на друг под управлението на ядрото на RTOS.

Най-често използваните ком. примитиви са:
*опашки
*поща

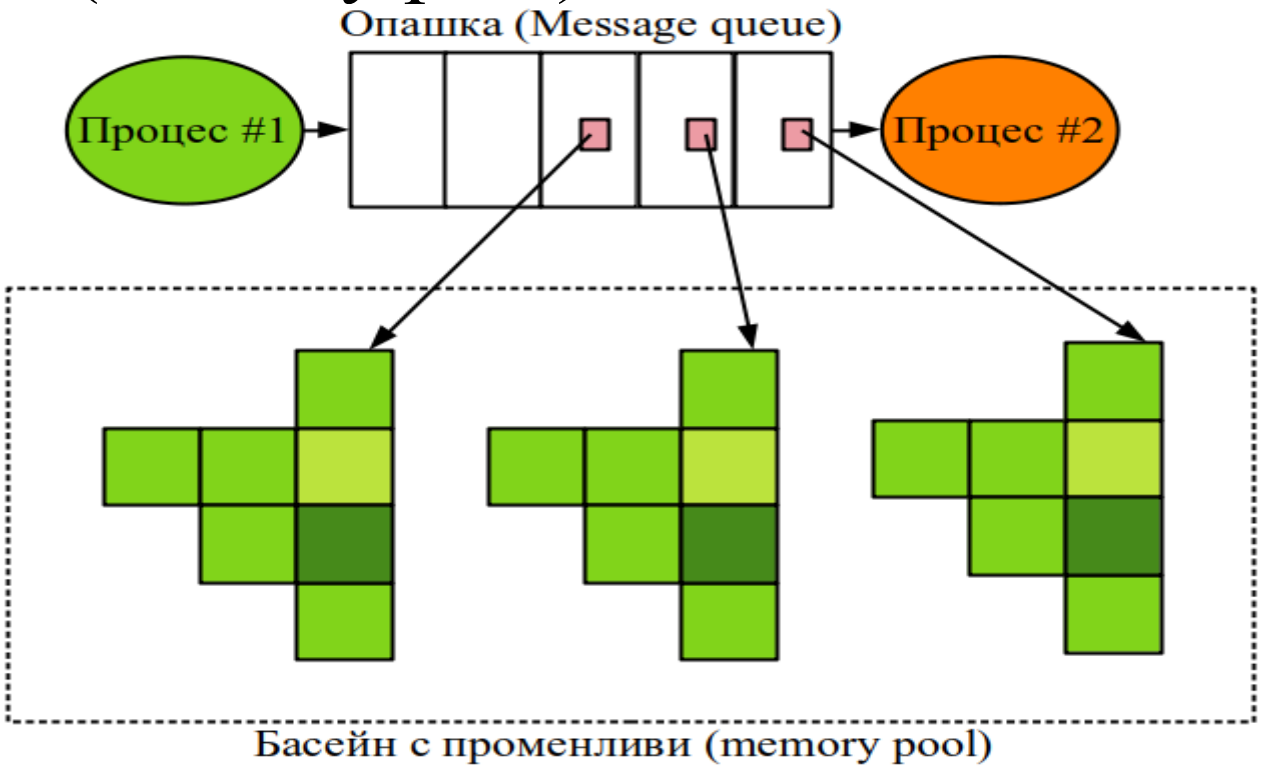
Комуникационни примитиви

Опашки (message queue) – софтуерни FIFO буфери, които съдържат скаларни величини (int, float, double, char, и т.н.). Те се създават, четат и записват с API функции на ядрото на RTOS [1].



Комуникационни примитиви

Поща (mail queue) – софтуерни FIFO буфери, които съдържат указатели към динамично заделени структури. Те се създават, четат и записват с API функции на ядрото на RTOS [1]. В този случай динамично заделената памет се нарича басейн с променливи (memory pool).



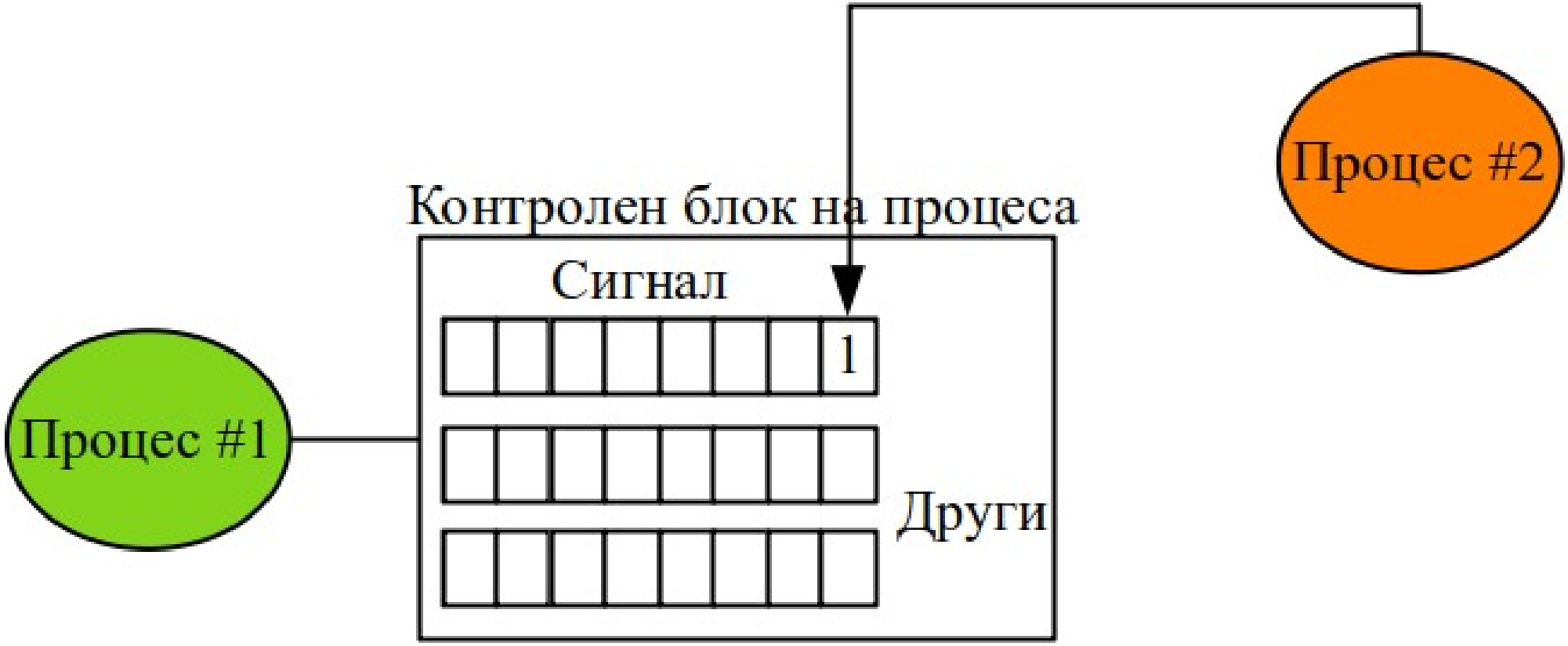
Синхронизационни примитиви

Синхронизационни примитиви (synchronization primitives) променливи, които се използват за синхронизиране изпълнението на един процес спрямо друг под управлението на ядрото на RTOS [1], [2].

Най-често се използват:
*сигнали
*семафори
*мютекси (двоични семафори)
*рандеву
*бариера

Синхронизационни примитиви

Сигнал (signal) – променлива, която се намира в структурата, която описва процеса. Ако друг процес я достъпи, може да променя битовете на променливата и да я постави принудително в режим “блокиран” (wait).

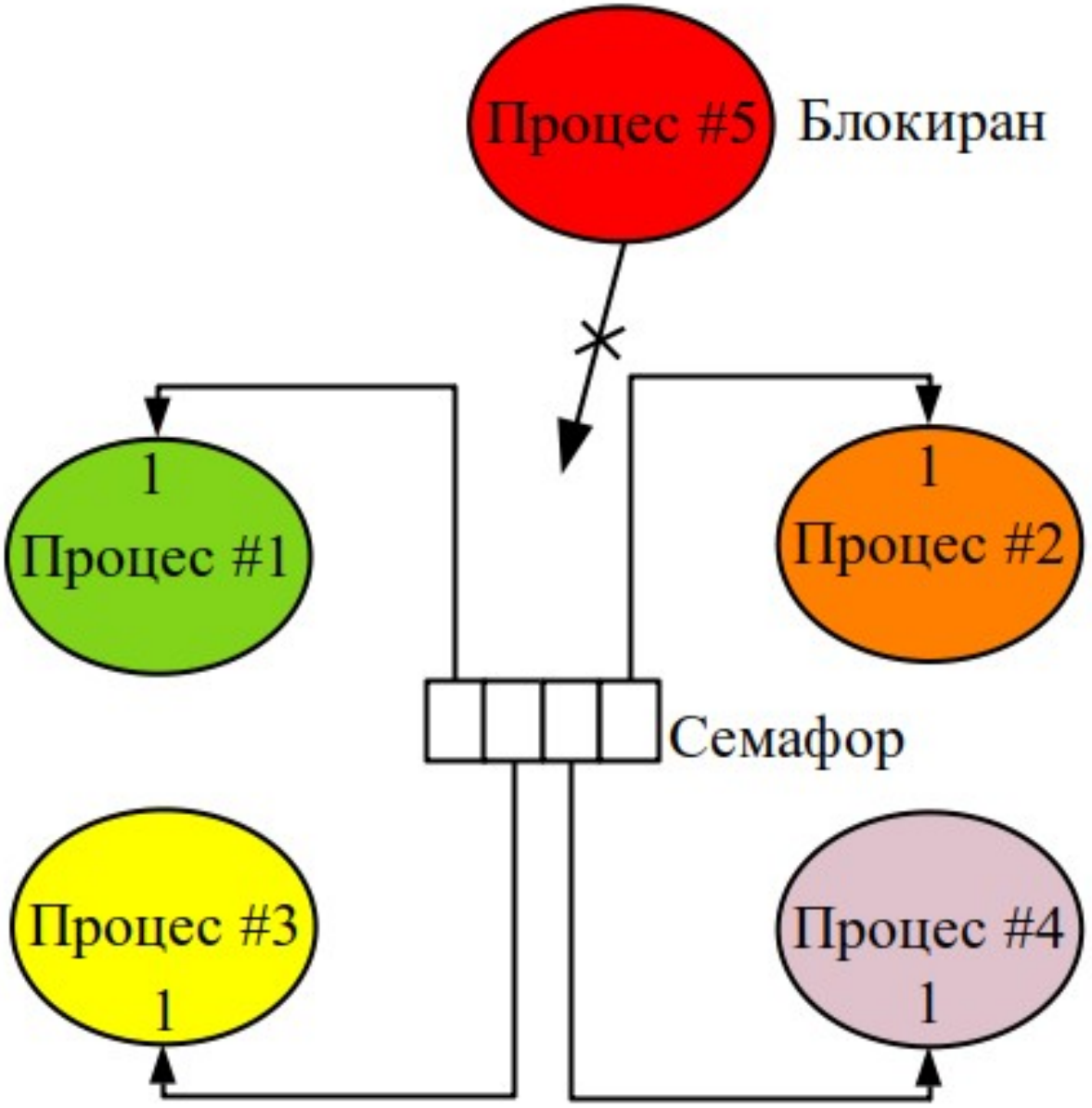


Синхронизационни примитиви

Семафор (semaphore) – променлива, която се зарежда с число от програмиста. Когато един процес се изпълнява, в тялото на кода му може да се извика функция, която достъпва тази променлива. Ако тя не е нула, процесът ще продължи напред. Ако тя е нула, процесът ще блокира, докато някоя друга нишка не върне число обратно в семафора.

Използва се, за да се контролира достъпа до ограничени ресурси. Например – 4 портова SRAM памет ще може да се достъпи от максимум 4 процеса наведнъж. Ако се появи 5-ти процес, той трябва да бъде блокиран, докато някой от другите не спре да използва паметта.

Синхронизационни примитиви

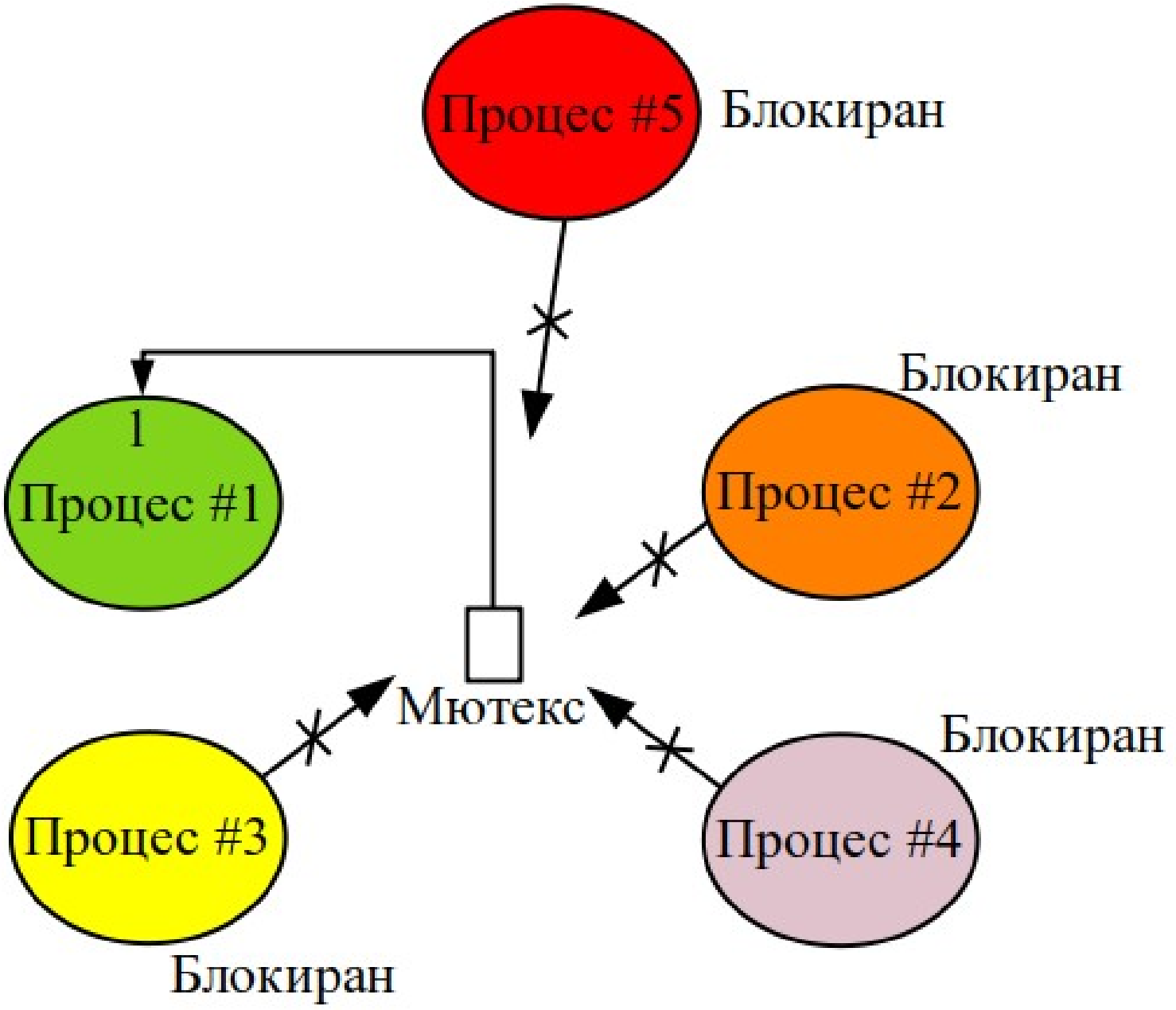


Синхронизационни примитиви

Мютекс (mutex) – променлива, която се зарежда с число, което може да е само 0 или 1, от програмиста. Действието му е аналогично на семафора.

Използва се, за да се контролира достъпа до едноканални/еднопортови ресурси. Например – UART интерфейса може да изпраща съобщение само от един процес, защото има само един сигнал TxD и изпраща данните серийно.

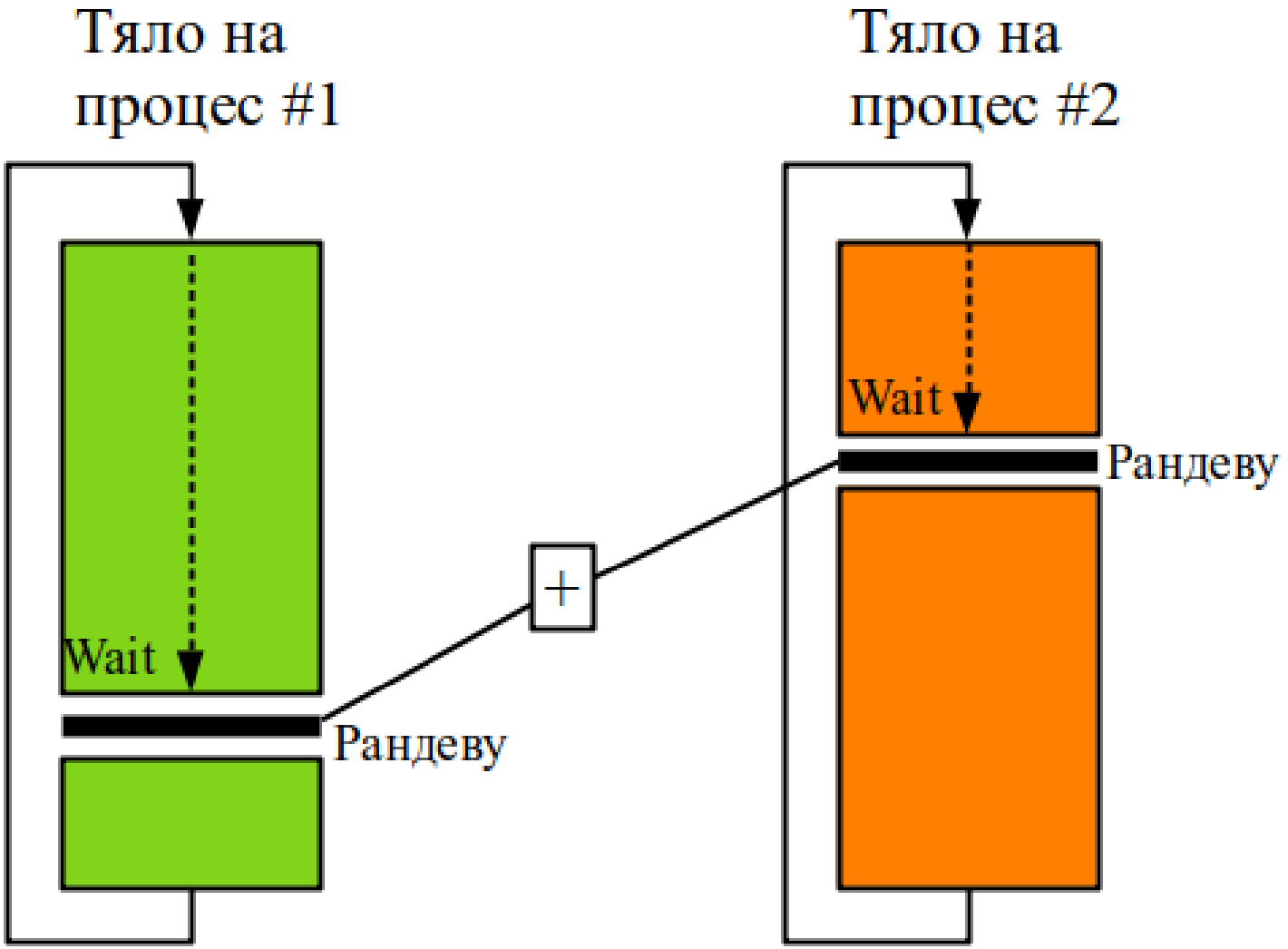
Синхронизационни примитиви



Синхронизационни примитиви

Рандеву (rendezvous) – променлива, която гарантира, че два процеса ще се синхронизират и ще продължат изпълнението на дадени части от кода си едновременно. Може да се направи аналогия с двувходово И – кодът на процес 1 ще продължи изпълнението си от адрес N, когато процес 2 стигне до адрес M от кода си.

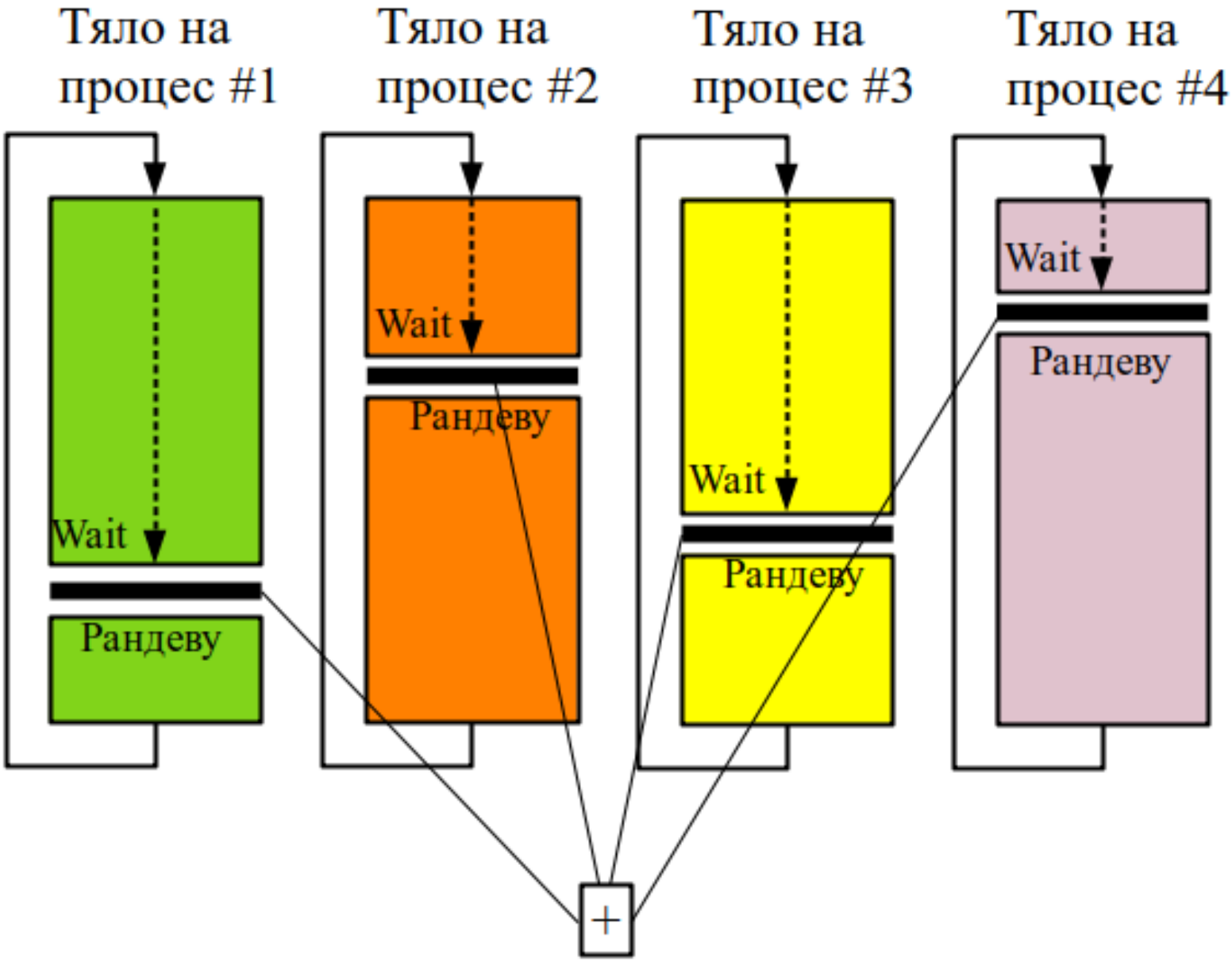
Синхронизационни примитиви



Синхронизационни примитиви

Бариера (barrier turnstile) – променлива, която е по-генерализиран вариант на рандевуто и гарантира, че два или повече процеса ще се синхронизират и ще продължат изпълнението на дадени части от кода си едновременно. Може да се направи аналогия с многовходово И – кодът на процес 1 ще продължи изпълнението си от адрес N, когато процес 2 стигне до адрес M от кода си, и процес 3 до адрес P, и процес 4 до адрес Q, и т.н.

Синхронизационни примитиви



Литература

[1] Trevor Martin, „The Designer's Guide to the Cortex-M Processor Family – A Tutorial Approach“, Elsevier, 2013.
[2] https://www.cs.columbia.edu/~hgs/os/sync.html