1. Структури от данни за комуникация между задачите

Заданията, работещи под управлението на операционната система за реално време, имат нужда да обменят данни помежду си, тъй като често са част от един конвейер за обработка на потока данни.

За целта системното програмно осигуряване трябва да осигури средствата за обмен на информация между заданията.

1.1. Семафор

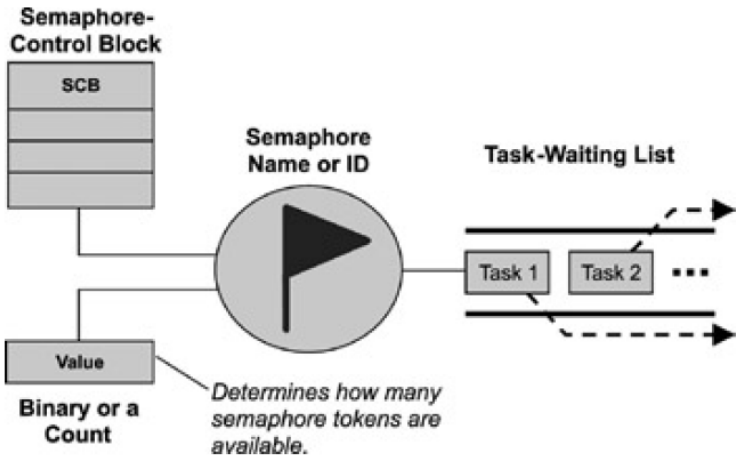
Семафорът е обект на ядрото, който една или повече задачи могат да изискат или да освободят, с цел:

- синхронизация или
- взаимно изключване.

Елементите на семафорите са:

- Контролен блок – Semaphore Control Block (SCB). Това е структурата, в която се пазят елементите, определящи еднозначно дадения семафор;
- Идентификатор – ID;
- Стойност – бинарна или целочислена;
- Списък на чакащите задачи.

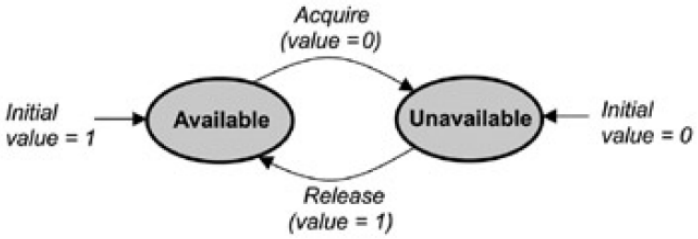
Структура на семафор



Бинарен семафор

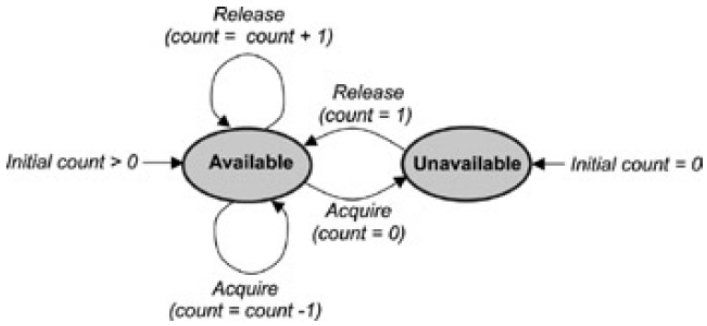
Бинарният семафор има само две състояния.
При стойност 0 семафорът се счита за “зает” (не свободен), а при 1 за “свободен” (“пълен”).

Диаграма на състоянията на бинарен семафор.



Броячен (counting) семафор

Броячният семафор има целочислена стойност и позволява да бъде заеман (респ. освобождаван) множество пъти (в зависимост от стойността).
При стойност 0 семафорът се счита за “зает” (не свободен), а при 1 и по-голяма от 1 за “свободен” (“пълен”).
Диаграма на състоянията на броячния семафор.



Семафор мютекс (**Mut**ex – **Mut**ual **Ex**clusion – Взаимно изключване)

Мютекс е един от вариантите на семафорните механизми за организация на взаимно изключване.
Мютекса е двоичен семафор, който може да се намира в едно от двете състояния: **открит** (**отключен**) или **закрит** (**заключен**). Когато дадена задача заеме обекта на мютекса, той преминава в закрито състояние. Ако задачата освободи мютекса, той преминава в открито състояние.
Задачата на мютекса е защитата от достъп до обекта на други задачи преди мютекса да бъде **отключен**. В даден момент само една задача може да ползва обекта, който е защитен с мютекса. Ако на друга задача е нужен достъп до този обект, то тя трябва да изчака, докато мютекса не бъде освободен.
Целта при използването на мютекса е защита от повреждане на данните при асинхронно им изменение или взаимна блокировка на задачите.
Рекурсивното заключване е механизъм, при който една задача може многократно да изисква mutex върху, който има собственост. Mutex, който поддържа това свойство се нарича **рекурсивен mutex**. Пример за ползата от това рекурсивно заключване е когато няколко функции от една и съща задача изискват съответния mutex.
Защитата от изтриване на задачата е механизъм, който не позволява задачата, която е собственик на mutex-а да бъде преждевременно изтрита (преди да го е освободила).

Типичните операции със семафори

➢ **Създаване и изтриване на семафорите.**

В зависимост от подържаните от ядрото на RTOS семафори се създават различни обекти и връзки между тях.

- **Binary –** определя състоянието на семафора и създава task-waiting order;
- **Counting** – задава броя и състоянието на семафора и създава task-waiting order;
- **Mutex -** създава task-waiting order и защитата от неправомерно изтриване на задачата, притежаваща в даен момент mutex-а.

Ако семафора ще се използва за синхронизация на две задачи – това ще е двоичен семафор.

Ако семафора ще се използва за съвместно ползване на данни – това ще е мютекс.

Типичните операции със семафори

➢ **Изискване (достъп) и освобождаване на семафорите.**

Операциите по освобождаване и изискване на семафорите съществуват в три разновидности (в зависимост от указания период на изчакване):

- **Безкраен интервал на чакане –** задачата изчаква до момента в който операцията може да се извърши, без значение кога ще настъпи този момент.
- **Изчакване за определен период от време –** задачата остава блокирана до освобождаване на семафора или до изтичането на указан интервал от време. След изтичането на този интервал задачата се премахва от списъка с чакащи задачи и преминава в състояние на “готовност” или “изпълнение”.
- **Без изчакване** – задачата се опитва да изпълни операцията по изискване, но ако достъпът е забранен, тя се отказва и не изпада в блокирано състояние

Типичните операции със семафори

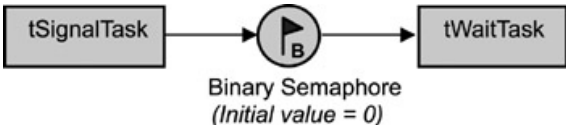
- **Изчистване на списъка с чакащите задачи.**
- **Получаване на информация за състоянието на семафора -** получената информация за състоянието е моментна снимка (аналогично на задачите) и е динамична.

ISR (Процедурите по обработка на прекъсвания) също могат да освобождават бинарни и броячни семафори. Те **НЕ могат** да заключват или отключват мютекси**!**

Употреба на семафорите

- **Изчакване и сигнализиране – Wait-and-Signal**

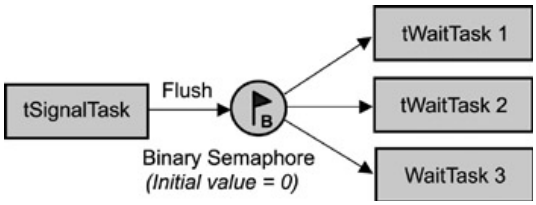
Използва се когато се налага две задачи да си комуникират без да обменят данни помежду си. Например само да се „договорят" за момента на трансфер на контрола.
В този случай се използват бинарни семафори.



Ако приоритета на чакащата задача е по-висок от изпълняваната, след "освобождаването" на семафора тя преминава в състояние "готовност", а след това и в "изпълнение".

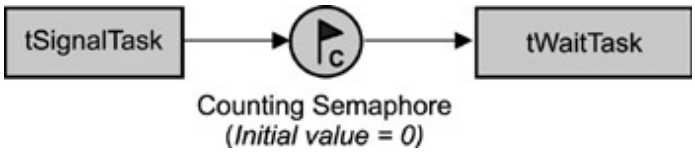
Употреба на семафорите

➢ **Изчакване и сигнализиране на множество от задачи.**
При синхронизиране работата на повече от две задачи се използва **Flush operation** с бинарен семафор над task-waiting list.



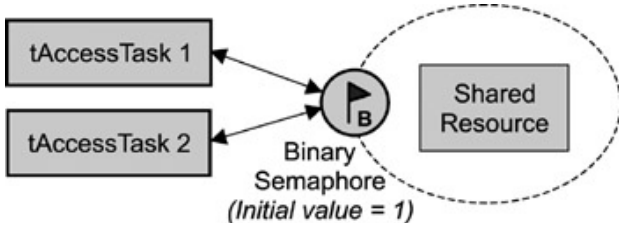
Употреба на семафорите

➢ **Проследяване на събития – Credit-Tracking Synchronization.**
В някои случаи честотата на изпълнение на сигнализиращата задача е по-висока от тази на обработващите задачи. Тогава се използва броячен семафор. Той брои постъпващите сигнали (събития) и изчаква последващата им обработка.



Употреба на семафорите

➢ **Синхронизация на достъпа до единичен споделен ресурс.**

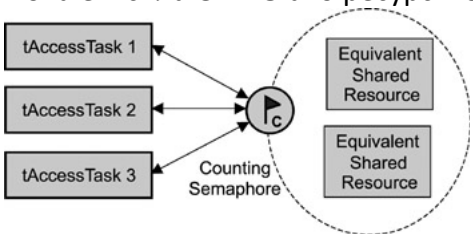


На фигурата е показана реализация на сценария с бинарен семафор. В зависимост от състоянието на ресурса семафора се инициализира с 0 или 1. След това задачите 1 и 2 съответно го изискват или освобождават, при което само една от тях има достъп в момента до ресурса.

Употреба на семафорите

➢ **Синхронизиране достъпа до множество споделени ресурси.**

Това е в сила **САМО** ако ресурсите са идентични.

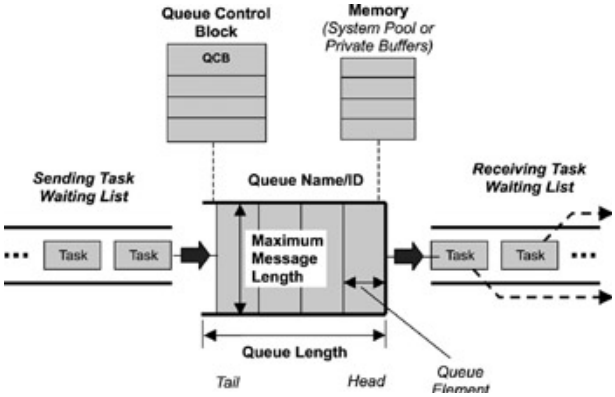


Използва се броячен семафор, чиято стойност е установена на максималния брой на идентичните ресурси. Използването на ресурс е изискване на семафора (т.е. намаляване стойността му), а приключването на работата с ресурс е освобождаване (увеличаване стойността на семафора).

1.2. Опашки със съобщения /Message Queues/

Дефиниране на опашка със съобщения.

Опашка със съобщения е буфер, в който задачите и ISR изпращат и приемат съобщения (данни).

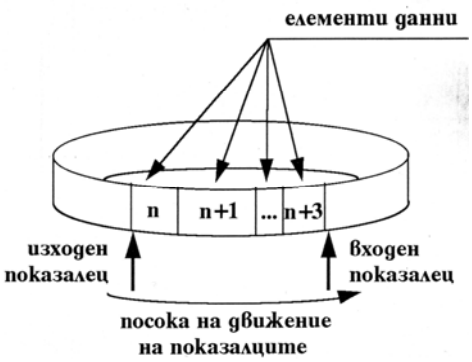


Опашката притежава няколко компонента, които се използват от ядрото. При първоначалното създаване на опашката се инициализират **Queue Control Block**, идентификатор (**Queue Name - ID**), буферна памет, дължина на буфера и чакащите задачи.

1.2. Опашки със съобщения /Message Queues/

Дефиниране на опашка със съобщения.

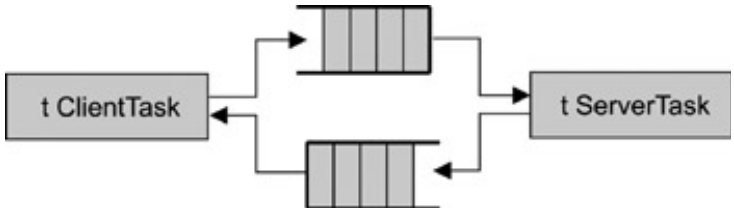
Най-често опашката се управлява като кръгов буфер с указатели – входен и изходен. По своята същност това е един стек FIFO (first in first out - първи влязъл, първи излязъл). Когато опашката е празна входният и изходният указатели имат еднакви стойности.



1.2. Опашки със съобщения /Message Queues/

Дефиниране на опашка със съобщения.

Опашката обикновено реализира връзка между две задания само в една посока - едното задание изпраща съобщения, другото ги чете и обработва. За реализиране на двупосочна връзка са необходими две опашки.



1.2. Опашки със съобщения /Message Queues/

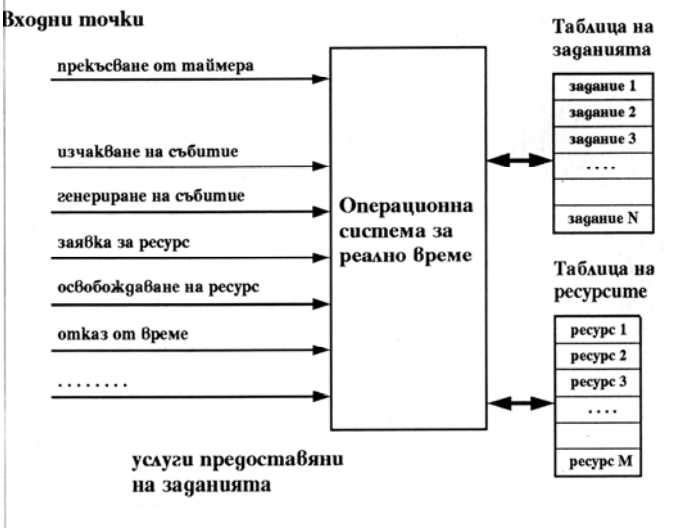
Опашката е важно средство за осигуряване на максимално използване на времето на процесора от заданията и служи като буфер между заданията при случаите на различен темп на постъпване и обработка на съобщенията.

Основните примитиви за работа с опашката са:
- Въвеждане на елемент данни в опашката;
- Извеждане на елемент данни от опашката.

Освен основните примитиви съществуват и **допълнителни**:
- проверка за празна или непразна опашка;
- копиране на първия елемент данни от опашката, като той остава в нея;
- изхвърляне на първия елемент от данни.

2. Функции и реализация на RTOS

Обобщена блокова схема на RTOS

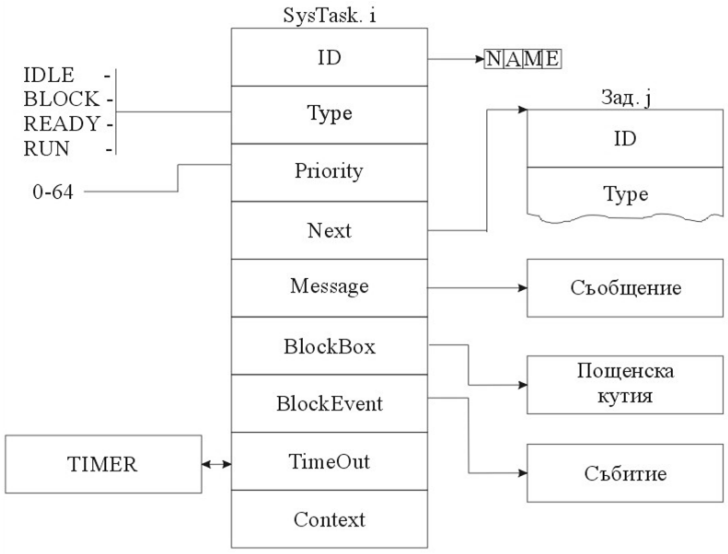


Входни точки

Операционната система за реално време е програма, която управлява разпределението на ресурсите и обслужва заявките от страна на задачите. За тази цел тя трябва да има определени входни точки.

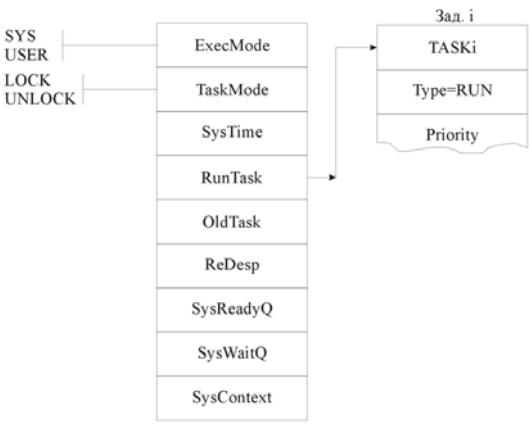
Основна входна точка е обслужването на прекъсването от системния таймер. След всяко прекъсване управлението се предава на тази точка и се извършват дейности по избор на задача, на която да се предостави процесорното време.
Останалите входни точки са заявките за услуги, които предоставя операционната система за реално време - искане на ресурс, генериране на събитие и т.н.
Достъпът до входните точки за заявки може да бъде програмно организиран различно за различните процесори, но най-често става чрез механизма на програмните прекъсвания.

Структурата на дескриптора на задачите



Системен процес

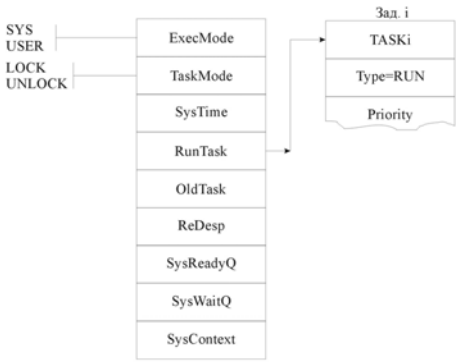
Преди създаването на известен брой приложни задачи в системата се стартира един системен процес, който в последствие поражда приложните задачи.
Той не се преустановява с пораждането на приложните задачи. Той продължава да се изпълнява паралелно с породените от него задачи.
По аналогия с приложните задачи, системния процес притежава също своя системна таблица – дескриптор.



Системният дескриптор (дескриптора на ядрото), за разлика от тези на приложните задачи няма свой собствен идентификатор, поради факта, че е единствен в системата.

Системен процес

Първото поле от системния дескриптор служи за указване на текущия режим на работа. Той може да бъде :
Потребителски (USER) – няма забранени прекъсвания, има предпоставки за диспечеризация на задачите от всички опашки, както и на задачата, която е активна в момента. Тя може да бъде превключена при изтичане на системния квант от време.
Системен (SYS) – в момента се извършват системни манипулации и активния процес е системният.



Полето TaskMode задава режимът на активната задача. Обикновено активната задача е “отключена” (UNLOCK). По този начин механизма на преразпределение на времето между задачите е разрешен. Текущата задача може да бъде заместена от друга при изтичане на системния интервал от време.

Реализация на RTOS

Обикновено операционните системи за реално време се реализират на асемблер за съответния процесор. Все по-често напоследък се срещат и операционни системи за реално време, реализирани на езика С. Той съчетава предимствата на езиците от високо ниво (бързо написване и настройка, кратка и прегледна програма) с предимствата на програмите, написани на асемблер (бързина в изпълнението и малък обем).
Операционните системи за реално време не са големи програми като брой на операторите поради изискването да са бързи и поради ограничения брой услуги, които обикновено предоставят на заданията.

3. Изисквания при избор на RTOS

Изборът на операционна система за реално време има съществено значение за определяне на характеристиките на цялостната система, а именно: нейната функционалност, производителност, надеждност, време за реализация на пазара и цена. Проектирането на системи с висока надеждност е още по-труден и отговорен процес и изисква внимателен избор на операционна система.

- **Бързодействие на системата.**

В приложения с висок коефициент на сигурност е изключително важно да се направи оценка на времевите параметри на системата, т.е. да се определи нейното бързодействие. Следва да се изчисли не само времето за свързване на RTOS и за изпълнение на програмния код на отделните нишки, но и допълнителното време, асоциирано към всяка нишка, а именно времето за превключване на контексти, за изпълнение на системни заявки, времена от прекъсвания и за изпълнение на процедурите за прекъсвания.

3. Изисквания при избор на RTOS

- **Защита на паметта.**

Толерансът към възникване на грешки, а следователно и надеждността на една система зависят основно от методите за защита на паметта. В момента микропроцесорите разполагат с вградено в чипа устройство за управление на паметта (MMU – memory management unit), което позволява на отделните задачи да работят в хардуерно защитено пространство памет.

- **Толеранс на грешките и достъпност.**

Дори и в най-добрия софтуер има скрити, латентни грешки.
Колкото по-сложно става дадено приложение, колкото повече функции се имплементират в него, толкова повече нараства възможността за грешки, вградени в самата RTOS. За целта операционната система следва да предоставя механизми за откриване на грешки и възстановяване на системата.
Естествено начините за възстановяване на системата след грешка е силно зависима от самото приложение.

3. Изисквания при избор на RTOS

- **Права за достъп до системни ресурси.**

За системи с критична надеждност е от съществено значение процесът с основна важност винаги да разполага с необходимия му ресурс памет. В повечето RTOS, паметта за поддържане на контролния блок на всяка задача или друг обект се разпределя от общ централен ресурс. Когато обаче има грешка или вирус в задачата може като резултат да се стигне до ситуация, в която тази задача създава много голям брой обекти, всеки от които иска обем памет и се изчерпват ресурсите на системата.

- **Гарантиран достъп до ресурс "време на процесора".**

При повечето RTOS се използва приоритетно базирания принцип на разпределение на процесорното време. [2,4,7] При тази схема, нишката с най-висок приоритет получава процесорно време за изпълнение. Когато има повече от една нишка с един и същи приоритет, се използва принципът на времеделение на малки порции, които се предоставят на всяка нишка равностойно. При този механизъм няма гаранция за обезпечаване на критичните задачи вътре в даден приоритет.