10. Механизмы синхронизации

Реализация семафоров

- Семафор это динамическая структура, создаваемая в ядре, с помощью специального примитива (сервиса) ядра;
- **`P`** и **`V`** операции – примитивы ядра;
- Примитив – процедура, реализованная в ядре, выполняющая неделимое действие; может вызываться из задач приложения
- Вызов примитива – событие, обрабатываемое ядром (осществляется через механизм программных прерываний)

Реализация механизма семафоров

ПРИЛОЖЕНИЕ

```
Создать_Семафор(Доступ,1);

P(Доступ);
   . . .
V(Доступ);
```

ЯДРО

```
type Семафор;
Создать_Семафор(S: Семафор; НачЗначение:
Integer);
P(S: Семафор);
V(S: Семафор);
```

Реализация механизма семафоров (2)

```
type Семафор = ^pСемафор
     pСемафор = record
                Очередь: УказательНаОчередь;
                НЗначение: 0..1;
                ТЗначение: 0..1;
     endtype
```

```
Создать_Семафор(S: Семафор;
                НачЗначение: Integer);
   1. S:= new(Семафор);
   2. S.Очередь:= СоздатьОчередь;
   3. S.НЗначение:= НачЗначение;
   4. S.ТЗначение:= НачЗначение;
```

Реализация механизма семафоров (3)

```
P(S: Семафор):

   1. Запретить_прерывания();
   2. if (S.TЗначение=0) then
      2.1. Поставить_В_Очередь(ТекПроц,S.
Очередь)
      2.2. R:= Взять_Из_Очереди(ОчередьГотовых);
      2.3. Переключение_Контекста(ТекПроц,R);
   3. S.TЗначение:= 0;
   3. Разрешить_прерывания();
```

- *Примечание. Подразумевается, что* **Переключение_Контекста()** *разрешает прерывания*

Реализация механизма семафоров (4)

```
V(S: Семафор):

    1. Запретить_Прерывания();
    2. R:= Взять_Из_Очереди(S.Очередь);
    3. Поставить_В_Очередь(R,ОчередьГотовых);
    4. if(Очередь_Пуста(S.Очередь))
           then S.TЗначение:= 1;
    5. Разрешить_Прерывания();
```

Пример сервисов, поддерживающих семафоры в RTEK

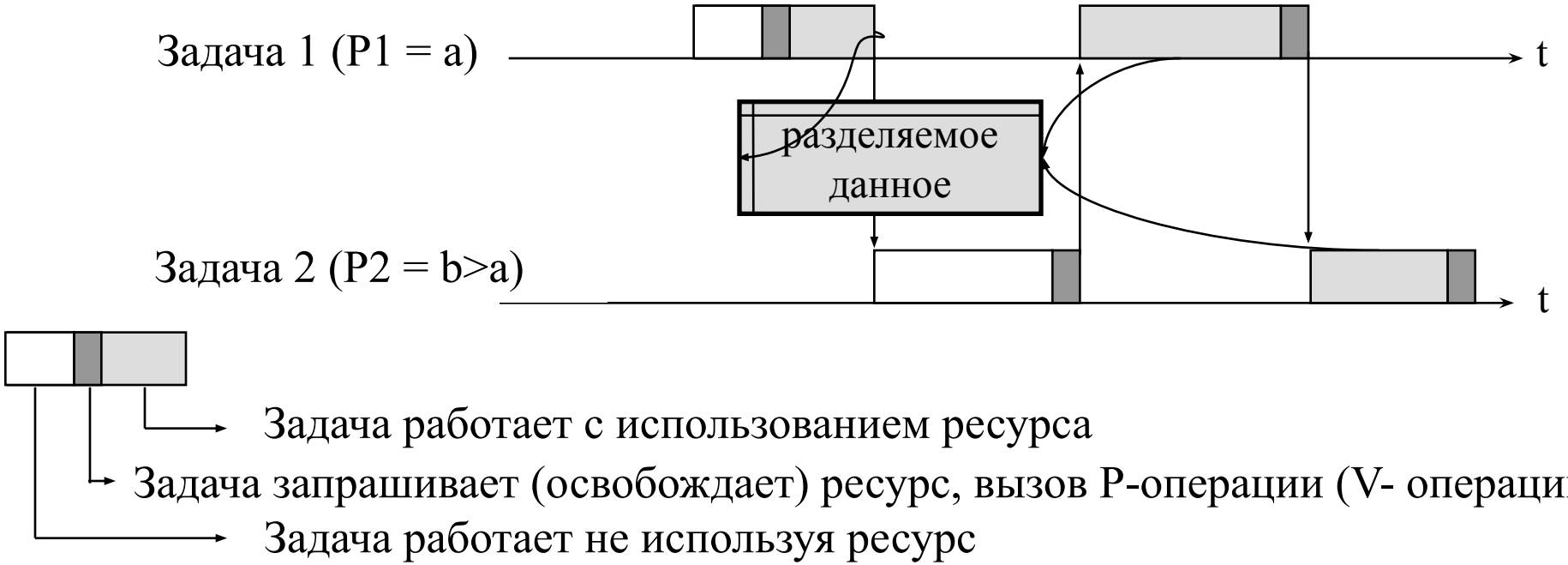
KC_CloseSema	–	end the use of dynamic semaphore
KC_DefSemaCount	–	define a semaphore count
KC_DefSemaName	–	define the name of a previously opened dynamic semaphore
KC_DefSemaName	–	define the properties of a semaphore
KC_GetSemaCount	–	get the current semaphore count
KC_GetSemaName	–	get the name of a semaphore
KC_GetSemaProp	–	get the properties of a semaphore
KC_GetSemaWaiters	–	get the number and list of tasks waiting on semaphore
KC_InitSemaClassProp	–	initialize the semaphore object class properties

Пример сервисов, поддерживающих семафоры в RTEK

KC_LookupSema	–	look up a semaphore's name to get its handle
KC_OpenSema	–	allocate and name a dynamic semaphore
KC_SignalSema	–	signal a semaphore
KC_SignalSemaM	–	signal multiply semaphores
KC_TestSema	–	test a semaphore
KC_TestSemaT	–	test a semaphore and wait for a specified time if the semaphore is not DONE
KC_TestSemaW	–	test a semaphore and wait if the semaphore is not DONE

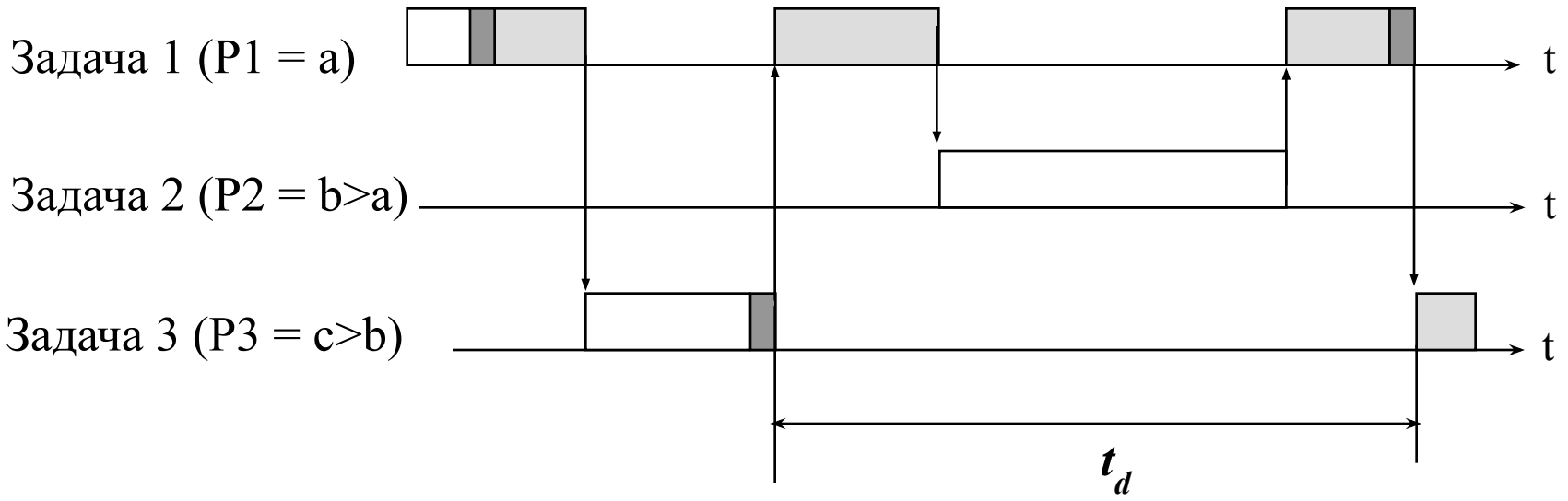
Использование семафоров

Разделяемый ресурс – данные, процедуры, устройства, одновременно требуемые для выполнения нескольких задач



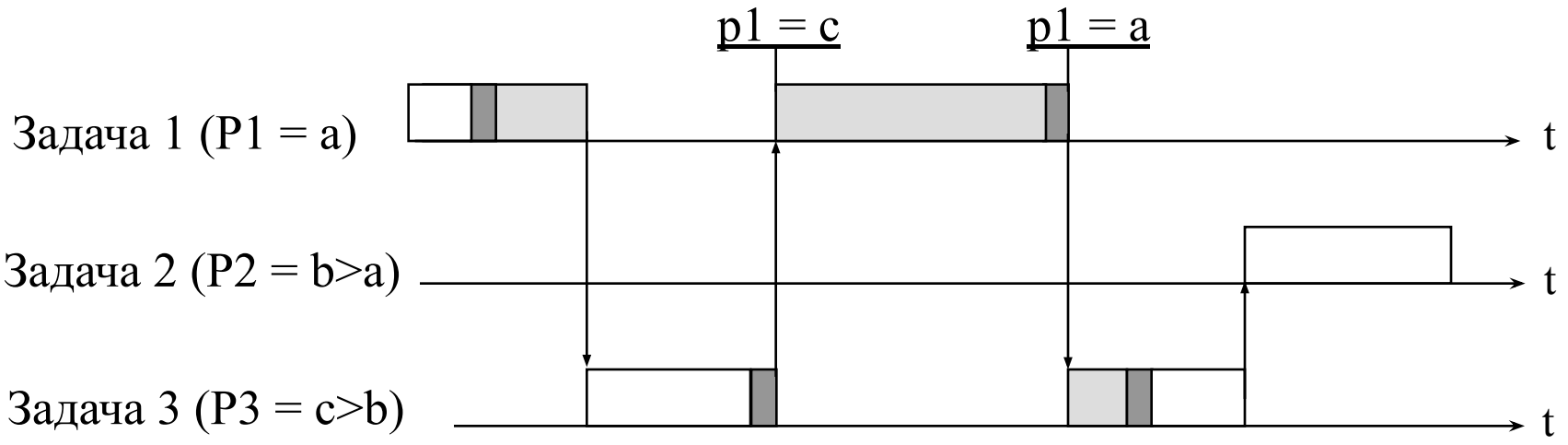
(!) Задача с более высоким приоритетом ждет пока менее приоритетная задача закончит работу с ресурсом

Проблема инверсии приоритетов



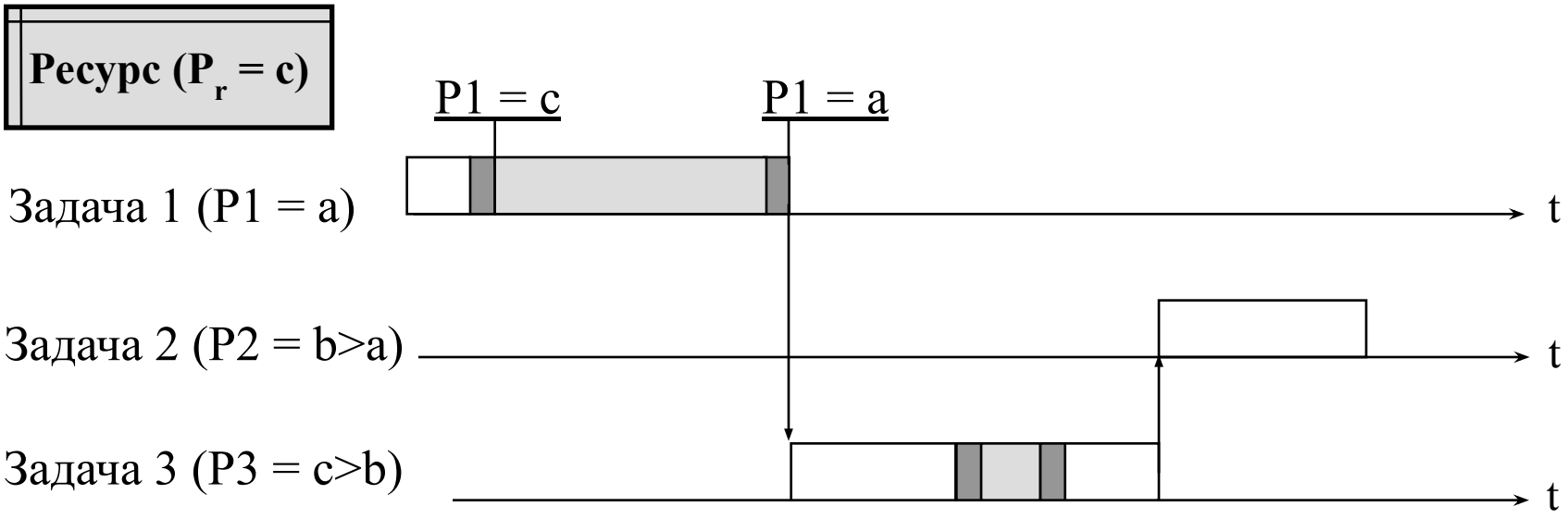
Высокоприоритетная Задача 3 удерживается низкоприоритетной Задачей 2 в течение временного интервала t_d, значение которого трудно просчитать заранее

Протокол наследования приоритетов



Приоритет Задачи 1, удерживающей ресурс, поднимается до значения приоритета Задачи 3 в момент, когда Задача 3 выставляет требование на этот ресурс

Priority Ceiling протокол



- С Ресурсом связывается понятие приоритета;
- В процессе проектирования приложения Ресурсу присваивается приоритет той использующей его задачи, который является наивысшим;
- При запросе Задачей Ресурса ей присваивается приоритет этого Ресурса

Недостатки механизма семафоров

- Низкий уровень, слабая защищенность от ошибок при программировании
    - Возможность использования **`P(S)`** и **`V(S)`** над одним семафором из разных задач (семафор **`S`** может быть установлен в 0 в одной задаче, а в 1 – в другой)
    - Возможность вызова V-операции над семафором без предварительного вызова P-операции
- Не позволяет реализовать протокол наследования приоритетов, т. к. не определен процесс-«владелец» семафора

Mutual Exclusion

Множество значений:

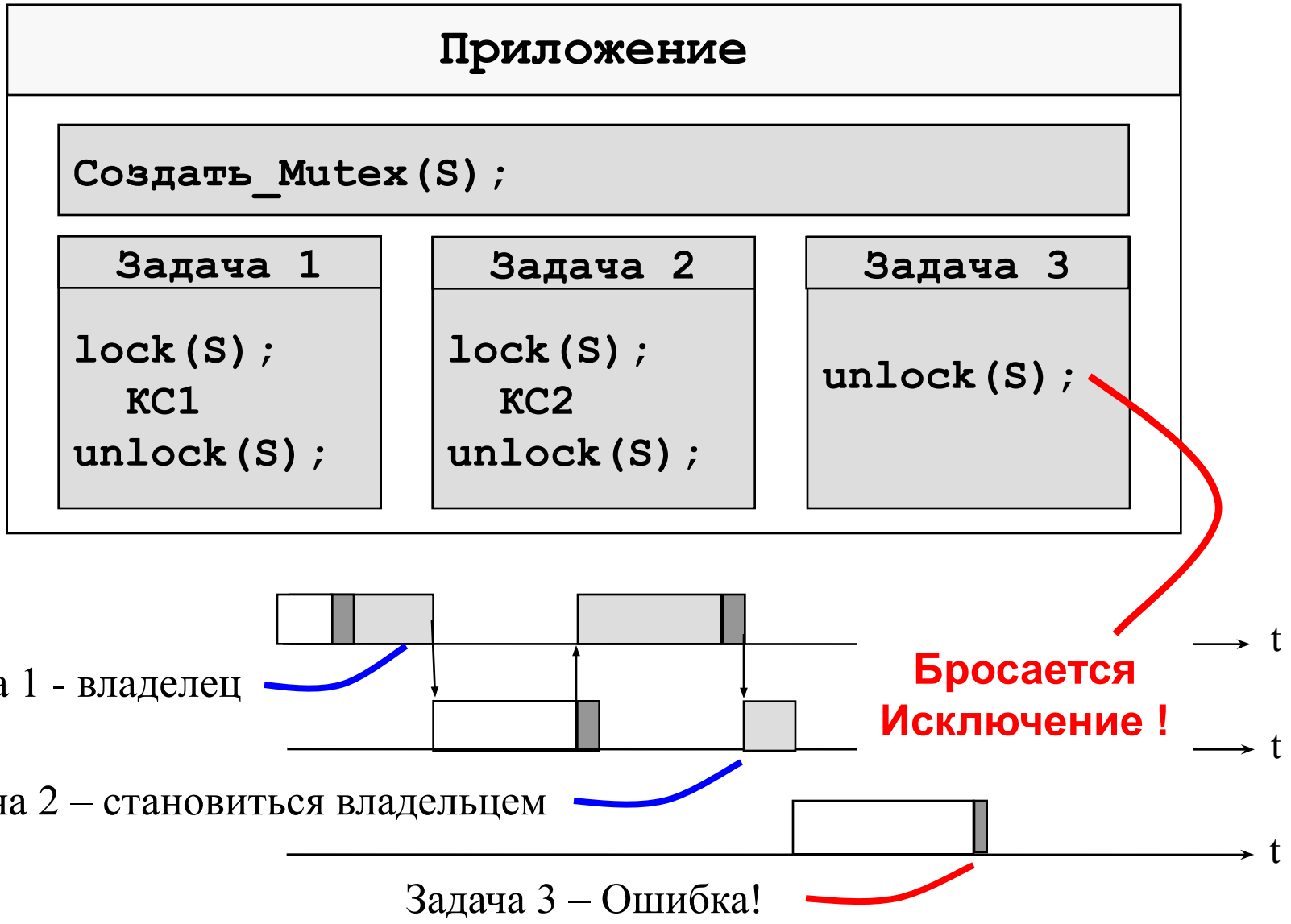
```
Type mutex = (занят, свободен);
```

Множество действий:

```
Var S: mutex;

lock(S):
      if(S=занят) then ждать(S=свободен);
      Владелец:= ТекущийПроцесс;
      S:= занят;
unlock(S):
      if(Владелец=Текущий процесс) then
         S:= свободен
      else
         исключение;
```

Mutual Exclusion (2)



Реализация mutex

```
type mutex = ^pMutex
     pMutex = record
                Очередь: УказательНаОчередь;
                Состояние: занят, свободен;
                Владелец: процесс;
     endtype
```

```
Создать_mutex(S: mutex);
   1. S:= new(mutex);
   2. S.Очередь:= СоздатьОчередь;
   3. S.Владелец:= nill;
   4. S.Состояние:= свободен;
```

Реализация mutex (2)

Реализация протокола наследования приоритетов требует модификации дескриптора задачи –TCB:

```
type TCB = ^pTCB
     pTCB = record
        Адрес: АдресПроцедуры;
        Состояние: (Готов,Активен,
Блокирован);
        Приоритет: 1..128;
(!)     ТПриоритет: 1..128;
        Контекст: АдресСтека
     endtype
```

Реализация mutex (3)

```
lock(S: mutex):
    1. Запретить_Прерывания;
    2. if(S.Состояние=занят) then
       2.1. S.Владелец.ТПриоритет:=
                          ТекПроц.Приоритет;
       2.2. ждать(S.Состояние=свободен);
    3. S.Владелец:= ТекПроц;
    4. S.Состояние:= занят;
    5. Разрешить_Прерывания;
```

- *Примечание. Реализация* **ждать()** *показана на слайде 4*

Реализация mutex (4)

```
unlock(S: mutex);
   1. Запретить_Прерывания;
   2. if(S.Владелец=ТекПроц) then
      2.1. R:= Взять_Из_Очереди(S.Очередь);
      2.2. Поставить_В_Очередь(R,
ОчердьГотовых)
      2.3. S.Состояние:= свободен;
      else Исключение;
   3. S.Владелец.ТПриоритет:=
                  S.Владелец.Приоритет;
   4. S.Владелец:= nill;
   5. Разрешить_Прерывания();
```