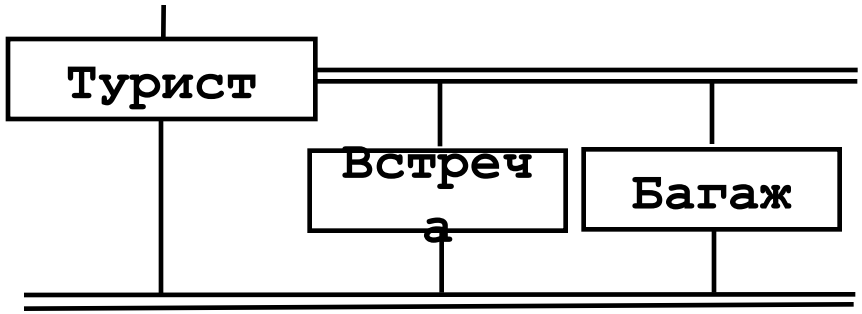
Задачи (язык Ада)

Описание тела задачи:

```
task body is <имя_задачи>
    <описание_данных>
begin
    <описание_действий>
end <имя_задачи>;
```

Инициирование задачи

```
program Турист

task body  Встреча is;
begin
    . . .
end Встреча;

task body  Багаж is;
begin
    . . .
end Багаж;

begin
    initiate(Встреча);
    initiate(Багаж);
end Турист;
```



Неявная инициализация:

```
begin
    nill;
end Турист;
```

(!) Активны три задачи

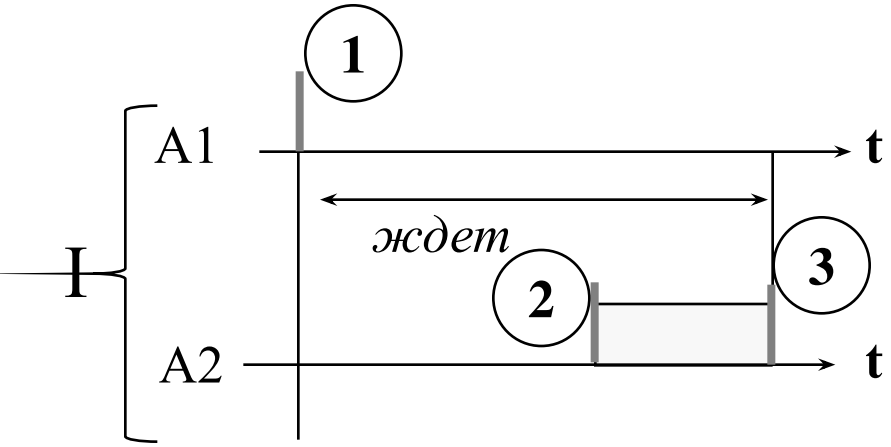
Механизм рандеву

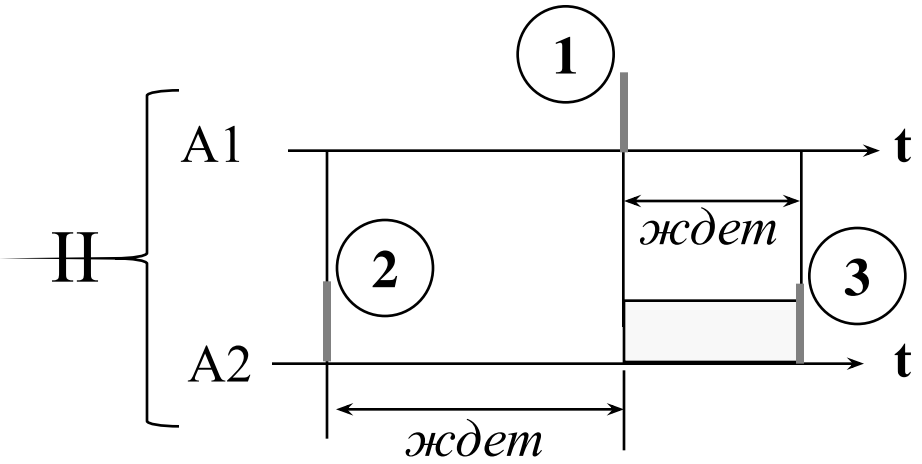
Вызов входа:

```
task body A1 is

begin
   . . .
   A2.R;                    (1)
   . . .
end A;
```

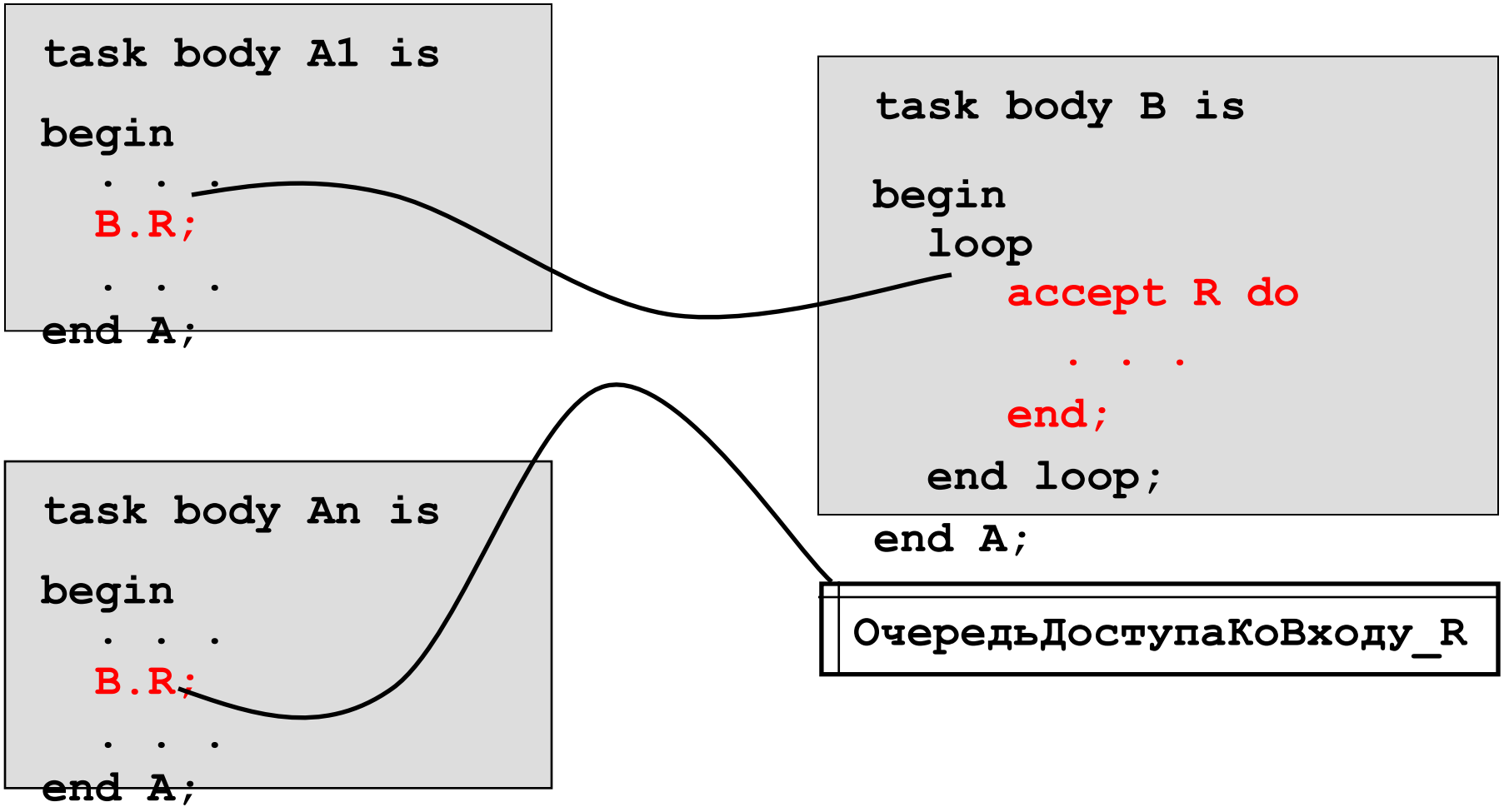
Вход:

```
task body A2 is

begin
   . . .
   accept R do              (2)
     <обработка входа>
   end;                     (3)
   . . .
end A;
```





Взаимное исключение



Описание задачи

Спецификация задачи:

```
task <имя_задачи> is
    entry <имя_входа> [(формальные_параметры)]
end <имя_задачи>;
```

Тело задачи:

```
task body is <имя_задачи>
    <описание_данных>
begin
    <описание_действий>
    accept <имя входа> [(<формальные параметры>)] do
        <описание_действий_входа>
    end;
    <описание_действий>
end <имя_задачи>;
```

Описание задачи, пример

```
program Преобразование_символа;

task Читать;
task Преобразовать
   entry R(P: in CHARACTER);
end Преобразовать;

task body Читать;
Ch: CHARACTER;
X: FILE;
begin
   Open (X);
   while not EOF(X) loop
    Get(Ch);
    Преобразовать.R(Ch);
   end loop;
   Close(X)
end Читать;
```

```
task body Преобразовать;
Y: CHARACTER;
begin
   loop
      accept R(P: in CHARACTER) do
         Y:= P;
      end;
      Put(UPPER(Y));
   end loop;
end;

begin
   nill;
end.
```

Реализация механизма семафоров

```
task body Процесс_1;
begin
   . . .
   Семафор.P;
   КритическаяСекция_1;
   Семафор.V;
. . .
end;
```

```
task body Процесс_2;
begin
    . . .
   Семафор.P;
   КритическаяСекция_2;
   Семафор.V;
    . . .
end;
```

```
task body Семафор;
begin
    loop
        accept P do end;
        accept V do end;
    end loop;
end Сигнал;
```

Реализация механизма сигналов

```
task body Процесс_1;
begin
   loop
    . . .
    Сигнал.Ждать;
    . . .
   end loop;
end;
```

```
task body Процесс_N;
begin
   loop
    . . .
       Сигнал.Послать;
    . . .
   end loop;
end;
```

```
task body Сигнал;
begin
   loop
      accept Послать do end;
    if Ждать'COUNT > 0 then
       accept Ждать do end;
    end if;
   end loop;
end Сигнал;
```

Пример – «Почтовый ящик»

```
task body ПочтовыйЯщик;
Буфер: СООБЩЕНИЕ;
begin
    loop
       accept Отправить(Передача: in СООБЩЕНИЕ) do
       Буфер:= Передача;
     end;
       accept Принять(Прием: out СООБЩЕНИЕ) do
       Прием:= Буфер;
     end;
    end loop;
end Почтовый_Ящик;
```

Пример – «Почтовый ящик» (2)

```
task body Отправитель;
begin
   loop
   . . .
   ПочтовыйЯщик.Отправить(Послание);
   . . .
   end loop;
end Отправитель;
task body Получатель;
begin
   loop
   . . .
   Почтовый_Ящик.Принять(Послание);
   . . .
   end loop;
end Получатель;
```

Пример – «Почтовый ящик» (3)

- Задача очень похожа на «Поставщик-Потребитель»
- Буфер рассчитан только одно сообщение и увеличивать его нет смысла так как :
- Очень жесткий способ синхронизации :

`Принять` и **`Отправить`** выполняются по очереди,

Отбор среди входов, задача «Поставщик-Потребитель»

```
select
       when (A) =>
        accept X do
            . . .
        end;
or
       when (B) =>
        accept Y do
            . . .
        end;
end select;
```

```
task body ПоставщикПотребитель;

Размер: constant INTEGER = 20;
Буфер: array(1..Размер) of СООБЩЕНИЕ;
ТекущийЗапись, ТекущийЧтение: INTEGER = 1;
Количество: INTEGER = 0;
```

Задача «Поставщик-Потребитель» (2)

```
begin
  loop
    select
      when (Количество < Размер) =>
       accept Записать(X: in СООБЩЕНИЕ) do
          Буфер(ТекущийЗапись):= X;
       end;
       Текущий:= (ТекущийЗапись + 1) mod Размер;
       Количество:= Количество + 1;
    or
      when (Количество > 0) =>
       accept Прочитать(Y: out СООБЩЕНИЕ) do
          Y:= Буфер(ТекущийЧтение);
       end;
       Текущий:= (ТекущийЧтение + 1) mod Размер;
       Количество:= Количество - 1;
    end select;
  end loop;
end ПоставщикПотребитель;
```

Оператор Отбора среди входов

```
select
    [when <условие>=>]
    <отбираемая альтернатива>
    [последовательность операторов]
{or
    [when <условие>=>]
    <отбираемая альтернатива>
    [последовательность операторов]}
[else
     последовательность операторов]
end select;
```

- Отбираемой альтернативой может быть
  - ператор входа **accept**
  - оператор задержки
- **delay <**выражение, указывающее величину задержки**>**

Отбор среди входов

- Открытый вход – вход, у которого условие входа в данный момент является истинным

- Если нет ни открытых операторов задержки, ни фразы **`else`**, задача будет неопределенно долго ждать вызова входа, связанного с открытым оператором входа **`accept`** и обработает первый поступивший вызов

- Если имеется открытый оператор задержки, задача будет ожидать вызова входа для открытого оператора входа в течение заданного временного интервала, а затем выполнит соответствующую альтернативу.

- Если нет ни открытых альтернатив, ни фразы **`else`**, возникает исключительная ситуация SELECT_ERROR

- Если нет открытого оператора задержки, но есть фраза **`else`**, то задача немедленно приступит к выполнению части **`else`**.