6. Механизм семафоров
Двоичный семафор

Множество значений:

```
Type ДвСемафор = 0..1;
```

Множество действий:

```
Var S: ДвСемафор;
P(S):
      if(S=0) then ждать(S>0);
      S:= 0;
V(S):
      S:= 1;
```

Двоичный семафор, особенности

- При обработке описания семафора **S** с ним связывается очередь заблокированных процессов; очередь обслуживается в соответствии с определенной дисциплиной, не допускающей бесконечного ожидания процесса (например, FIFO)
- При **S=0** операция **ждать(S>0)** блокирует процесс, вызвавший **P**–операцию и и ставит его в очередь до момента выполнения условия **S>0;**
- При выполнении **V**–операции активизируется первый процесс из очереди; продолжение этого процесса происходит с действия **S:=0** приостановленной **P**–операции
- **P** и **V** операции над семафором **S** являются неделимыми (могут выполняться только одним процессом)

Обеспечение взаимного исключения

```
Var S: ДвСемафор = 1;
Ri: Begin
        loop
            P(S);
            КС_I;
            V(S);
            Остальное_I;
        endloop;
    End;

Parbegin
   R1;. . .Rn;
Parend.
```

- **P** и **V** операции неделимы; т.е. выполнять **P(S)** может только один процесс ;
- Если один из процессов выполнил **P(S)** и вошел в свою **КС**, то **S=0** и все остальные процессы, выполняющие **P(S),** будут блокироваться. Т.е взаимное исключение обеспечено
- При выполнении V(S) соблюдается очередность, т.е. никто не будет ждать бесконечно долго

Общий (счетный) семафор

Множество значений:

```
Type ОбСемафор = 0..N;
```

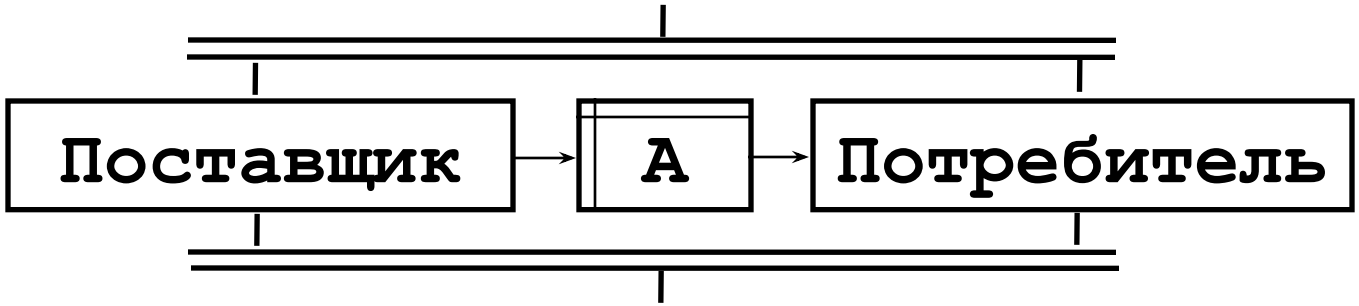
Множество действий:

```
Var S: ОбСемафор;
P(S):
      if(S=0) then ждать(S>0);
      S:= S-1;
V(S):
      S:= S+1;
```

Общий семафор, особенности

В отличие от двоичного семафора процесс, вызвавший **P** -операцию блокируется только в том случае, если до него **P**–операция была вызвана **N** процессами и ни один из них не вызывал **V** -операцию

Пример: «Поставщик-Потребитель»



Поставщик: производит единицы информации и записывает их в буфер **A**, размер которого **N**

Потребитель: читает и обрабатывает информацию из буфера **A**

Требуется реализовать процессы Поставщик и Потребитель таким образом, чтобы выполнялись условия:

а) **Поставщик** не может записывать в полный буфер;

б) **Потребитель** не может читать из пустого буфера;

в) одновременные запись и чтение недопустимы

«Поставщик-Потребитель» - реализация

```
Var Доступ: ДвСемафор = 1;
    Полон: ОбСемафор = N; Пуст: ОбСемафор = 0;
```

```
Поставщик:
   loop
      Производство;
(*)   P(Полон);
(**)  P(Доступ);
      Запись_в_буфер;
      V(Доступ);
      V(Пуст);
   endloop.
```

```
Потребитель:
   loop
      P(Пуст);
      P(Доступ);
      Чтение_из_буфера;
      V(Доступ);
      V(Полон);
      Обработка;
   endloop.
```

```
   Parbegin  Поставщик; Потребитель  Parend;
```

«Поставщик-Потребитель» - пояснение

Проследим работу схемы в следующих ситуациях:

а) Поставщик пишет, **Потребитель** остановлен в действии **Обработка**. Перед каждой записью в буфер значение семафора **Полон** уменьшается на 1. При полном буфере **Поставщик** блокируется на семафоре **Полон** в операции **P(Полон)**

б) Потребитель читает, Поставщик остановлен в действии **Производство**. Перед каждым чтением уменьшается значение семафора **Пуст**. При пустом буфере потребитель блокируется в операции **P(Пуст)**

в) Взаимное исключение доступа обеспечивается семафором **Доступ**

Недостатки механизма семафоров

- Низкий уровень, слабая защищенность от ошибок при программировании
    - Перестановка местами действий (*) и (**) в примере на слайде 7 приводит к блокировке
    - 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)
- *Не позволяет реализовать протокол наследования приоритетов, т. к. не определен процесс-«владелец» семафора*