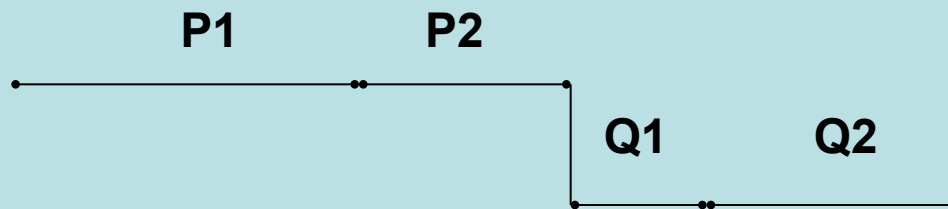


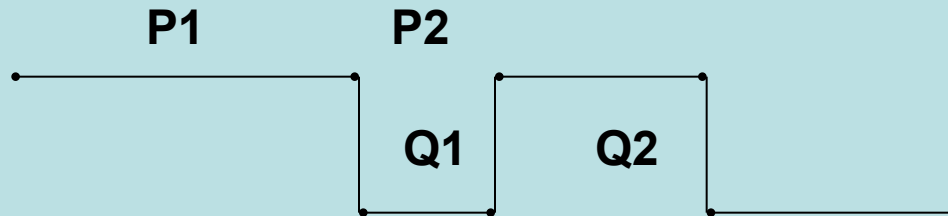
# **Взаимодействующие параллельные процессы**

# Параллельные процессы

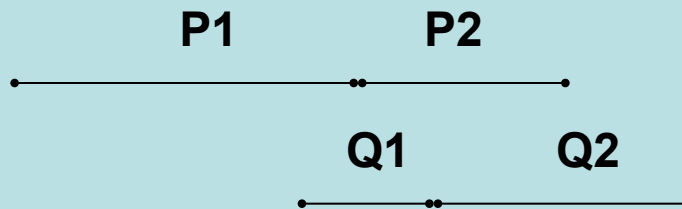
Последовательные процессы



Логические параллельные процессы

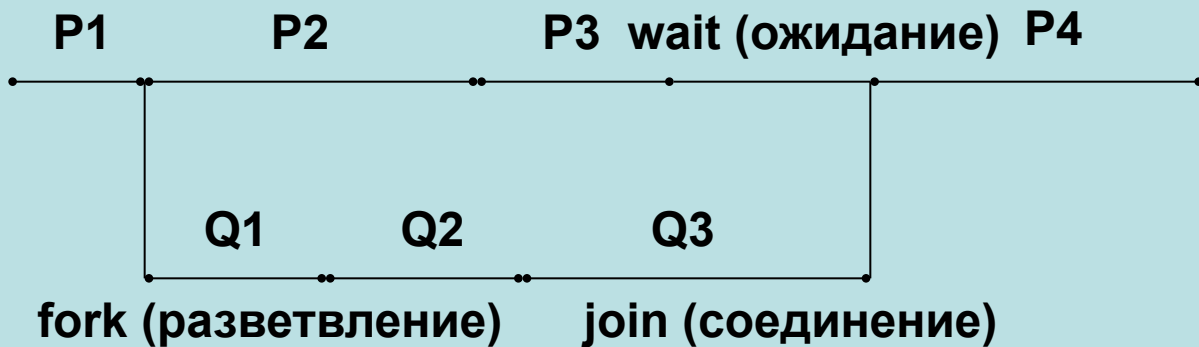


Физические параллельные процессы

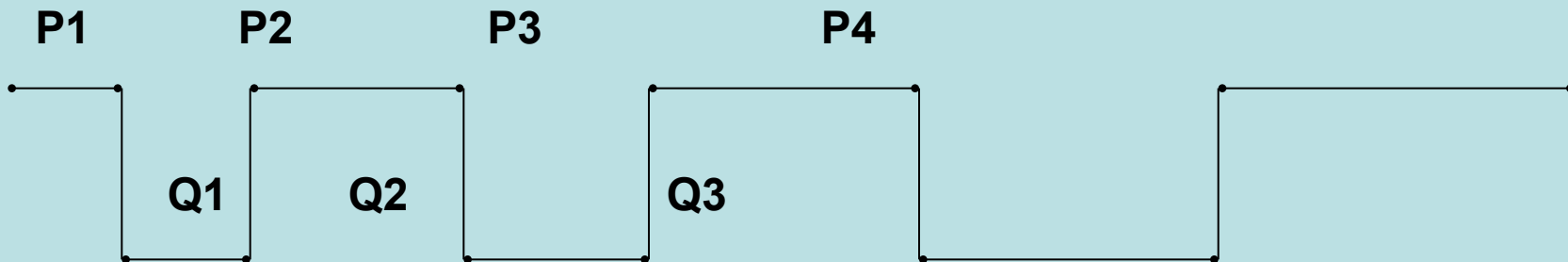


# Взаимодействующие процессы

Физически параллельные



Логически параллельные



# Взаимодействующие процессы

**Независимые процессы** имеют свое множество переменных и ресурсов. Другие процессы не могут изменить значения переменных этого процесса.

**Взаимодействующие процессы** – совместно используют общие ресурсы, и выполнение одного процесса влияет на результат другого. Ресурсами могут быть области памяти, файлы данных, ВУ и т.д.

Взаимодействовать могут **конкурирующие процессы**, каждый из которых использует совместный ресурс только для своих целей, либо процессы, совместно выполняющие общую работу – **асинхронные процессы**.

# Использование общего ресурса

## Создание

```
1 mov CX,Count
2 inc CX
3 mov Count,CX
```

## Завершение

```
4 mov CX,Count
5 dec CX
6 mov Count,CX
```

В результате прерывания последовательность действий обеих программ может измениться. Пусть `Count = 10` и эта последовательность станет 1-4-5-6-2-3.

```
1 CX = 10
2 CX = 11
3 Count = 11
4 CX = 10
5 CX = 9
6 Count = 9
```

Правильное значение `CX = 10` Эта ситуация называется *коллизией*. Работа с `Count` не является единой неделимой операцией.

```
1 inc Count
```

```
2 dec Count
```

# Проблема критического участка

Общий ресурс, совместно используемый несколькими параллельными процессами, получил название – **критический ресурс**.

Часть программы, использующая критический ресурс, называется **критическим участком** (критическим интервалом, критической секцией, критической областью).

Требования к критическому участку программы:

- только один процесс может находиться внутри критического участка (**взаимное исключение**);
- ни один процесс не должен ждать бесконечно долго входа в критический участок;
- ни один процесс не может оставаться внутри критического интервала бесконечно долго;
- операции взаимного исключения должны выполняться корректно при нарушении работы одного или нескольких процессов вне критического участка (устойчивость к нарушениям);
- вход и выход взаимного исключения должны быть идентичными для всех процессов и не зависеть от их числа (симметрия).

# Методы взаимного исключения

Используется множество методов взаимного исключения взаимодействующих параллельных процессов в критических участках:

- взаимное исключение с активным ожиданием:
  - запрещение прерываний,
  - строгое чередование,
  - алгоритмы Деккера и Петерсона,
  - операция проверки и установки;
- семафоры и мьютексы;
- мониторный механизм взаимного исключения;
- обмен сообщениями между процессами;

# Параллельные процессы без взаимоисключения

- 1 (переменные управления взаимодействием)
- |                               |                               |
|-------------------------------|-------------------------------|
| <u>procedure</u> PROC1;       | <u>procedure</u> PROC2;       |
| <u>begin</u>                  | <u>begin</u>                  |
| <u>while</u> (true) <u>do</u> | <u>while</u> (true) <u>do</u> |
| <u>begin</u>                  | <u>begin</u>                  |
| 2 {вход взаимодействия;}      | {вход взаимодействия;}        |
| критический участок 1;        | критический участок 2;        |
| 3 {выход взаимодействия;}     | {выход взаимодействия;}       |
| независимая часть 1;          | независимая часть 2;          |
| <u>end</u>                    | <u>end</u>                    |
| <u>end</u> ;                  | <u>end</u> ;                  |
- 4 Cobegin  
(нач. установка)  
PROC1; PROC2;  
coend



# Взаимоисключение строгим чередованием процессов

```
1      var NP: 1,2;

      procedure PROC1;                procedure PROC2;
      begin                            begin
      while (true) do                  while (true) do
      begin                            begin
2      while NP=2 do;                while NP=1 do;
      критический участок 1;          критический участок 2;
3      NP:=2;                          NP:=1;
      независимая часть 1;            независимая часть 1;
      end                                end
      end;                              end;

4      Begin
      NP:=1;
      cobegin
      PROC1; PROC2; coend;
      end.
```

# Попытка взаимного исключения с использованием флагов

```
1      var C1, C2: boolean;

      procedure PROC1;                procedure PROC2;
      begin                            begin
      while (true) do                  while (true) do
      begin                            begin
2      while C2 do;                    while C1 do;
      C1:=true;                          C2:=true;
      критический участок 1;              критический участок 2;
3      C1:=false;                        C2:=false;
      независимая часть 1;                независимая часть 2;
      end                                end
      end;                              end;

      Begin
4      C1:=false; C2:=false;
      cobegin
      PROC1; PROC2; coend;
      end.
```

# Алгоритм Деккера

VAR C1,C2:Boolean; NP:1,2;

procedure PROC1;

begin

while (true) do

begin

C1:=TRUE;

while C2 do

  If NP=2 then

    begin

      C1:=FALSE;

      While NP=2 do;

      C1:=TRUE;

    end;

критический участок 1;

NP:=2; C1:=FALSE;

независимая часть 1;

end end;

procedure PROC2;

begin

while (true) do

begin

C2:=TRUE;

while C1 do

  If NP=1 then

    begin

      C2:=FALSE;

      While NP=1 do;

      C2:=TRUE;

    end;

критический участок 2;

NP:=1; C2:=FALSE;

независимая часть 2;

end end;

begin

NP:=1;

C1:=FALSE; C2:=FALSE;

Cobegin PROC1; PROC2; coend;

end.

# Алгоритм Петерсона

```
1      var C1, C2: boolean; var NP:1,2;

      procedure PROC1;                procedure PROC2;
      begin                            begin
      while (true) do                  while (true) do
      begin                            begin
2      C1:=true;                        C2:=true;
      NP:=2;                            NP:=1;
      while (C2 and NP=2) do;          while (C1 and NP=1) do;
3      критический участок 1;          критический участок 2;
      C1:=false;                        C2:=false;
      независимая часть 1;              независимая часть 2;
      end                                end
      end;                              end;

4      begin
          C1:=false; C2:=false;
      cobegin PROC1; PROC2; coend;
      end.
```

# Взаимоисключение операцией проверка и установка (Test and Set)

```
1      Var Common:boolean;  
      Procedure TS (Лок, Общ);  
      begin Лок:=Общ;  
          Общ:=true; end;  
  
      procedure PROC1;                procedure PROC2;  
      Var C1:boolean;                Var C2:boolean;  
      begin                            begin  
      while (true) do                while (true) do  
2      Begin C1:= true;                Begin C2:=true;  
      while C1 do TS (C1,Common);    while C2 do TS (C2,Common);  
      критический участок 1;        критический участок 2;  
3      Common:=false;  
      независимая часть 1;          Common:=false;  
      end                            независимая часть 1;  
      end                            end  
      end;  
      end;  
  
4      begin Common:=false;  
      cobegin PROC1; PROC2; coend;  
      end.
```

# Операция Test and Set

```
Procedure TS (Лок, Общ);  
begin  
    Лок:=Общ;  
    Общ:=TRUE;  
end;
```

```
Общ:=false;  
(критич. участок свободен)  
Лок1:=True;  
While Лок1 do TS(Лок1,Общ);  
    true    false  
    false <- false  
    false    true
```

```
Общ:=true;  
(критич. Участок занят)  
Лок2:=True;  
While Лок2 do TS(Лок2,Общ);  
    true    true  
    true <- true  
    true    true
```

**Команда** BTS источник, индекс

Переносит бит по адресу источник[индекс] -> CF

(Лок),

Затем бит источник[индекс] <- 1 (Общ).

```
L: BTSM, 1      ; вход  
    JC L        ; взаимоиключения  
; критическая секция
```

# Семафоры

Семафоры, как средство синхронизации параллельных процессов, предложил голландский математик Э. Дейкстра (E. W. Dijkstra) в 1965 г.

Семафор  $S$  это агрегат данных, который состоит из счетчика с целыми значениями  $S.C$  и очереди процессов  $S.Q$ , ждущих входа в критический участок. При создании семафора счетчик принимает начальное значение  $C \geq 0$ , а очередь – пустая.

Две операции над числовыми семафорами.

$P(S)$ - проверить (proberen)  
Down( $S$ )

$V(S)$  - увеличить (verhogen)  
Up( $S$ )

```
S.C:=S.C-1
If S.C < 0 then
begin
  перевести Процесс в
  состояние «Ожидание»;
  S.Q:= процесс
End
```

```
S.C:=S.C+1
If S.C <= 0 then
  перевести первый Процесс
  в S.Q в состояние
  «Готовность»
```

# Свойства числового семафора

Работу числового семафора можно сравнить с работой автоматизированной двери, которая открывается, если бросить жетон. Жетон пропускает только одного человека. Жетон бросает не тот, кто проходит, а другой.

Свойства числовых семафоров.

Пусть  $C_0$  – начальное значение  $S.C$ ,  $nP$  и  $nV$  – общее число выполнения операций  $P(S)$  и  $V(S)$ .

Тогда:

- текущее значение счетчика семафора:  $S.C = C_0 - nP + nV$ ;
- число процессов в состоянии ожидания:  $nB = \max(0, -S.C)$ ;
- число форсирований:  $nF = \min(nP, C_0 - nV)$ .

Последний параметр показывает насколько  $nP$  больше  $nV$ . По аналогии с автоматической дверью  $nF$  дает знать, что количество прошедших равно наименьшему из двух чисел, одно из которых есть общее количество опущенных жетонов  $C_0 + nV(S)$ , а другое – число желающих пройти дверь.



# Логический семафор - mutex

Вместо числовой переменной S.C может использоваться переменная логического типа. Такой логический семафор получил название **мьютекс** (mutex – MUltual EXclusion semaphor, семафор взаимного исключения).

S.C принимает значения TRUE и FALSE, а операции P(S) и V(S) выражаются действиями:

P(S)

```
If S.C
then S.C:=FALSE
else
begin
перевести Процесс в
состояние «Ожидание»;
S.Q:= процесс
end;
```

V(S)

```
If S.Q=Null {очередь пуста}
then
S.C:=TRUE
else
перевести первый Процесс
из S.Q в состояние
«Готовность»;
```

Двоичные семафоры используются для операции взаимного исключения нескольких процессов в случае, когда в критическом участке должен находиться только один процесс, числовые семафоры обладают также другими расширенными возможностями.

# Взаимоисключение числовым семафором

VAR S:Semaphore;

```
procedure PROC1;  
begin  
while (true) do  
begin  
    P(S);  
    критический участок 1;  
    V(S);  
    независимая часть 1;  
end  
end;
```

```
procedure PROC2;  
begin  
while (true) do  
begin  
    P(S);  
    критический участок 2;  
    V(S);  
    независимая часть 2;  
end  
end;
```

```
begin  
S.C:=1;  
cobegin  
    PROC1;  
    PROC2;  
coend;  
end.
```

# Синхронизация процессов «Главный – Подчиненный»

```
VAR Event : Semaphore;
```

```
procedure MASTER;
```

```
begin
```

```
    предшествующая часть 1;
```

```
        P(Event) ;
```

```
    оставшаяся часть 1;
```

```
end;
```

```
procedure SLAVE;
```

```
begin
```

```
    предшествующая часть 2;
```

```
        V(Event) ;
```

```
    оставшаяся часть 2;
```

```
end;
```

```
begin
```

```
    Event.C:=0;
```

```
    cobegin MASTER; SLAVE; coend;
```

```
end.
```

Обратите внимание, что здесь начальное значение счетчика семафора Event (событие) равно 0, т.е. семафор закрыт. Операция P(Event) переводит главный процесс в состояние «Ожидание», если значение семафора не было изменено. Открыть семафор может подчиненный процесс, сменив значение счетчика на 1, если подчиненный процесс выполнит V(Event) раньше.

# Синхронизация процессов «Производитель – Потребитель»

```
VAR Buf:Record;  
Start,Finish:Semaphore;
```

```
procedure PRODUSER;  
VAR Rec:Record;  
begin  
    создать запись;  
    P(Finish);  
    Write(Rec,Buf);  
    V(Start);  
end;
```

```
procedure CONSUMER;  
VAR Rec:Record;  
begin  
    P(Start);  
    Read(Rec,Buf);  
    V(Finish);  
    обработать запись;  
end;
```

```
begin  
Start.C:=0; Finish.C:=1;  
cobegin  
    Repeat PRODUSER Until FALSE;  
    Repeat CONSUMER Until FALSE;  
coend;  
end.
```

Обратите внимание на начальные значения счетчиков семафоров.

# «Производитель – Потребитель»

## множественный буфер

```
VAR Buf:array [1..N] of Record;  
Full, Empty, S:Semaphore;
```

```
procedure PRODUSER;  
VAR Rec:Record;  
begin  
    создать запись;  
    P (Empty) ;  
    P (S) ;  
Write (Rec, Buf) ;  
    V (S) ;  
    V (Full) ;  
end;
```

```
procedure CONSUMER;  
VAR Rec:Record;  
begin  
    P (Full) ;  
    P (S) ;  
    Read (Rec, Buf) ;  
    V (S) ;  
    V (Empty) ;  
    обработать запись;  
end;
```

```
begin  
S.C:=1; Full.C:= ; Empty.C:= ;  
cobegin  
    Repeat PRODUSER Until FALSE;  
    Repeat CONSUMER Until FALSE;  
coend;  
end.
```

# «Читатели – Писатели» с приоритетом читателей

VAR Nrdr:integer; W,R:Semaphore;

procedure READER;

begin

P(R);

Nrdr:=Nrdr+1;

If Nrdr = 1 then P(W);

V(R);

Читать данные;

P(R);

Nrdr:=Nrdr-1;

If Nrdr = 0 then V(W);

V(R);

end;

procedure WRITER;

begin

P(W);

Писать данные;

V(W);

End;

Begin Nrdr:=0; W.C:=1; R.C:=1;

cobegin

Repeat READER Until FALSE;

. . .

Repeat READER Until FALSE;

Repeat WRITER Until FALSE;

. . .

Repeat WRITER Until FALSE;

coend; end.

# «Читатели – Писатели» с приоритетом писателей

```
VAR Nrdr:integer; W,R,S:Semaphore;
```

```
procedure READER;
```

```
begin
```

```
    P(S);
```

```
    P(R);
```

```
Nrdr:=Nrdr+1;
```

```
If Nrdr = 1 then P(W);
```

```
    V(S);
```

```
    V(R);
```

```
    Читать данные;
```

```
    P(R);
```

```
Nrdr:=Nrdr-1;
```

```
If Nrdr = 0 then V(W);
```

```
    V(R);
```

```
end;
```

```
procedure WRITER;
```

```
begin
```

```
    P(S);
```

```
    P(W);
```

```
    Писать данные;
```

```
    V(S);
```

```
    V(W);
```

```
End;
```

```
Begin    Nrdr:=0; W.C:=1; R.C:=1; S.C:=1;
```

```
cobegin
```

```
    Repeat READER Until FALSE;
```

```
    . . .
```

```
    Repeat READER Until FALSE;
```

```
    Repeat WRITER Until FALSE;
```

```
    . . .
```

```
    Repeat WRITER Until FALSE;
```

```
coend;
```

```
end.
```