Взаимодействие процессов

Причины для взаимодействия

- Повышение скорости работы
- Совместное использование данных
- Модульная конструкция системы
- Удобство пользователя

Категории средств общения

- Сигнальные
- Канальные
- Разделяемая память

Канальные средства связи

- Безопасность
- Информативность
- Могут быть использованы для связи между процессами на разных системах

Буферизация

Сохраняет ли канал информацию, переданную одним процессом, до получения её другим?

1. Буфер отсутствует
2. Буфер ограниченной ёмкости
3. Неограниченный буфер

Модели передачи данных

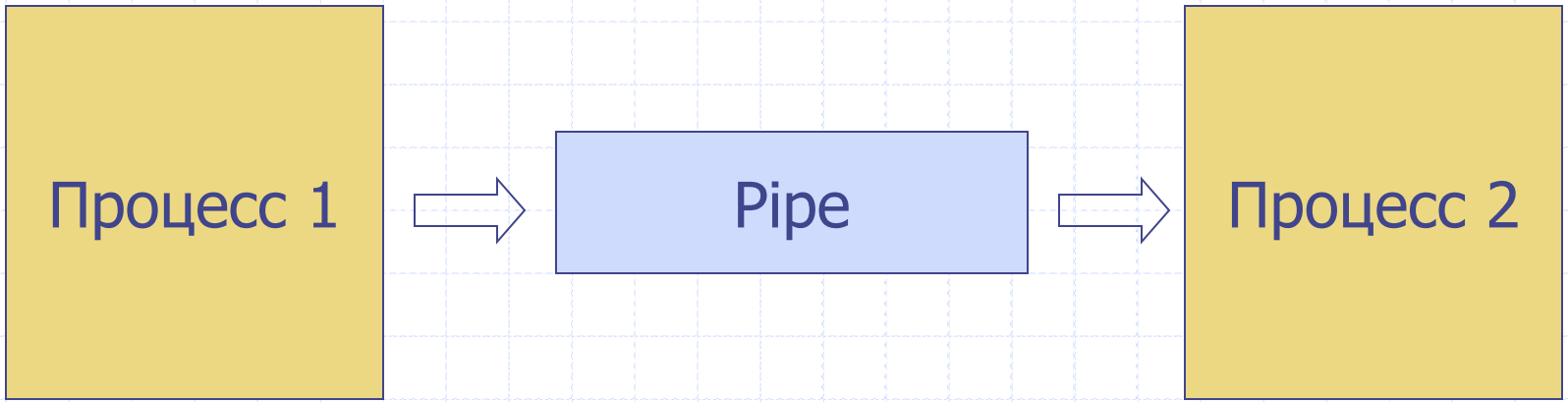
- Поток ввода-вывода

  Данные представляются как единый неструктурированный поток байтов

- Сообщения

  Вводится структура данных, подразумевающая по крайней мере границы сообщений

Pipe



- Неименованый канал – pipe
  - может использоваться внутри процесса или передаваться наследникам
- Именованный канал – named pipe, FIFO
  - Может использоваться для связи между любыми процессами

Надёжность линии связи

1. Не происходит потери информации
2. Не происходит искажения информации
3. Не появляется лишней информации
4. Не нарушается порядок данных в процессе обмена

Завершение связи

- Процедура завершения связи
- Что происходит при несогласованном завершении связи?
  - Завершение ожидающего процесса
  - Специальный код уведомления

Алгоритмы синхронизации

Interleaving

Пусть имеются две активности:

P: a b c

Q: d e f

Параллельное выполнение в режиме разделения времени:

a b c d e f

a b d c e f

a b d e c f

a b d e f c

a d b c e f

.....

d e f a b c

Interleaving

P: x=2 y=x-1

Q: x=3 y=x+1

Возможные результаты:

(x, y): (3, 4), (2, 1), (2, 3) и (3, 2)

Interleaving

- Детерминированный набор активностей
  - Результат всегда одинаков.
- Недетерминированный набор активностей
  - Результат зависит от неконтролируемых особенностей порядка выполнения.

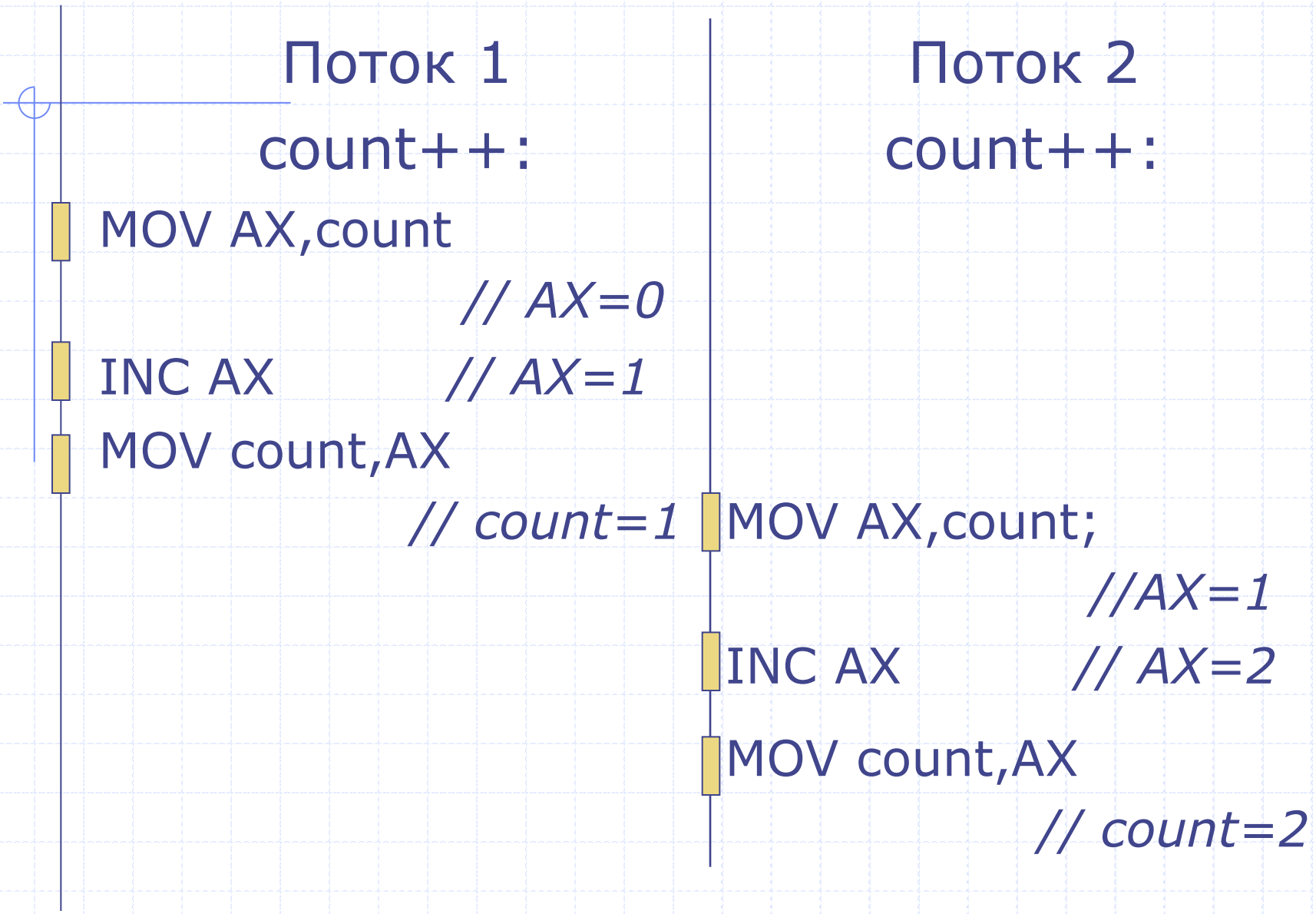
Атомарные операции

- атомарная операция – операция, которая выполняется как единое целое или не выполняется вообще.
- Во многих языках программирования все обычные операции не являются атомарными.

Недетерминированное выполнение

Поток 1	Поток 2
count++:	count++:
MOV AX,count	
// ax=0	
	MOV AX,count;
	// ax=0;
INC AX	
//ax=1	
	INC AX
	//ax=1
MOV count,AX	
// count=1	
	MOV count,AX
	// count=1

Недетерминированное выполнение



Условия Бернстайна

R(P) – набор входных переменных для всех атомарных операций в P

W(P) – набор выходных переменных для всех атомарных операций в P

Если $W(P) \cap W(Q) = \varnothing$, $W(P) \cap R(Q) = \varnothing$ и $R(P) \cap W(Q) = \varnothing$ то выполнение P и Q детерминированно

Средства синхронизации

- атомарные операции
  - в C11 и C++ 11 есть специальные типы _Atomic, atomic и функции atomic_...;
  - в Windows API есть функции interlocked....
- Объекты для синхронизации и функции ожидания
  - события, семафоры, мьютексы, условные переменные,...
- Критические секции

Race Condition

Если набор активностей не детерминирован, то говорят что он находится в состоянии гонки (race).

Взаимное исключение одновременного выполнения процессов называют mutual exclusion.

Критическая секция

Место в программе, в котором возможно возникновение race condition называют критической секцией

```
while (some condition)
{
        entry section           // пролог
        critical section
        exit section            // эпилог
        remainder section
}
```

Требования к алгоритму

- Задача решается программным способом, атомарно только выполнение инструкций процессора
- Не должны использоваться предположения о скорости выполнения процессов и числе процессоров
- Только один процесс должен находится в критической секции (mutual exclusion)
- Процессы вне критических участков не должны блокировать другие процессы
- Не должно возникать неограниченно долгого ожидания входа в критический участок

Запрет прерываний

while (*some condition*)
{
запретить все прерывания
critical section
разрешить все прерывания
remainder section
}

Переменная-замок

```
shared int lock = 0;
 while (some condition)
{
    while(lock);
    lock = 1;
    critical section
    lock = 0;
    remainder section
}
```

Строгое чередование

```
shared int turn = 0;

while (some condition)
{
    while(turn != i);
    critical section
    turn = 1-i;
    remainder section
}
```

Флаги готовности

```
shared int ready[2] = {0, 0};

while (some condition)
{
    ready[i] = 1;
    while(ready[1-i]);
    critical section
    ready[i] = 0;
    remainder section
}
```

Алгоритм Петерсона

```
shared int ready[2] = {0, 0};
shared int turn;
while (some condition)
{
    ready[i] = 1;
    turn =1-i;
    while(ready[1-i] && turn == 1-i);
    critical section
    ready[i] = 0;
    remainder section
}
```

Алгоритм булочной

```
shared bool choosing[n];
shared int number[n];

while (some condition)
{
    choosing[i] = true;
    number[i] = max(number[0], ..., number[n-1]) + 1;
    choosing[i] = false;
    for(j = 0; j < n; j++)
    {
        while(choosing[j]);
        while(number[j] != 0 && (number[j],j) <  (number[i],i));
    }
    critical section
    number[i] = 0;
    remainder section
}
```

(a,b)<(c,d) если a<c или
a=c и b<d