Асинхронные параллельные процессы

Процесс -	Абстрактное понятие, относящееся к программе. Часто процессом называют программу и все её элементы: адресное пространство, глобальные переменные , регистры, стек, счетчик команд, состояние, открытые файлы, дочерние процессы и т. д.
Параллельные процессы -	Выполняющиеся вместе процессы.
Асинхронные параллельные процессы -	Строго независимые процессы, поэтому возникает необходимость в их синхронизации.

Пример из САПР

Задача: Найти периметр прямоугольника с точками с координатами (x1,x2) и (y1,y2)

Решение:

1. *var1 = x2 - x1*
2. *var2 = y2 - y1*
3. *var3 = var1 + var2*
4. *var4 = var3 * 2*

Дейкстра предложил 2 оператора parbegin и parend:

1. *parbegin*
   var1 = x2 - x1
   var2 = y2 - y1
   parend
2. *var3 = var1 + var2*
3. *var4 = var3 * 2*

Вывод: при распараллеливании процессов экономим время!

Синхронизация процессов.

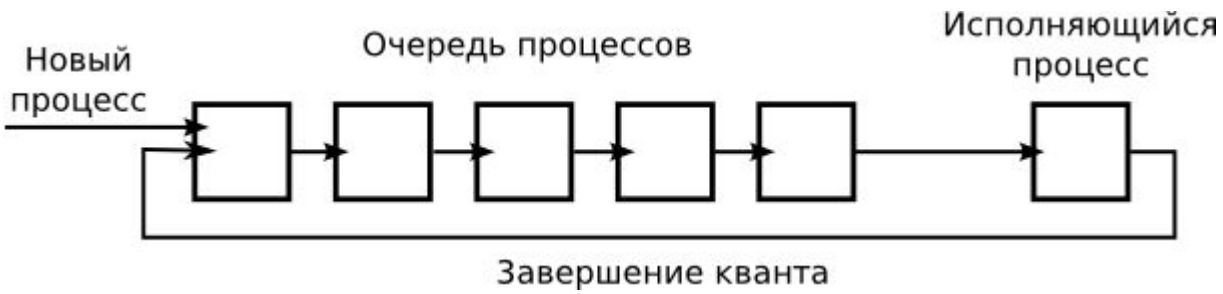
Процесс обмена сообщениями между процессами для исключения гонок и тупиков.

Способы:

- Критические секции
- Семафоры
- Мьютексы
- События
- Таймеры

Планирование процессов.

- определение момента времени переключения процесса – кванта времени
- выбор нового процесса для выполнения



Диспетчеризация процессов.

«с приоритетом»

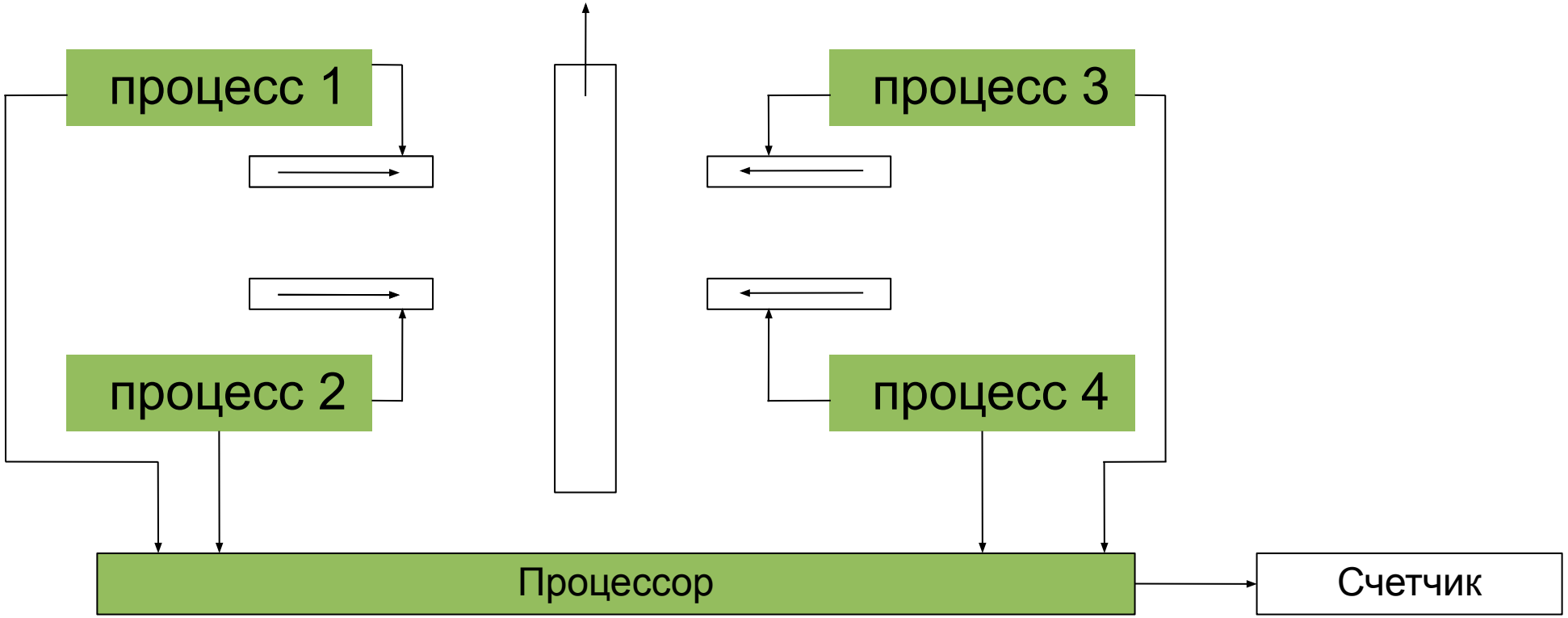
«карусель»

Каждому процессу присваивается приоритет, т.о. определяется жесткая очередность их выполнения.

Каждому готовому к исполнению процессу дается равный шанс запуска.

Очередь процессов – множество процессов готовых к исполнению.

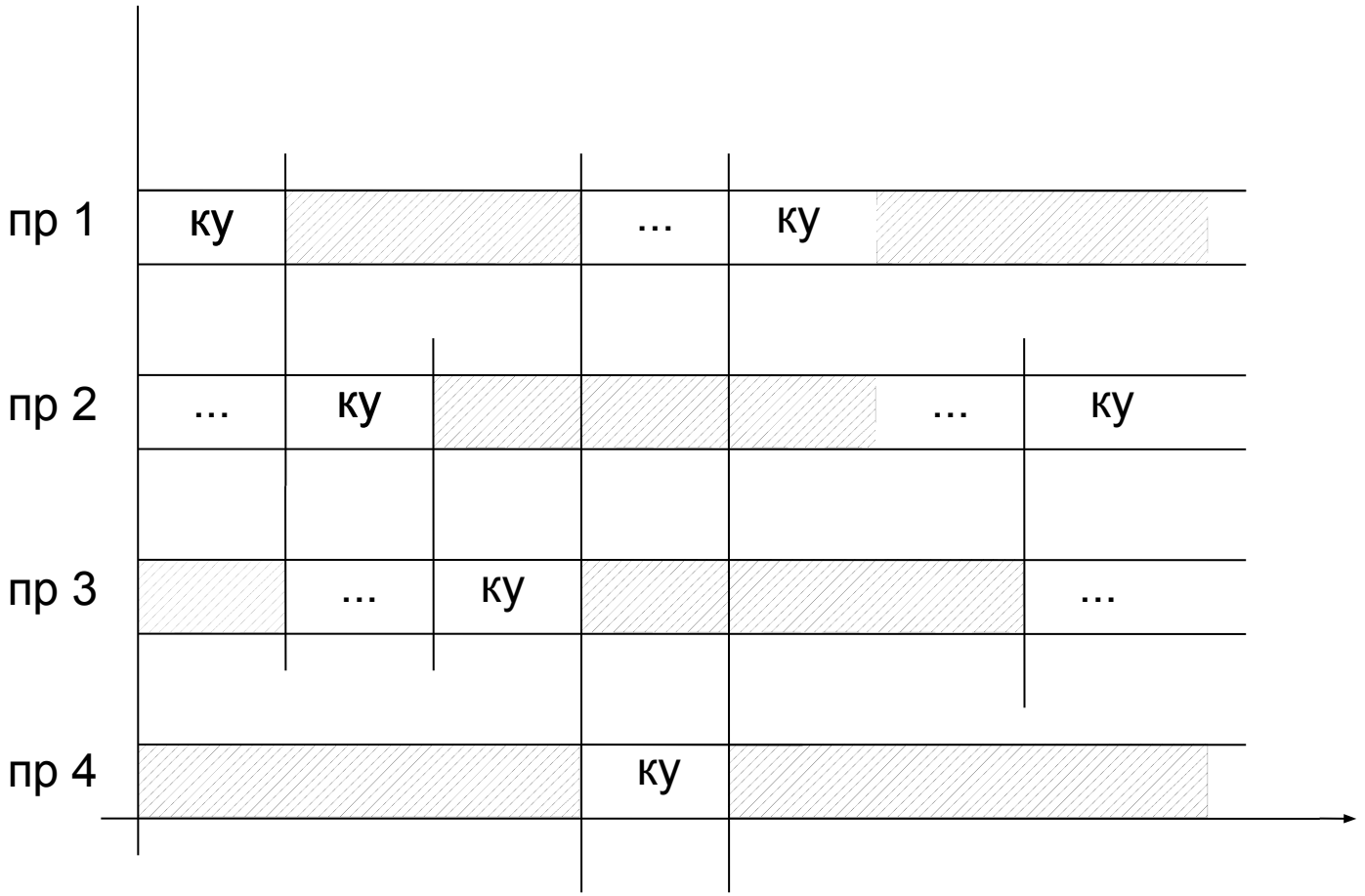
Взаимоисключение.



Если процессы могут перехватить процессор друг у друга, то может произойти потеря данных. Во избежания этого, необходимо ввести *ВЗАИМОИСКЛЮЧЕНИЕ*, то есть не разрешать перехват при обращении к разделяемым данным.

Критический участок.

это часть программы, во время выполнения которой обращения других процессов к разделяемым ресурсам запрещены. Обрамляют критический участок 2 примитива: *входвзаимоисключение* и *выходвзаимоисключение*.



ку -критический участок процесса

-некритический участок процесса

... - ожидание выхода из критической области другого процесса

Реализация взаимоисключения.

Задача: создать механизм взаимоисключений для следующих ограничений:

- на машине нет специальных команд взаимоисключение;
- скорости ассинхронных процессов заранее неизвестны;
- процессы, находящиеся вне критических участков, не должны мешать другим процессам входить в их собственные критические участки;
- не должно быть бесконечного откладывания момента входа процесса в критическую область

Алгоритм *Деккера (первый вариант)*.

```
NOMER_PROC=1

PROCEDURE 1

IF (NOMER_PROC = 1) DO
крит. участок
NOMER_PROC = 2
остальные операторы
ELSE
ждать
END

PROCEDURE 2

IF (NOMER_PROC = 2) DO
крит. участок
NOMER_PROC = 1
остальные операторы
ELSE
ждать
END
```

Преимущества метода: просто реализуется взаимоисключение

Недостатки метода: сначала должен реализоваться процесс 1

-- возможно только поочередное вхождение процессов в критические области

-- если один процесс обращается в критическую область больше, чем другой, то это невозможно

Алгоритм *Деккера (второй вариант)*.

```
PR1WNYTRI=0
PR2WNYTRI=0

PROCEDURE 1

IF (PR2WNYTRI = 0)
  PR1WNYTRI = 1
  крит. участок
  PR1WNYTRI = 0
END

PROCEDURE 2

if (PR1WNYTRI = 0)
  PR2WNYTRI = 1
  крит. участок
  PR2WNYTRI = 0
END
```

Преимущества метода: не надо процессам чередоваться

Недостатки метода: после условия и перед присвоением значения перменной PR?WNYTRI возможен вход в критическую область обоих процессов

Алгоритм *Деккера (третий вариант, усовершенствование второго)*.

```
PR1WNYTRI=0
PR2WNYTRI=0

PROCEDURE 1


PR1WNYTRI = 1
IF (PR2WNYTRI = 0)
крит. участок
PR1WNYTRI = 0
END

PROCEDURE 2

PR2WNYTRI = 1
IF (PR1WNYTRI = 0)
крит. участок
PR2WNYTRI = 0
END
```

Преимущества метода: оба процесса одновременно не могут войти в критическую область

Недостатки метода: возможен тупик (бесконечное ожидание), если оба процесса установят переменные одновременно

Алгоритм *Деккера (четвертый вариант)*.

```
VAR MAINPR: (1,2)
PR1WANT,PR2WANT:logical
PROCEDURE 1;
BEGIN
WHILE 1 DO
BEGIN
PR1WANT:='T';
WHILE PR2WANT DO
IF (MAINPR=2)
BEGIN
PR1WANT:='F';
WHILE (MAINPRC=2) DO;
PR1WANT:='T';
END
критический участок
MAINPR:2
PR1WANT:='F';
остальные операнды
END
END
```

Пояснения: процесс 1 устанавливает флаг PR1WANT в истину (PR1WANT:='T')

- если у второго процесса флаг имеет значение F (false) то переходим к выполнению критического участка (WHILE PR2WANT DO)
- если второй процесс тоже хочет войти в КО, входим во внутренний цикл (IF (MAINPR=2)) :
  - если избранный процесс – первый, пропускаем тело (IF (MAINPR=2)) и ждем, когда процесс 2 сбросит флаг
  - если избранный процесс – второй, входим в тело (IF (MAINPR=2)), сбрасываем флаг первого процесса (PR1WANT:='F') и зацикливаемся пока процесс 2 не сбросит флаг (WHILE (MAINPR=2) DO), выполнив КО, т. е. сбрасывая флаг PR1WANT:='F' процесс 1 дает возможность второму процессу войти в КО.

Аппаратная реализация взаимоисключения.

команда testandset

testandset(a,b) –

- читает значение логической переменной b
- копирует его в a
- устанавливает для b значение истина