

Нити исполнения

# Нити исполнения (threads)

Ввести массив a

Ввести массив b

$$a = a + b$$

$$c = a + c$$

Вывести массив a

Ввести массив a

Ожидание окончания  
операции ввода

Ввести массив b

Ожидание окончания  
операции ввода

Ввести массив c

Ожидание окончания  
операции ввода

$$a = a + b$$

$$c = a + c$$

Вывести массив c

# Нити исполнения (threads)

## Процесс 1

Создание процесса 2

Создание общей памяти

Ввести массив A

Ожидание ввода A

Ввести массив B

Ожидание ввода B

Ввести массив C

Ожидание ввода C

$C=A+C$

Вывести массив C

Ожидание вывода C

## Процесс 2

Переключение контекста

Создание общей памяти

Переключение контекста

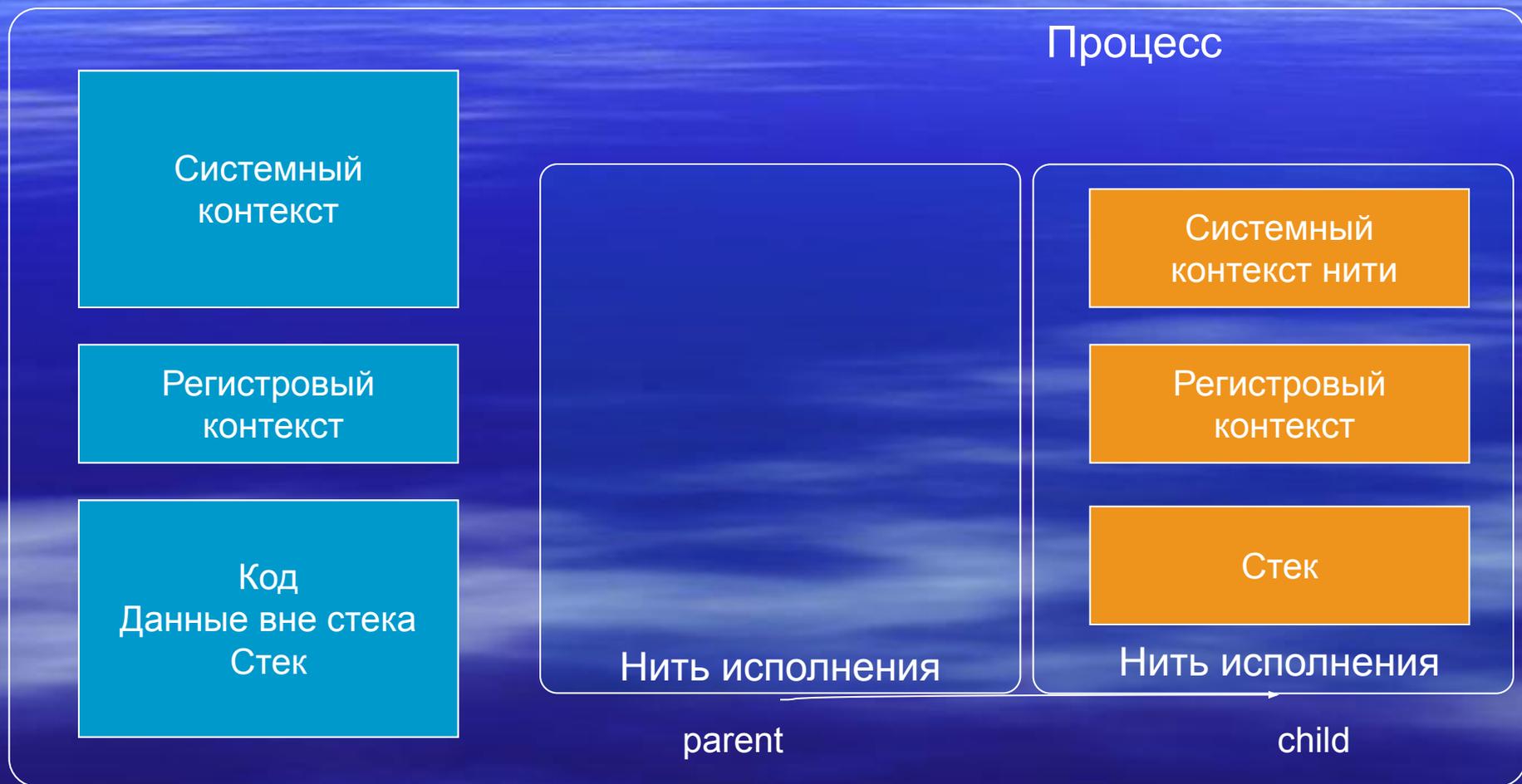
Ожидание ввода A и B

Переключение контекста

$A=A+B$

Переключение контекста

# Нити исполнения (threads)



# Нити исполнения (threads)



# Нити исполнения (threads)

## Нить 1

Создание нити 2

Ввести массив A  
Ожидание ввода A  
Ввести массив B  
Ожидание ввода B  
Ввести массив C  
Ожидание ввода C

$C=A+C$   
Вывести массив C  
Ожидание вывода C

## Нить 2

Переключение контекста  
Переключение контекста

Ожидание ввода A и B

Переключение контекста  
Переключение контекста

$A=A+B$

# Активности и атомарные операции

Активность : приготовление бутерброда

- Отрезать ломтик хлеба
  - Отрезать ломтик колбасы
  - Намазать хлеб маслом
  - Положить колбасу на хлеб
- Атомарные или неделимые операции
- 
- The diagram consists of four bullet points on the left, each with an arrow pointing to a single point on the right. From this point, four arrows point back to each of the four bullet points. This visualizes the decomposition of a task into its atomic components.

*Активность* - последовательное выполнение ряда действий, направленных на достижение определенной цели

# Interleaving

Активность P: a b c

Активность Q: d e f

Последовательное выполнение PQ: a b c d e f

Псевдопараллельное выполнение  
(режим разделения времени) :

?  
a b d c e f  
a b d e c f  
a b d e f c  
...  
d e f a b c

# Детерминированные и недетерминированные наборы активностей

$$P: x=2 \\ y=x-1$$

$$Q: x=3 \\ y=x+1$$

(3, 1) (3, 4) (3, )

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

- *Недетерминированный* набор – при одинаковых начальных данных возможны разные результаты
- *Детерминированный* набор – при одинаковых начальных данных **всегда** один результат

# Условия Бернштейна (Bernstain)

P: 1)  $x=u+v$

Q: 2)  $y=x*w$

Входные переменные

$$R_1 = \{u, v\}$$

$$R_2 = \{x, w\}$$

$$R(P) = \{u, v, x, w\}$$

Выходные переменные

$$W_1 = \{x\}$$

$$W_2 = \{y\}$$

$$W(P) = \{x, y\}$$

Если:

$$1) W(P) \cap W(Q) = \{\emptyset\}$$

$$2) W(P) \cap R(Q) = \{\emptyset\}$$

$$3) R(P) \cap W(Q) = \{\emptyset\}$$

то набор активностей  $\{P, Q\}$  является  
детерминированным

# Критическая секция

Время	Студент 1	Студент 2	Студент 3
17-05	Приходит в комнату		
17-07	Достает 6 бут. пива		
17-09		Приходит в комнату	
17-11		Уходит за пивом	
17-13			Приходит в комнату
17-15			Уходит за пивом
17-17			
17-19		Покупает 6 бут. пива	
17-21			Покупает 6 бут. пива
17-23			
17-25			Приносит пиво Приходит в комнату
17-27			

# Структура процесса, участвующего во взаимодействии

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

# Программные алгоритмы организации взаимодействия

Требования, предъявляемые к алгоритмам

1. Программный алгоритм должен быть программным
2. Нет предположений об относительных скоростях выполнения и числе процессоров
3. Выполняется условие взаимоисключения (mutual exclusion) для критических участков
4. Выполняется условие прогресса (progress)
5. Выполняется условие ограниченного ожидания (bound waiting)

# Программные алгоритмы организации взаимодействия

## Запрет прерываний

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

Обычно используется внутри ОС

# Программные алгоритмы организации взаимодействия

## Переменная-замок

```
Shared int lock = 0;
```

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

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

Нарушается условие взаимного исключения

# Программные алгоритмы организации взаимодействия

## Строгое чередование

	Shared int turn = 0;		Shared int turn = 1;
$P_0$		$P_1$	
	while (some condition) {		while (some condition) {
	while (turn != i); <i>while (turn != 0);</i>		while (turn != 1);
	critical section		critical section
	turn = 1-i;		turn = 0;
	remainder section		remainder section
	}		}

Нарушается условие прогресса

Условие взаимного исключения выполняется

# Программные алгоритмы организации взаимодействия

## Флаги готовности

$P_0$	$P_1$
Shared int ready[2] = {0, 0};	Shared int ready[2] = {1, 1};
while (some condition) {	while (some condition) {
ready[i] = 1; <i>ready[0] = 1;</i>	ready[1] = 1;
while (ready[1-i]); <i>while (ready [1]);</i>	while (ready [0]);
critical section	critical section
ready[i] = 0; <i>ready[0] = 0;</i>	ready[1] = 0;
remainder section	remainder section
}	}

2-я часть условия прогресса нарушается

1-я часть условия прогресса выполняется

Условие взаимоисключения выполняется

# Программные алгоритмы организации взаимодействия

## Алгоритм Петерсона

Shared int ready[2] = {0, 0};

Shared int turn;

$P_0$

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

$P_1$

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

Все 5 условий выполняются