

# АРХИТЕКТУРА ОПЕРАЦИОННЫХ СИСТЕМ

ЛЕКЦИЯ 1.6

# АППАРАТНАЯ ПОДДЕРЖКА ВЗАИМОИСКЛЮЧЕНИЙ

## Команда Test-And-Set

```
int Test-And-Set (int *a) {  
    int tmp = *a;  
    *a = 1;  
    return tmp;  
}
```

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

```
while (some condition) {  
    while (Test-And-Set (&lock));  
    critical section  
    lock = 0;  
    remainder section  
}
```

Нарушается условие ограниченного ожидания



# АППАРАТНАЯ ПОДДЕРЖКА ВЗАИМОИСКЛЮЧЕНИЙ

## Команда Swap

```
void Swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}
```

```
Shared int lock = 0;  
int key = 0;  
  
while (some condition) {  
    key = 1;  
    do Swap (&lock, &key);  
    while (key);  
        critical section  
    lock = 0;  
        remainder section  
}
```

Нарушается условие ограниченного ожидания

# НЕДОСТАТКИ ПРОГРАММНЫХ АЛГОРИТМОВ

- НЕПРОИЗВОДИТЕЛЬНАЯ ТРАТА ПРОЦЕССОРНОГО ВРЕМЕНИ В ЦИКЛАХ ПРОЛОГА
- ВОЗМОЖНОСТЬ ВОЗНИКНОВЕНИЯ ТУПИКОВЫХ СИТУАЦИЙ ПРИ ПРИОРИТЕТНОМ ПЛАНИРОВАНИИ

L

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

H

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



# СЕМАФОРЫ ДЕЙКСТРЫ (DIJKSTRA)

S – семафор – целая разделяемая переменная с неотрицательными значениями

При создании может быть инициализирована любым неотрицательным значением

## ДОПУСТИМЫЕ АТОМАРНЫЕ ОПЕРАЦИИ

- P(S): пока  $S == 0$  ПРОЦЕСС БЛОКИРУЕТСЯ;

$$S = S - 1$$

- V(S):  $S = S + 1$

# ПРОБЛЕМА PRODUCER-CONSUMER

Producer:

```
while (1) {  
    produce_item();  
    put_item();  
}
```

Consumer:

```
while (1) {  
    get_item();  
    consume_item();  
}
```

Информация передается через буфер конечного  
размера – N



# ПРОБЛЕМА PRODUCER-CONSUMER

## Решение с помощью семафоров

```
Semaphore mut_ex = 1;  
Semaphore full = 0;  
Semaphore empty = N;
```

Producer:

```
while (1) {  
    produce_item();  
    P(empty);  
    P(mut_ex);  
    put_item();  
    V(mut_ex);  
    V(full);  
}
```

Consumer:

```
while (1) {  
    P(full);  
    P(mut_ex);  
    get_item();  
    V(mut_ex);  
    V(empty);  
    consume_item();  
}
```

# МОНИТОРЫ ХОРА (HOARE)

## Структура

```
Monitor monitor_name {  
    Описание переменных;  
    void m1(...) { ... }  
    void m2(...) { ... }  
    ...  
    void mn(...) { ... }  
    Блок инициализации переменных;  
}
```



# МОНИТОРЫ ХОРА (HOARE)

Условные переменные (condition variables)

Condition C;

- Процесс, выполнивший операцию wait над условной переменной, всегда блокируется
- C.SIGNAL

Выполнение операции signal приводит к разблокированию только одного процесса, ожидающего этого (если он существует)

Процесс, выполнивший операцию signal, немедленно покидает монитор

# ПРОБЛЕМА PRODUCER-CONSUMER

## Решение с помощью мониторов

```
Monitor PC {
    Condition full, empty;
    int count;

    void put () {
        if (count == N) full.wait;
        put_item(); count++;
        if (count == 1) empty.signal;
    }

    void get () {
        if (count == 0) empty.wait;
        get_item(); count--;
        if (count == N-1) full.signal;
    }

    { count = 0; }
}
```

Producer:

```
while (1) {
    produce_item();
    PC.put ();
}
```

Consumer:

```
while (1) {
    PC.get ();
    consume_item();
}
```



# СООБЩЕНИЯ

## ПРИМИТИВЫ ДЛЯ ОБМЕНА ИНФОРМАЦИЕЙ МЕЖДУ ПРОЦЕССОРАМИ

- Для ПЕРЕДАЧИ ДАННЫХ:

SEND (ADDRESS, MESSAGE)

БЛОКИРУЕТСЯ ПРИ ПОПЫТКЕ ЗАПИСИ В  
ЗАПОЛНЕННЫЙ БУФЕР

- Для ПРИЕМА ДАННЫХ

RECEIVE (ADDRESS, MESSAGE)

БЛОКИРУЕТСЯ ПРИ ПОПЫТКЕ ЧТЕНИЯ ИЗ

ПУСТОГО БУФЕРА

Обеспечивают взаимоисключения при работе с  
буфером

# ПРОБЛЕМА PRODUCER-CONSUMER

Решение с помощью сообщений

Producer:

```
while (1) {  
    produce_item();  
    send (address, item)  
}
```

Consumer:

```
while (1) {  
    receive (address,item);  
    consume_item()  
}
```