

# Архитектура операционных систем

Лекция 1.7

# Эквивалентность семафоров, мониторов и сообщений

## Реализация мониторов через семафоры

Semaphore mut\_ex = 1; /\* Для организации взаимного исключения \*/

При входе в монитор

```
void mon_enter (void){  
    P(mut_ex);  
}
```

При нормальном выходе из монитора

```
void mon_exit (void){  
    V(mut_ex);  
}
```

Semaphore c<sub>i</sub> = 0; int f<sub>i</sub> = 0; /\* Для каждой условной переменной \*/

Для операции wait

```
void wait (i){  
    fi += 1;  
    V(mut_ex); P(ci);  
    fi -= 1;  
}
```

Для операции signal

```
void signal_exit (i){  
    if (fi) V(ci);  
    else V(mut_ex);  
}
```

# Эквивалентность семафоров, мониторов и сообщений

## Реализация сообщений через семафоры

Для каждого процесса: Semaphore  $c_i = 0$ ;  
Semaphore  $c_j = 0$ ;

Один на всех: Semaphore  $mut\_ex = 0$ ;

Чтение

P( $mut\_ex$ )

Есть msg?

- нет — встать в очередь
  - V( $mut\_ex$ )
  - P( $c_i$ )
- да — прочитать
  - есть кто на запись?
    - нет — V( $mut\_ex$ )
    - да — удалить
      - V( $c_j$ )

буфер



Очередь на чтение



Очередь на запись



# Эквивалентность семафоров, мониторов и сообщений

## Реализация сообщений через семафоры

Для каждого процесса: Semaphore  $c_i = 0$ ;  
Semaphore  $c_j = 0$ ;

Один на всех: Semaphore  $mut\_ex = 0$ ;

Запись

$P(mut\_ex)$

Есть место?

- нет – встать в очередь
  - $V(mut\_ex)$
  - $P(c_i)$
- да – записать
  - есть кто на чтение?
    - нет –  $V(mut\_ex)$
    - да – удалить
      - $V(c_j)$

буфер



Очередь на чтение



Очередь на запись



# Эквивалентность семафоров, мониторов и сообщений

## Реализация семафоров через мониторы

```
Monitor sem {  
    int count;  
    Condition ci;      /* для каждого процесса */  
    очередь для ожидающих процессов;  
    void P(void){  
        if (count == 0) { добавить себя в очередь;  
            ci.wait;  
        }  
        count = count - 1;  
    }  
    void V(void){  
        count = count + 1;  
        if(очередь не пуста) { удалить процесс Pj из очереди;  
            cj.signal;  
        }  
    }  
    { count = N; }  
}
```



# Реализация семафоров через сообщения

P<sub>m</sub>



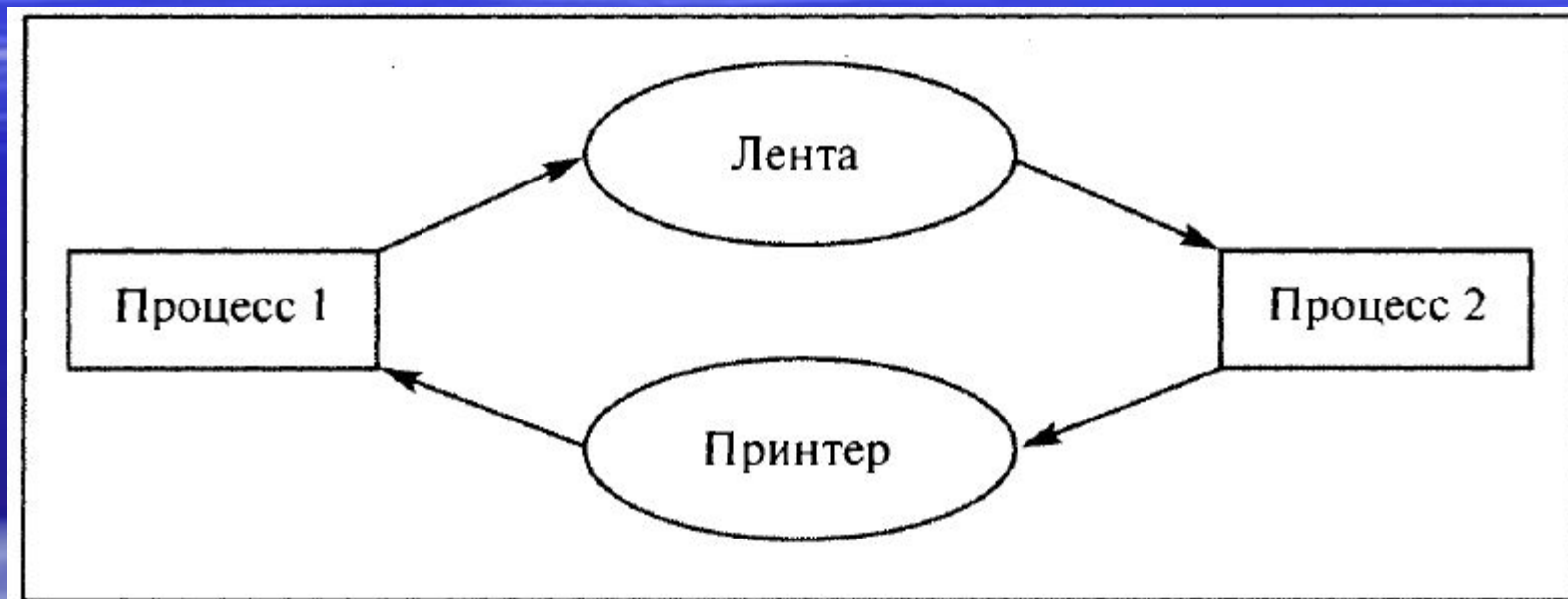
```
V: send (A, "V,Pm");  
    receive (Pm, msg);
```



```
send(Pi, msg);
if(есть ждущие){
    удалить из очереди;
    send (Pk, msg); }
else count = count+1;
```

}

# Тупики





# Условия возникновения тупиков

- 1 Взаимоисключения
- 2 Ожидания ресурсов
- 3 Неперераспределяемости
- 4 Кругового ожидания

# Основные направления борьбы с тупиками

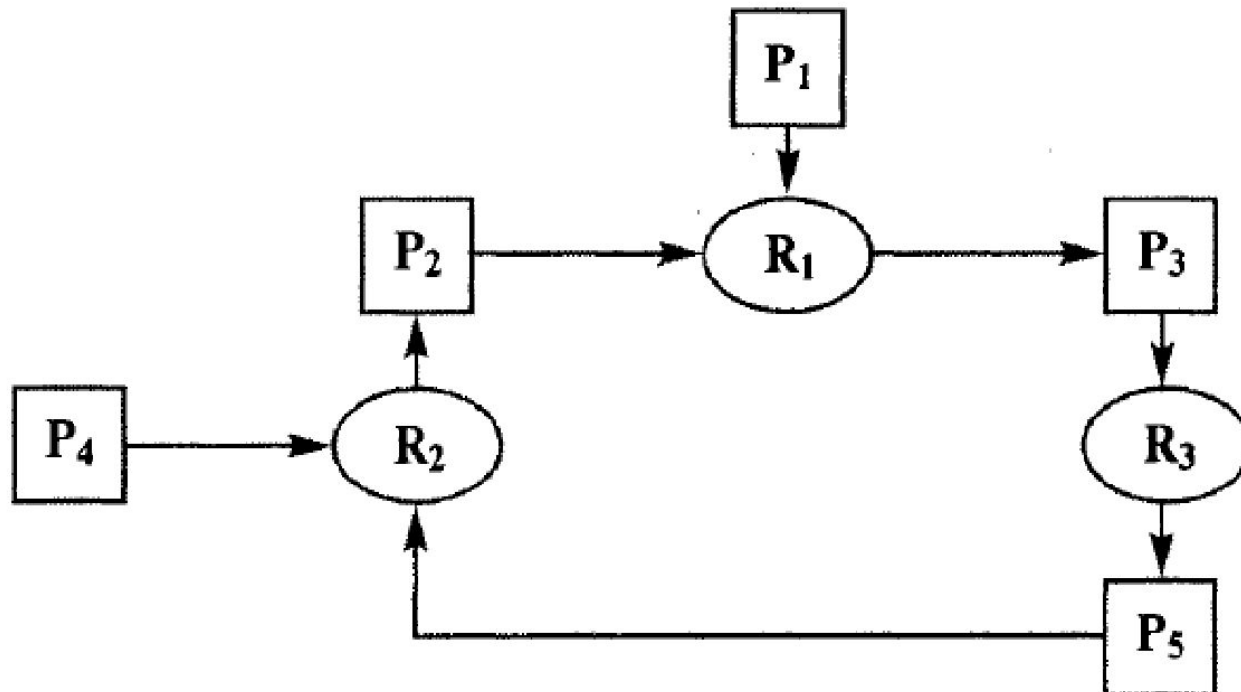
1 Игнорирование проблемы в целом

2 Предотвращение тупиков

3 Обнаружение тупиков

4 Восстановление после тупиков

<b>Пользователи</b>	<b>Максимальная потребность в ресурсах</b>	<b>Выделенное пользователям количество ресурсов</b>
<b>Первый</b>	9	6
<b>Второй</b>	10	2
<b>Третий</b>	3	1



# Управление памятью



# Иерархия памяти

Стоимость  
одного бита

Регистр  
ы

Время доступа

Объем

Кэш

*Управляется менеджером памяти*

*Управляется ОС*

# Принцип локальности

Большинство реальных программ в течение некоторого отрезка времени работает с небольшим набором адресов памяти – это *принцип локальности*

Принцип локальности связан с особенностями человеческого мышления

# Проблема разрешения адресов

Человеку свойственно символическое мышление.

Адреса (имена) переменных описываются идентификаторами, формируя символическое адресное пространство

Как ? ↓ Когда ?

Оперативная физическая память может быть представлена в виде массива ячеек с линейными адресами.

Совокупность всех доступных физических адресов в вычислительной системе – это ее физическое адресное пространство

# Связывание адресов



# Логическое адресное пространство

Символьное адресное пространство – совокупность всех допустимых идентификаторов переменных



Логическое адресное пространство – совокупность всех допустимых адресов, с которыми работает процессор



Физическое адресное пространство – совокупность всех доступных физических адресов в вычислительной системе



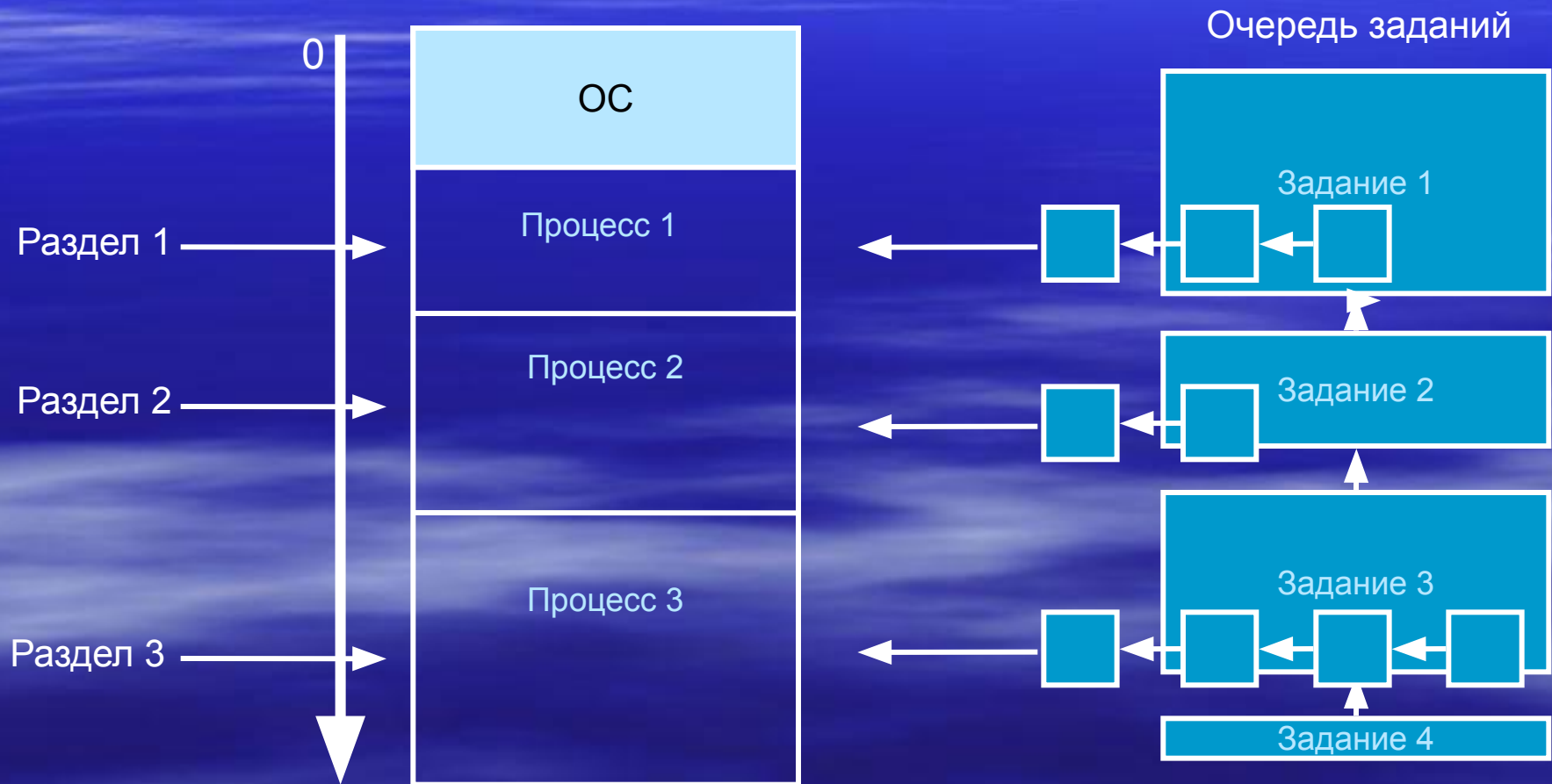
# Функции ОС и hardware для управления памятью

- Отображение логического адресного пространства процесса на физическое адресное пространство
- Распределение памяти между конкурирующими процессами
- Контроль доступа к адресным пространствам процессов
- Выгрузка процессов (целиком или частично) во внешнюю память
- Учет свободной и занятой памяти

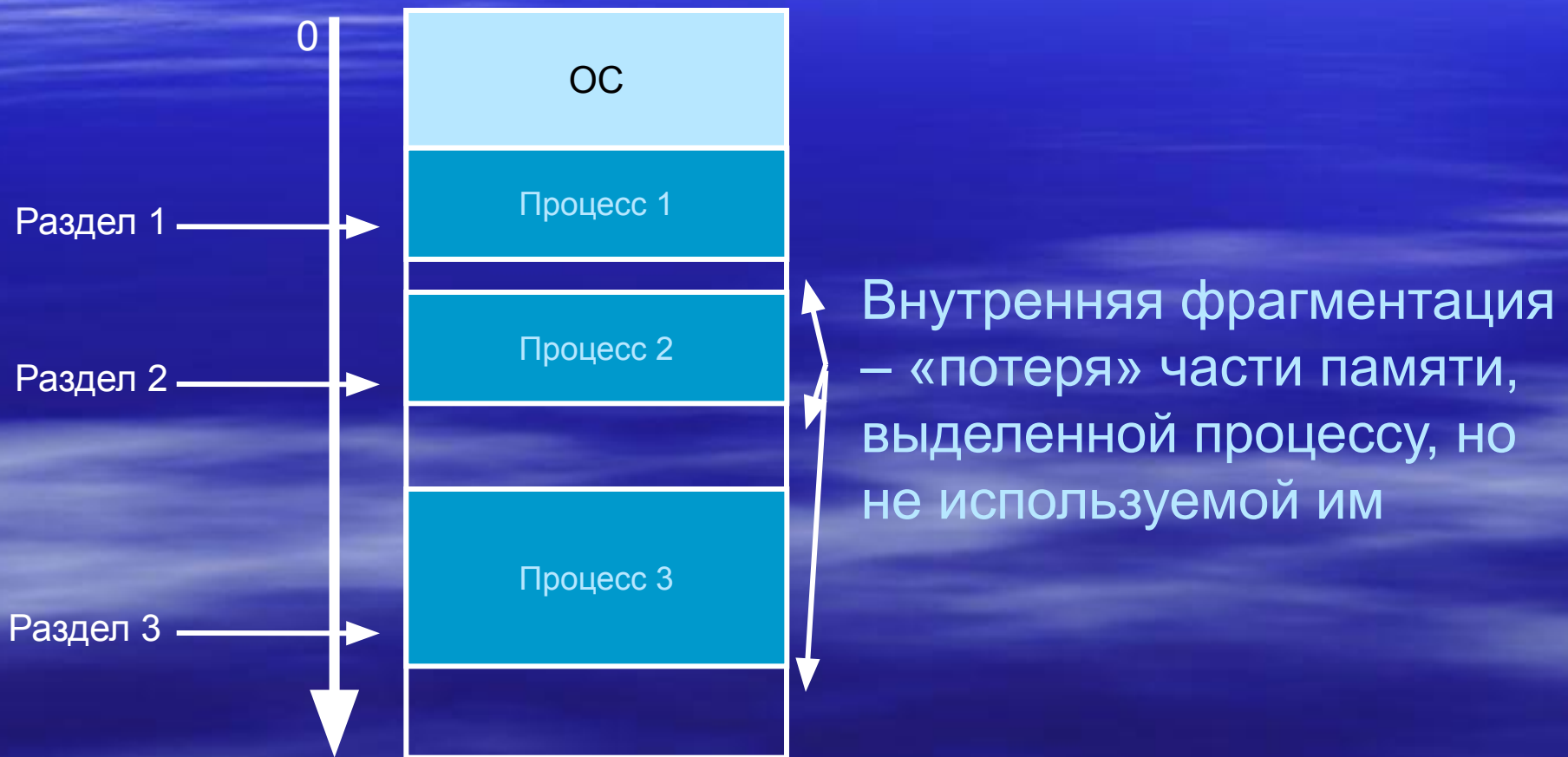
# Однопрограммная вычислительная система



# Схема с фиксированными разделами



# Внутренняя фрагментация



# Способы организации больших программ

- Оверлейная структура

Программа разбивается на несколько частей. Постоянно в памяти находится только загрузчик оверлеев, небольшое количество общих данных и процедур, а части загружаются по очереди

- Динамическая загрузка процедур

Процедуры загружаются в память только по мере необходимости, после обращения к ним

Оба способа основаны на применении  
принципа локальности