ДИСЦИПЛИНА	Операционные системы
	(полное наименование дисциплины без сокращений)
ИНСТИТУТ	Институт информационных технологий
КАФЕДРА	информационных технологий в атомной энергетике
	(полное наименование кафедры)
ВИД УЧЕБНОГО МАТЕРИАЛА	Лекция
	(в соответствии с пп 1-11)
ПРЕПОДАВАТЕЛЬ	Пугачев Андрей Васильевич
	(фамилия, имя, отчество)
СЕМЕСТР	IV семестр 2024 – 2025 учебный год
	(указать семестр обучения, учебный год)

Тема № 3: ”Взаимоисключения и взаимолокировки”

Москва. 2024-2025 у.г.

Механизмы взаимоисключения

Задача «потребитель-производитель»

```
#include <pthread.h>
#include <stdio.h>

#define COUNT (5000000)
volatile unsigned int ret =0;
static void* func1(void* arg )
{
  int i;
  for ( i = 0 ; i < COUNT ; i++ ) ret++;
  return NULL ;
}
static void* func2(void* arg )
{
  int i;
  for ( i=0 ; i < COUNT ; i++ ) ret--;
  return NULL ;
}

int main(int argc, char* argv[])
{
      pthread_t thread;
      void* retv ;

      printf ("COUNT=0x%X\n",COUNT);
      pthread_create(&thread,NULL,func1, NULL);
      func2( NULL );
      pthread_join ( thread ,& retv ) ;
      printf ( "RET=0x%X\n", ret ) ;
      return ret ;
}
```

Простая программа

```
# count=10 ; while [ "$count" != "0" ] ; do \
> /tmp/1.exe ; let count=count-1 ; done
COUNT=0x4C4B40      COUNT=0x4C4B40
RET=0xB7796000      RET=0xB7745000
COUNT=0x4C4B40      COUNT=0x4C4B40
RET=0xB7796000      RET=0xB7759000
COUNT=0x4C4B40      COUNT=0x4C4B40
RET=0xB7738000      RET=0xB775F000
COUNT=0x4C4B40      COUNT=0x4C4B40
RET=0xB76D7000      RET=0xB7708000
COUNT=0x4C4B40      COUNT=0x4C4B40
RET=0xB7712000      RET=0xB772D000
```

Простая программа

```
....... ! ;*******************************************
....... ! ; function func1 (local)
....... ! ;*******************************************
....... ! func1:                          ;xref o80485ed
....... !   push        ebp
8048569 !   mov         ebp, esp
804856b !   sub         esp, 10h
804856e !   mov         dword ptr [ebp-4], 0
8048575 !   jmp         loc_8048588
8048577 !
....... ! loc_8048577:                    ;xref j804858f
....... !   mov         eax, [ret]
804857c !   add         eax, 1
804857f !   mov         [ret], eax
8048584 !   add         dword ptr [ebp-4], 1
8048588 !
....... ! loc_8048588:                    ;xref j8048575
....... !   cmp         dword ptr [ebp-4], 4c4b3fh
804858f !   jng         loc_8048577
8048591 !   mov         eax, 0
8048596 !   leave
8048597 !   ret
8048598 !
```

```
....... ! ;*******************************************
....... ! ; function func2 (local)
....... ! ;*******************************************
....... ! func2:                          ;xref c8048610
....... !   push        ebp
8048599 !   mov         ebp, esp
804859b !   sub         esp, 10h
804859e !   mov         dword ptr [ebp-4], 0
80485a5 !   jmp         loc_80485b8
80485a7 !
....... ! loc_80485a7:                    ;xref j80485bf
....... !   mov         eax, [ret]
80485ac !   sub         eax, 1
80485af !   mov         [ret], eax
80485b4 !   add         dword ptr [ebp-4], 1
80485b8 !
....... ! loc_80485b8:                    ;xref j80485a5
....... !   cmp         dword ptr [ebp-4], 4c4b3fh
80485bf !   jng         loc_80485a7
80485c1 !   mov         eax, 0
80485c6 !   leave
80485c7 !   ret
80485c8 !
```

Определение

Асинхронные параллельные потоки — потоки, которые существуют в системе одновременно и выполняются независимо друг от друга, но периодически должны синхронизироваться и взаимодействовать.

Методы доступа к общим данным

- параллельный;
- последовательный.

Определение

Состояние гонки — ситуация, при которой два или более асинхронных потока или процесса пытаются получить доступ к общему ресурсу.

Критическая секция — область программного кода, в которой возникает состояние гонки.

Определение

Асинхронные потоки (процессы) должны согласовывать свои действия при входе в критическую секцию, и при выходе из нее, исключая возможность появления состояния гонки. Это и называется *взаимоисключением потоков (процессов)*.

Виды взаимоисключений

- алгоритмические;
- аппаратные;
- программные.

Алгоритмические взаимоисключения

- Алгоритм Деккера;
- Алгоритм Петерсона;
- Алгоритм Лампорта.

Архитектурные взаимоисключения

Архитектурные взаимоисключения

- запрет прерываний;
- команды типа «test-and-set»;
- команды обмена.

Программные взаимоисключения

Семафоры

Определение

Семафор - объект операционной системы, ограничивающий выполнение в критической секции более чем для N процессов (потоков).

procedure P(S)
    if $S > 0$ **then**
        $S \leftarrow (S - 1)$;
    else
        Поток в очередь
    end if
end procedure

procedure V(S)
    if Очередь не пустая **then**
        Выбираем новый процесс
    else
        $S \leftarrow (S + 1)$;
    end if
end procedure

Виды семафоров

мьютекс - одноместный семафор.

фьютекс - специальная реализация семафора в Linux.

Простая программа

```
#include <pthread.h>
#include <stdio.h>

#define COUNT (5000000)
volatile unsigned int ret =0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

static void* func1(void* arg )
{
  int i;
  for ( i = 0 ; i < COUNT ; i++ ) {
       pthread_mutex_lock( &mutex );
       ret++;
       pthread_mutex_unlock( &mutex );
  }
  return NULL ;
}
static void* func2(void* arg )
{
  int i;
  for ( i=0 ; i < COUNT ; i++ ) {
       pthread_mutex_lock( &mutex );
       ret--;
       pthread_mutex_unlock( &mutex );
  }
  return NULL ;
}
```

Взаимоблокировки

Определение

Взаимоблокировка (тупиковая ситуация) — состояние системы, при котором хотя бы один процесс (поток) вынужден ожидать событие, которое никогда не наступит.

Основная причина

Конкурентная борьба за право обладания выделяемыми ресурсами!

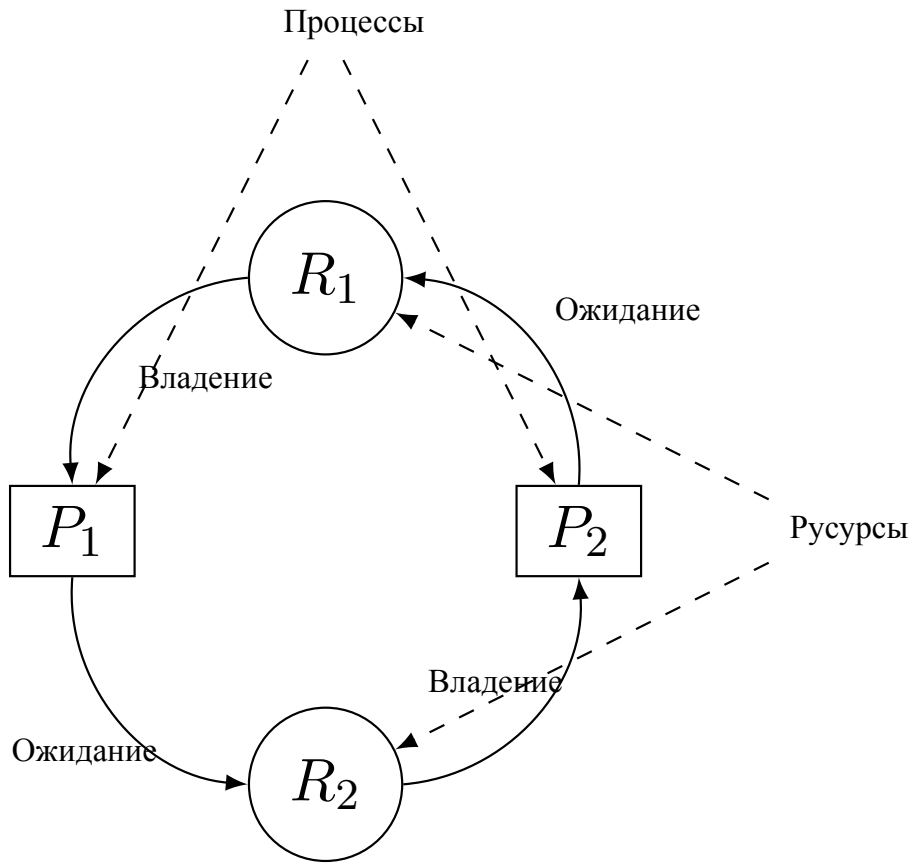
```c
#include <pthread.h>
#include <stdio.h>

#define COUNT (5000000)
volatile unsigned int ret1 = 0, ret2 = 0;
pthread_mutex_t mutex1, mutex2;
static void* func1(void* arg )
{
     int i;
     for ( i = 0 ; i < (int)arg ; i++ ) {
          pthread_mutex_lock( &mutex1 );  ret1++;
          pthread_mutex_lock( &mutex2 );  ret2--;
          pthread_mutex_unlock( &mutex1 );
          pthread_mutex_unlock( &mutex2 );
     } return NULL ;
}
int main(int argc, char* argv[])
{
     pthread_t thread;
     void* retv ;
     int i;
     pthread_mutex_init(&mutex1,NULL);
     pthread_mutex_init(&mutex2,NULL);
     pthread_create(&thread,NULL,func1, (void*)COUNT);
     for ( i = 0 ; i < COUNT; i++ ) {
          pthread_mutex_lock( &mutex2 );  ret2++;
          pthread_mutex_lock( &mutex1 );  ret1--;
          pthread_mutex_unlock( &mutex2 );
          pthread_mutex_unlock( &mutex1 );
     }
     pthread_join ( thread,& retv ) ;
     printf ( "RET:␣0x%X;␣0x%X\n", ret1, ret2 ) ;
     return 0 ;
}
```

```c
#include <pthread.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
	int i, ret = 10;
	pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
	for ( i = 0 ; i < 10 ; i++ ) {
		pthread_mutex_lock( &mutex );
		if ( (--ret) == 7 ) continue;
		printf("%d ",ret);
		pthread_mutex_unlock( &mutex );
	}
	return 0;
}
```

Круговое ожидание



Концепция ресурсов

Возможность совместного использования

- перераспределяемые;
- неперераспределяемые.

Возможность повторного использования

- реентерабельностые;
- нереентерабельностые.

Условия возникновения взаимоблокировки

1. Использование механизмов взаимоисключения.
2. Ожидание дополнительных ресурсов.
3. Неперераспределяемость занятых и ожидаемых ресурсов;
4. Кругового ожидания.

Решения проблемы взаимоблокировки

1. Предотвращение (deadlock prevention);
2. Обход (deadlock avoidance);
3. Обнаружение (deadlock detection);
4. Восстановление после взаимоблокировок.

Предотвращение взаимоблокировки

Хавендер показал, что возникновение взаимоблокировки невозможно при нарушении хотя бы одно из описанных ранее условий.

Стратегии предотвращения взаимоблокировок

- запрос всех необходимых ресурсов;
- добровольное освобождение ресурсов;
- линейное упорядочивание ресурсов.

Запрос всех необходимых ресурсов

Цель

Нарушение условия ”Ожидание дополнительных ресурсов”

Недостатки

- нерациональное использование ресурсов;
- вероятность возникновение бесконечного откладывания.

Добровольное освобождение ресурсов

Цель

Нарушение условия ”неперераспределяемости”

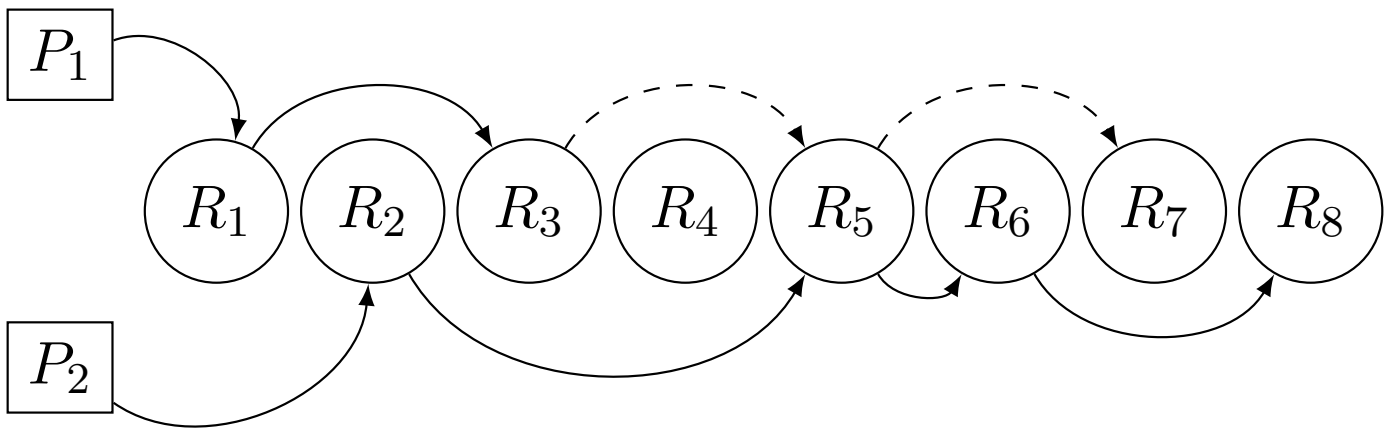
Недостатки

- возможна потеря данных.

Линейное упорядочивание ресурсов

Принцип

1. Все ресурсы нумеруются.
2. При необходимости получения нескольких ресурсов, доступ к ресурсу с большим номеров возможен только после получения доступа к ресурсам с меньшими номерами.

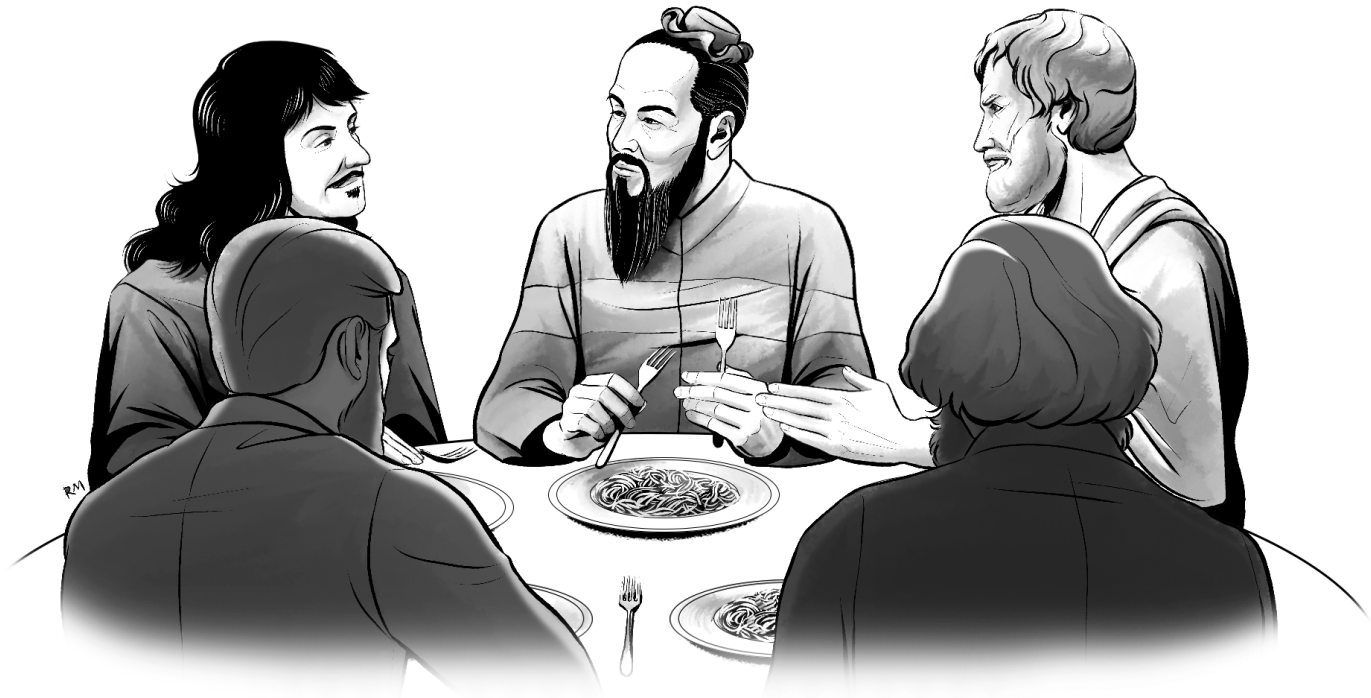


Линейное упорядочивание ресурсов

Недостатки:

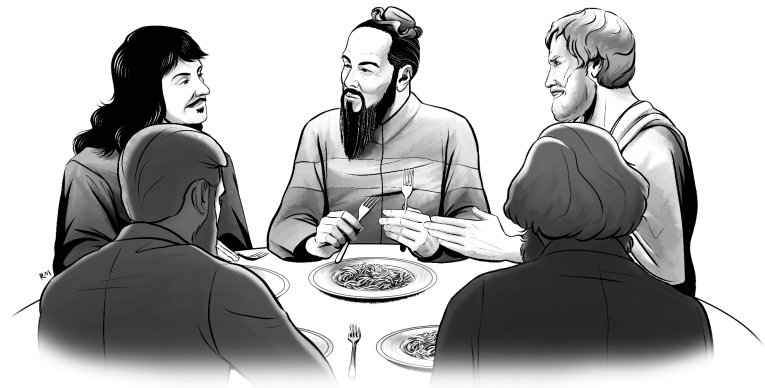
- невозможен произвольный порядок использования ресурсов;
- необходимо сохранение нумерации ресурсов в течение долгого времени;
- плохая масштабируемость.

Проблема обедающих философов



Проблема обедающих философов

Проблема обедающих философов



- запрос всех необходимых ресурсов;
- добровольное освобождение ресурсов;
- линейное упорядочивание ресурсов.

Обедающие философы

Бесконечное откладывание (ресурсное голодания) — ситуация, при которой предоставление процессу некоторого ресурса будет откладываться на неопределённо долгий срок, в то время как система будет уделять внимание другим процессам.

Обход взаимоблокировок

Алгоритм Банкира

Основные обозначения

t - число ресурсов в системе.

n - число процессов в системе.

$max(P_i)$ - максимальное число ресурсов для процесса P_i.

$loan(P_i)$ - число ресурсов уже выделенное процессу.

$claim(P_i)$ - число ресурсов, которые понадобятся процессу.

a - оставшееся число ресурсов в системе.

$$claim(P_i) = max(P_i) - loan(P_i), \forall i \in [1, ..., n]$$

$$a = t - \sum_{i=1}^{n} loan(P_i)$$

Состояния системы

Безопасное состояние - состояние системы, при котором у каждого процесса в системе есть возможность нормального завершения работы.

Небезопасное состояние - состояние системы, при котором возможно возникновение взаимоблокировки.

Безопасное состояние

$$\exists i \in [1, ..., n], a \geq claim(P_i)$$

Пример

Безопасное состояние ($t = 12$)

Процесс	max	load	claim
1	4	1	3
2	6	4	2
3	8	5	3

Пример

Небезопасное состояние ($t = 12$)

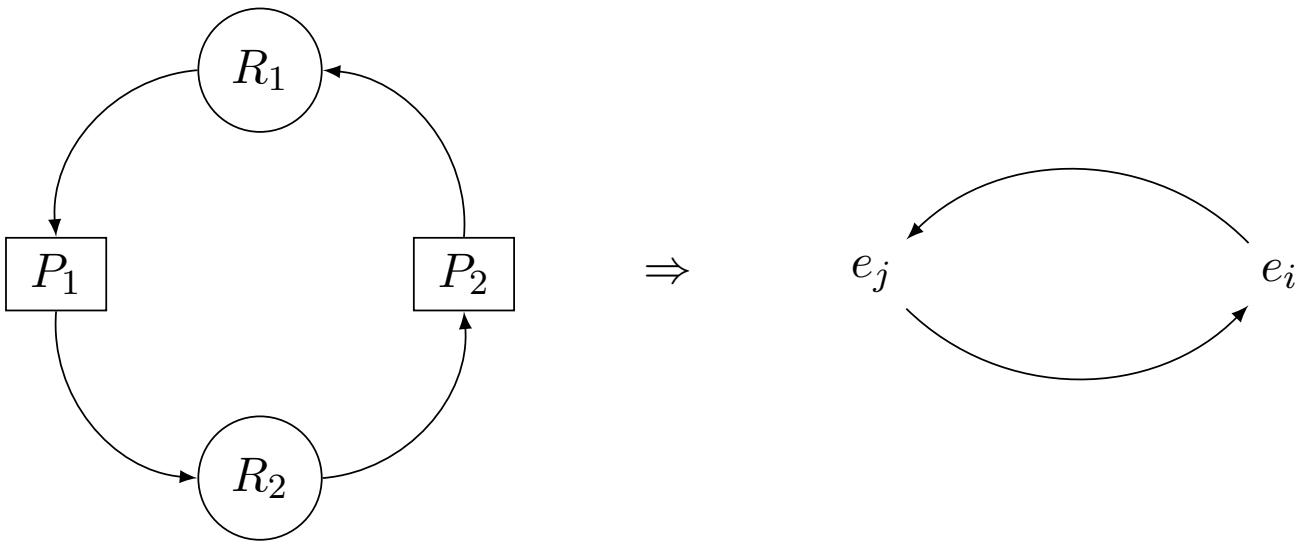
Процесс	max	load	claim
1	4	2	2
2	6	4	2
3	8	5	3

Особенности

- фиксированное количество ресурсов;
- фиксированное количество процессов;
- не подходит для систем реального времени;
- необходимость возврата ресурса в конечный период времени;
- необходимость заранее указывать максимальное количество ресурсов.

Обнаружение взаимоблокировок

Круговое ожидание и граф распределения ресурсов



- m – общее число различных классов ресурсов в системе;
- n – общее число процессов, функционирующих в настоящей момент в системе;
- вектор $T = (t_1, \ldots, t_m)$, где t_i – общее число ресурсов i-го класса;
- матрица $C = \{c_{i,j}\}_{n \times m}$ – матрица текущего распределения ресурсов, где значение $c_{i,j}$ определяет число экземпляров ресурса j-го класса, занятых i-ым процессом;
- матрица $R = \{r_{i,j}\}_{n \times m}$ – матрица текущего состояние запросов на получение ресурсов, где значение $r_{i,j}$ определяет число экземпляров ресурса j-го класса, которое запросил i – ый процесс;
- вектор $A = (a_1, \ldots, a_m)$, где a_j – количество экземпляров ресурсов j-го класса, доступных (нераспределенных) в текущий момент времени.

1. В матрице R выбирается i-я строка, удовлетворяющая следующему условию.

$$\begin{cases} \sum\limits_{j=1}^{m} c_{i,j} \neq 0, \\ \sum\limits_{j=1}^{m} r_{i,j} \neq 0, \\ r_{i,j} \leq a_j, \forall j \in 1, \ldots, m \end{cases}$$

   если строка найдена – перейти к шагу 2, если нет – алгоритм завершен.
2. Прибавить к элементам вектора A соответствующие элементы i-ой строки матрицы C. Перейти к шагу 3.
3. Все элементы i-ой строки матрицы R и матрицы C обнулить. Перейти к шагу 1.

Вопросы?