

Взаимное исключение критических интервалов

Взаимное исключение критических интервалов

При взаимодействии через общую память нити должны синхронизовать свое выполнение.

Thread0: $pi = pi + val$; && Thread1: $pi = pi + val$;

Время	Thread 0	Thread 1
1	LOAD R1,pi	
2	LOAD R2,val	
3	ADD R1,R2	LOAD R3,pi
4	STORE R1,pi	LOAD R4,val
5		ADD R3,R4
6		STORE R3,pi

Результат зависит от порядка выполнения команд. Требуется взаимное исключение критических интервалов.

Взаимное исключение критических интервалов в однопроцессорной ЭВМ.

1. Блокировка внешних прерываний (может нарушаться управление внешними устройствами, возможны внутренние прерывания при работе с виртуальной памятью).
2. Блокировка переключения на другие процессы (MONO, MULTI).

```
lock → while(1){
        disable interrupts ()
        critical section
unlock → enable interrupts ()
        other code
}
```

Программные решения

Shared

```
int turn=1;
```

Process 1

```
while(1){  
    while(turn == 2); // lock  
    critical section  
    turn = 2; // unlock  
    other code  
}
```

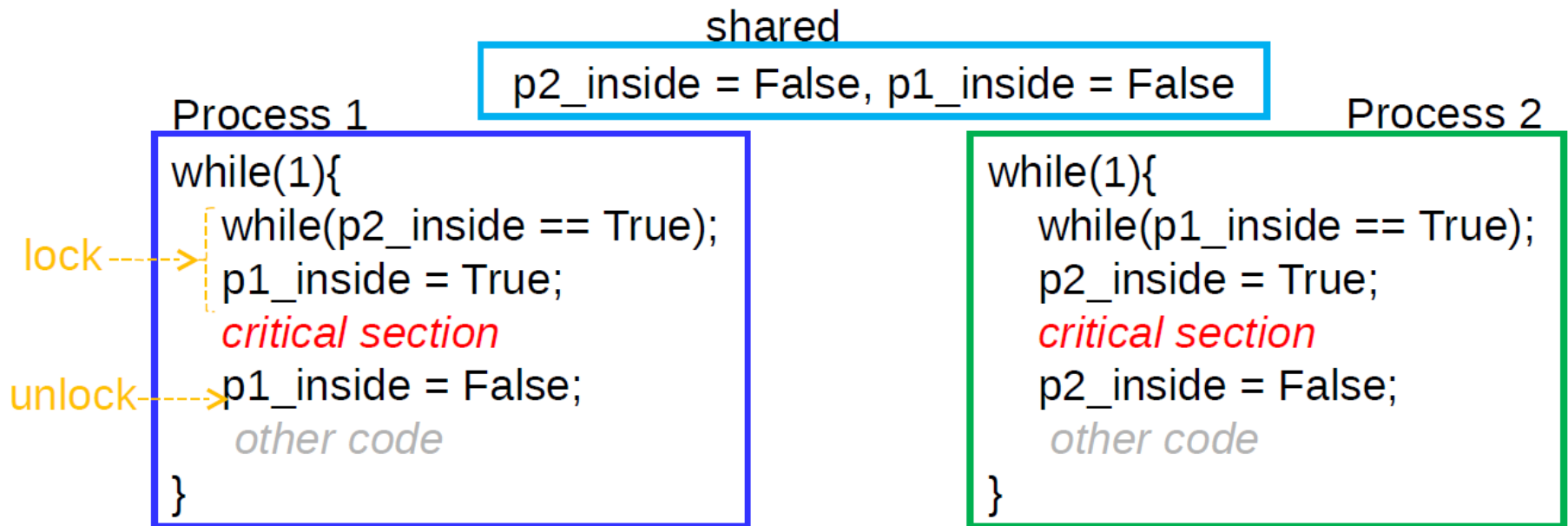
Process 2

```
while(1){  
    while(turn == 1); // lock  
    critical section  
    turn = 1; // unlock  
    other code  
}
```

Требования

- в любой момент времени только один процесс может находиться внутри критического интервала;
- если ни один процесс не находится в критическом интервале, то любой процесс, желающий войти в критический интервал, должен получить разрешение без какой либо задержки;
- ни один процесс не должен бесконечно долго ждать разрешения на вход в критический интервал (если ни один процесс не будет находиться внутри критического интервала бесконечно долго);
- не должно существовать никаких предположений о скоростях процессоров.

Программные решения



Программные решения

globally defined

p2_wants_to_enter, p1_wants_to_enter

Process 1

Process 2

```
while(1){
  lock-----> { p1_wants_to_enter = True
                while(p2_wants_to_enter = True);
                } critical section
  unlock-----> p1_wants_to_enter = False
                other code
}
```

```
while(1){
  p2_wants_to_enter = True
  while(p1_wants_to_enter = True);
  critical section
  p2_wants_to_enter = False
  other code
}
```

Алгоритм Деккера (1968)

```
int turn;
boolean flag[2 ];
proc( int i )
{
    while (TRUE)
    {
        <вычисления>;
        enter_region( i );
        <критический интервал>;
        leave_region( i );
    }
}
```


Алгоритм Деккера (1968)

```
void enter_region( int i )
{
try: flag[ i ]=TRUE;
    while (flag [( i+1 ) % 2])
    {
        if ( turn == i ) continue;
        flag[ i ] = FALSE;
        while ( turn != i ); /* Цикл ожидания*/
        goto try;
    }
}

void leave_region( int i )
{
    turn = ( i +1 ) % 2;
    flag[i ] = FALSE;
}

turn = 0; flag[ 0 ] = FALSE; flag[ 1 ] = FALSE;
proc( 0 ) AND proc( 1 ) /* запустили 2 процесса */
```

Алгоритм Петерсона (1981)

```
void enter_region( int i )
{
    int other;           /* номер другого процесса */
    other = 1 - i;
    flag[ i ] = TRUE;
    turn = i;
    while (turn == i && flag[ other ] == TRUE) /* пустой оператор */;
}
```

```
void leave_region( int i )
{
    flag[ i ] = FALSE;
}
```

Программное решение

```
while(1){  
    while(lock != 0);  
    lock= 1; // lock  
    critical section  
    lock = 0; // unlock  
    other code  
}
```

Использование неделимой операции TEST_and_SET_LOCK

TSL(r,s): [r = s; s = 1]

enter_region:

```
    tsl reg, flag
    cmp reg, #0      /* сравниваем с нулем */
    jnz enter_region /* если не нуль - повторяем попытку */
    ret
```

leave_region:

```
    mov flag, #0    /* присваиваем нуль*/
    ret
```

xchg reg, mem

Семафоры Дейкстры (1965)

Семафор - неотрицательная целая переменная, которая может изменяться и проверяться только посредством двух функций:

Функция запроса семафора **P(s)**:

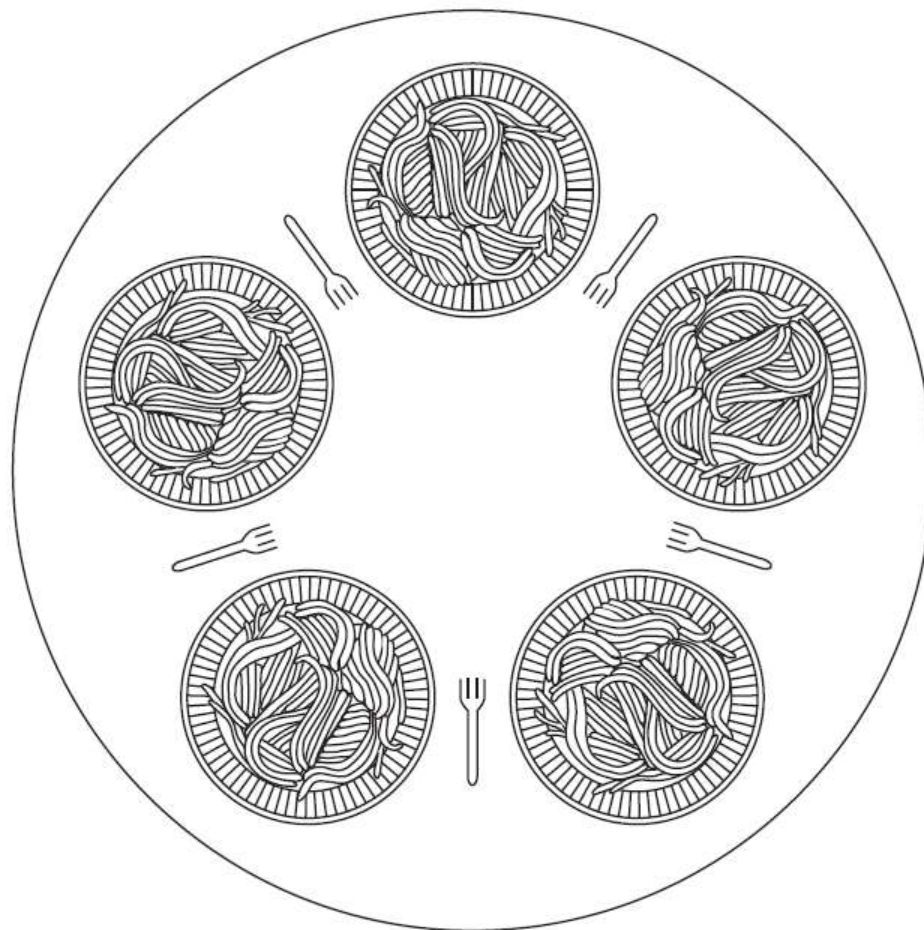
```
[if (s == 0) <заблокировать текущий процесс>; else s = s-1;]
```

Функция освобождения семафора **V(s)**:

```
[if (s == 0) <разблокировать один из заблокированных процессов>;  
s = s+1;]
```

Классические задачи взаимодействия процессов

Обедающие философы



Обедающие философы

```
#define N 5 /* количество философов */
void philosopher(int i) /* i: номер философа (от 0 до 4) */
{
    while (TRUE) {
        think( ); /* философ размышляет */
        take_fork(i); /* берет левую вилку */
        take_fork((i+1) % N); /* берет правую вилку; */
        /* % - оператор деления по модулю */
        eat(); /* ест спагетти */
        put_fork(i); /* кладет на стол левую вилку */
        put_fork((i+1) % N); /* кладет на стол правую вилку */
    }
}
```


Обедающие философы

```
#define N 5 /* количество философов */
semaphore mutex=1;
void philosopher(int i) /* i: номер философа (от 0 до 4) */
{
    while (TRUE) {
        think( ); /* философ размышляет */
        P(mutex);
        take_fork(i); /* берет левую вилку */
        take_fork((i+1) % N); /* берет правую вилку; */
        /* % - оператор деления по модулю */
        eat(); /* ест спагетти */
        put_fork(i); /* кладет на стол левую вилку */
        put_fork((i+1) % N); /* кладет на стол правую вилку */
        V(mutex);
    }
}
```

Обедающие философы

```
#define N 5 /* количество философов */
#define LEFT (i+N-1) %N /* номер левого соседа для i-го философа */
#define RIGHT (i+1) %N /* номер правого соседа для i-го философа */
#define THINKING 0 /* философ размышляет */
#define HUNGRY 1 /* философ пытается взять вилки */
#define EATING 2 /* философ ест спагетти */
int state[N]; /* массив для отслеживания состояния каждого философа */
semaphore mutex = 1; /* Взаимное исключение входа в критическую область */
semaphore s[N]; /* по одному семафору на каждого философа */

void philosopher(int i) /* i – номер философа (от 0 до N-1) */
{
    while (TRUE) { /* бесконечный цикл */
        think(); /* философ размышляет */
        take_forks(i); /* берет две вилки или блокируется */
        eat(); /* ест спагетти */
        put_forks(i); /* кладет обе вилки на стол */
    }
}
```

Обедающие философы

```
void take_forks(int i) /* i – номер философа (от 0 до N-1) */ {
    P(mutex); /* вход в критическую область */
    state[i] = HUNGRY; /* запись факта стремления философа поесть */
    test(i); /* попытка взять две вилки */
    V(mutex); /* выход из критической области */
    P(s[i]); /* блокирование, если вилки взять не удалось */
}

void put_forks(i) /* i – номер философа (от 0 до N-1) */ {
    P(mutex); /* вход в критическую область */
    state[i] = THINKING; /* философ наелся */
    test(LEFT); /* проверка готовности к еде соседа слева */
    test(RIGHT); /* проверка готовности к еде соседа справа */
    V(mutex); /* выход из критической области */
}

void test(i) /* i – номер философа (от 0 до N-1) */
{
    if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
        state[i] = EATING; V(s[i]);
    }
}
```

Читатели и писатели

```
semaphore mutex = 1; /* управляет доступом к 'rc' */
semaphore db = 1; /* управляет доступом к базе данных */
int rc = 0; /* количество читающих или желающих читать процессов */
void reader(void)
{
    while (TRUE) { /* бесконечный цикл */
        P(mutex); /* получение исключительного доступа к 'rc' */
        rc = rc + 1; /* теперь на одного читателя больше */
        if (rc == 1) P(db); /* если это первый читатель... */
        V(mutex); /* завершение исключительного доступа к 'rc' */

        read_data_base( ); /* доступ к данным */

        P(mutex); /* получение исключительного доступа к 'rc' */
        rc = rc - 1; /* теперь на одного читателя меньше */
        if (rc == 0) V(db); /* если это последний читатель... */
        V(mutex); /* завершение исключительного доступа к 'rc' */
        use_data_read(); /* не критическая область */
    }
}
```

Читатели и писатели

```
void writer(void)
{
    while (TRUE) { /* бесконечный цикл */
        think_up_data( ); /* не критическая область */
        P(db); /* получение исключительного доступа */
        write_data_base( ); /* обновление данных */
        V(db); /* завершение исключительного доступа */
    }
}
```

Читатели и писатели

```
semaphore reader=1;  
semaphore writer=1;  
semaphore access=1;  
int rd=0;
```

```
Writer_enter() {  
    P(writer);  
    P(reader);  
}
```

```
Writer_leave() {  
    V(reader);  
    V(writer);  
}
```

```
Reader_enter() {  
    P(writer);  
    P(access);  
    rd++;  
    if(rd==1)P(reader);  
    V(access);  
    V(writer);  
}
```

```
Reader_leave() {  
    P(access);  
    rd--;  
    if(rd==0) V(reader);  
    V(access);  
}
```

Производитель-потребитель

```
semaphore mutex = 1;
```

```
semaphore full = 0;
```

```
semaphore empty = N;
```

```
producer()
```

```
{
```

```
    int item;
```

```
    while (TRUE)
```

```
    {
```

```
        produce_item(&item);
```

```
        P(empty);
```

```
        P(mutex);
```

```
        enter_item(item);
```

```
        V(mutex);
```

```
        V(full)
```

```
    }
```

```
}
```

```
consumer()
```

```
{
```

```
    int item;
```

```
    while (TRUE)
```

```
    {
```

```
        P(full);
```

```
        P(mutex);
```

```
        remove_item(&item);
```

```
        V(mutex);
```

```
        V(empty);
```

```
        consume_item(item);
```

```
    }
```

```
}
```