Основы Операционных Систем

МФТИ-2025

Тема 5

Механизмы синхронизации

Алгоритмы синхронизации

Требования к программным 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)

Алгоритмы синхронизации

Программные – запрет прерываний

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

Обычно используется внутри ОС

Алгоритмы синхронизации

Программные – «переменная-замок»

```
Shared int lock = 0;

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

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

Нарушается условие взаимоисключения

Алгоритмы синхронизации

Программные – «строгое чередование»

Shared int turn = 0;

P_0

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

P_1

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

Условие взаимного исключения выполняется

Нарушается условие прогресса

Алгоритмы синхронизации

Программные – «флаги готовности»

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

P0
while (some condition) {
        ready[0] = 1;
        while (ready[1]);
                critical section
        ready[0] = 0;
                remainder section
}

P1
while (some condition) {
        ready[1] = 1;
        while (ready [0]);
                critical section
        ready[1] = 0;
                remainder section
}
```

1-я часть условия прогресса нарушается

Отступление

Вопрос: является ли атомарной операция x += 1 для переменной int x в ЯП C?
Какие операции вообще можем считать атомарными?

А на наших слайдах по синхронизации?

Алгоритмы синхронизации

Программные – алгоритм Петерсона

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

P_0

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

Аппаратная поддержка

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

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

Аппаратная поддержка

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

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

Механизмы синхронизации

Недостатки программных алгоритмов

- Непроизводительная трата процессорного времени в циклах пролога
- Возможность возникновения тупиковых ситуаций при приоритетном планировании использования процессора

L

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

H

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

Механизмы синхронизации

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

Механизмы синхронизации

Проблема Producer-Consumer

Producer:

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

Consumer:

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

Информация передается через буфер конечного размера – N.

Если в буфере нет места - Producer блокируется. Если в буфере пусто – Consumer блокируется.

Механизмы синхронизации

Проблема Producer-Consumer. Семафоры

```
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();
}
```

Механизмы синхронизации

Producer-Consumer. Семафоры Дейкстры

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

Producer:

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

Consumer:

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

Механизмы синхронизации

Мониторы Хора (Hoare)

Структура

Monitor monitor_name {

Описание внутренних переменных;

void m_1(…) { … }

void m_2(…) { … }

…

void m_n(…) { … }

Блок инициализации переменных;

}

Механизмы синхронизации

Мониторы Хора (Hoare)

Условные переменные (condition variables)

Condition C;

- C.wait
- C.signal

Процесс, выполнивший операцию wait над условной переменной, **всегда** блокируется

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

Процесс, выполнивший операцию signal, **немедленно** покидает монитор

Механизмы синхронизации

Producer-Consumer. Мониторы Хора

```
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();
    }
```

Механизмы синхронизации

Очереди сообщений

Примитивы для обмена информацией между процессами

- 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;

получение сообщений в порядке FIFO.

Обеспечивают взаимоисключения при работе с буфером

Механизмы синхронизации

Producer-Consumer. Очереди сообщений

Producer:

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

Consumer:

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

Эквивалентность механизмов

Реализация мониторов через семафоры

Semaphore mut_ex = 1; /* Для организации взаимоисключения */

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_i = 0; int f_i = 0; /* Для каждой условной переменной */

Для операции **wait**

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

Для операции **signal**

```
void signal_exit (i) {
    if (f_i) V(c_i);
    else V(mut_ex);
}
```

Эквивалентность механизмов

Реализация сообщений через семафоры

```
Semaphore Amut_ex = 1;
Semaphore Afull = 0;
Semaphore Aempty = N;

send (A, msg) {
    P(Aempty);
    P(Amut_ex);
    put_item(A, msg);
    V(Amut_ex);
    V(Afull);
}

receive (A, msg) {
    P(Afull);
    P(Amut_ex);
    get_item(A, msg);
    V(Amut_ex);
    V(Aempty);
}
```

Реализуется модель “Producer-Consumer”

Эквивалентность механизмов

Реализация семафоров через мониторы

```
Monitor sem {
    unsigned int count;
    Condition c_i;        /* для каждого процесса */

    void P(void){
        if (count == 0)   c_i.wait; count = count -1;
    }

    void V(void){
        count = count+1; c_j.signal;
    }

    { count = N; }
}
```

Эквивалентность механизмов

Реализация семафоров через сообщения

Создание и инициализация семафора

```
void Sem_init (int N) {
        int i;
        создать очередь сообщений M;
        for(i = 0; i < N;  i++)  send (M, msg);
}
```

Операция P

```
void Sem_P () {
        receive (M, msg);
}
```

Операция V

```
void Sem_V () {
        send (M, msg);
}
```

Самое время для вопросов…

Прил. 1: ссылка на видео-2022

Ютуб-канал «Дистанционные занятия МФТИ»
Плейлист «Компьютерные технологии», Осень 2022
Лектор Ефанов Н.Н.