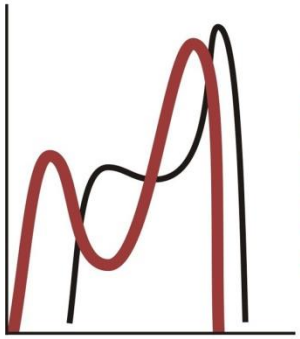

ИНСТИТУТ
МАТЕМАТИКИ
МЕХАНИКИ
КОМПЬЮТЕРНЫХ
НАУК

имени И.И. Воровича —

Архитектура компьютера и операционные системы

Лекция 17. Алгоритмы и механизмы синхронизации

Андреева Евгения Михайловна
доцент кафедры информатики и вычислительного эксперимента

ИНСТИТУТ
МАТЕМАТИКИ
МЕХАНИКИ
КОМПЬЮТЕРНЫХ
НАУК
имени И.И. Воровича

План лекции

- Программные алгоритмы синхронизации
- Аппаратная поддержка взаимоисключений
- Механизмы синхронизации

Требования, предъявляемые к алгоритмам

1. 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)
   - Процессы, которые находятся вне своих критических участков и не собираются входить в них, не могут препятствовать другим процессам входить в их собственные критические участки.
   - Решение не должно приниматься бесконечно долго.
5. Выполняется условие ограниченного ожидания (bound waiting)

Запрет прерываний

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

Обычно используется внутри ОС

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Переменная-замок

Shared int lock = 0;

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

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

Нарушается условие взаимоисключения

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

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
}
```

Нарушается условие прогресса

Условие взаимоисключения выполняется

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Флаги готовности

Shared int ready[2] = {false, false};

P_0

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

P_1

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

2-я часть условия прогресса нарушается

1-я часть условия прогресса выполняется

Условие взаимоисключения выполняется

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Алгоритм Петерсона

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

P_0

```
while (some condition) {
    ready[0] = true;
    turn = 1;
    while (ready[1] && turn == 1);
        critical section
    ready[0] = false;
        remainder section
}
```

P_1

```
while (some condition) {
    ready[1] = true;
    turn = 0;
    while (ready [0] && turn == 0);
        critical section
    ready[1] = false;
        remainder section
}
```

Все 5 условий выполняются

Алгоритм Лампорта

```
ready: array [1..N] of  bool = {false};
turn: array [1..N] of integer = {0};
lock(integer i) {
        ready[i] = true;
        turn[i] = 1 + max(turn [1], ..., turn [N]);
        ready[i] = false;
         for (integer j = 1; j <= N; j++) {
                // Ждём пока поток j получит свой номер:
                while (ready[j]) ;
                // Ждём пока все потоки с меньшим номером или с таким же номером,
                // но с более высоким приоритетом, закончат свою работу:
                while ((turn [j] != 0) && ((turn [j], j) < (turn [i], i))) ;
        }
 }
unlock(integer i) {
                 turn [i] = 0;
}
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

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
}
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

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
}
```

Особенности аппаратной реализации

- Преимущества
  - Простота.
  - Применимость к любому количеству процессов и процессоров с общей памятью.
  - Поддержка нескольких критических разделов.

- Недостатки
  - Использование пережидания.
  - Возможность голодания.
  - Возможность взаимной блокировки (процессы с разными приоритетами).

Механизмы синхронизации

- Семафоры
- Мониторы
- Сообщения

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Семафоры Дейкстры (Dijkstra)

- S – семафор – целая разделяемая переменная с неотрицательными значениями
- При создании может быть инициализирована любым неотрицательным значением

- Допустимые атомарные операции
  - down(S): пока S == 0 процесс блокируется;
    S = S - 1
  - up(S): S = S + 1

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Проблема Producer-Consumer

- Информация передается через буфер конечного размера – N
- Производитель помещает информацию в буфер, потребитель извлекает ее оттуда.
- Нельзя помещать данные в буфер, если из него читают
- Нельзя читать из буфера, если в него помещают данные

Producer:

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

Consumer:

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

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Решение с помощью семафоров

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

Producer:

```
while (1) {
        produce_item();
        down(empty);
        down(mut_ex);
        put_item();
        up(mut_ex);
        up(full);
}
```

Consumer:

```
while (1) {
        down(full);
        down(mut_ex);
        get_item();
        up(mut_ex);
        up(empty);
        consume_item();
}
```

Мониторы Хора (Hoare)

- Структура

Monitor monitor_name {

Описание переменных;

void m_1(…) { … }

void m_2(…) { … }

…

void m_n(…) { … }

Блок инициализации переменных;

}

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Мониторы Хора (Hoare)

- Условные переменные (condition variables)
- Condition C;
  - C.wait
    - 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
    - Условные переменные, не умеют запоминать предысторию. Если операция signal выполняется над условной переменной, с которой не связано ни одного заблокированного процесса, то информация о произошедшем событии будет утеряна. Следовательно, выполнение операции wait всегда будет приводить к блокированию процесса.

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

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();
}
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Языки программирования, поддерживающие мониторы

- Ада
- C# (и другие языки, использующие .NET Framework)
- Concurrent Pascal
- D
- Java (с помощью ключевого слова synchronized)
- Mesa
- Модула-3
- Ruby
- Squeak Smalltalk
- uC++ и др.

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

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Решение с помощью сообщений

Producer:

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

Consumer:

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

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Синхронизация в POSIX

- POSIX определяет набор интерфейсов (функций заголовочных файлов) для программирования потоков. Эти рекомендации носят название POSIX threads или Pthreads.
- Подпрограммы, составляющие API Pthreads, могут быть неформально сгруппированы в четыре основные группы:
  - **Управление потоками:** процедуры**,** которые работают непосредственно с потоками - создание, отключение, объединение и т.д.
  - **Мьютексы:** процедуры, которые связаны с синхронизацией. Они обеспечивают создание, уничтожение, блокировку и разблокирование мьютексов. Дополняются функциями, которые устанавливают или изменяют атрибуты, связанные с мьютексами.
  - **Условные переменные:** группа включает функции для создания, уничтожения, ожидания и сигнализации на основе указанных значений переменных. Также включены функции для установки атрибутов условных переменных.
  - **Синхронизация:** процедуры, которые управляют блокировкой чтения и записи, а также барьерами.
- Соглашения об именах: все идентификаторы в библиотеке потоков начинаются с **pthread_** .

Mutex

- int **pthread_mutex_init** (mutex, attr)
- pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER
- int **pthread_mutex_destroy** (mutex)

- int **pthread_mutexattr_init** (attr)
- int **pthread_mutexattr_destroy** (attr)

- int **pthread_mutex_lock** (mutex)
- int **pthread_mutex_trylock** (mutex)
- int **pthread_mutex_unlock** (mutex)

Пример

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
typedef struct{
        double     *a;
        double     *b;
        double     sum;
        int     veclen;
} DOTDATA;
#define NUMTHRDS 4
#define VECLEN 100
DOTDATA dotstr;
pthread_t  callThd[NUMTHRDS];
pthread_mutex_t  mutexsum=PTHREAD_MUTEX_INITIALIZER;
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Пример (продолжение)

```
void *dotprod(void *arg){
        int i, start, end, len;
        long offset;
        double mysum, *x, *y;
        offset = (long)arg;
        len = dotstr.veclen;
        start = offset * len;
        end = start + len;
        x = dotstr.a;
        y = dotstr.b;
        mysum = 0;

        for (i = start; i < end; i++) {
                mysum += (x[i] * y[i]);
         }
        pthread_mutex_lock(&mutexsum);
        dotstr.sum += mysum;
        pthread_mutex_unlock(&mutexsum);

        pthread_exit((void*)0);
}
```

Пример (продолжение)

```
int main(int argc, char *argv[]){
        long i;
        double *a, *b;
        void *status;
        pthread_attr_t attr;
        a = (double*) malloc(NUMTHRDS*VECLEN * sizeof(double));
        b = (double*) malloc(NUMTHRDS*VECLEN * sizeof(double));
        for (i = 0; i < VECLEN*NUMTHRDS; i++){
                a[i] = 1.0;
                b[i] = a[i];    }
        dotstr.veclen = VECLEN;
        dotstr.a = a;
        dotstr.b = b;
        dotstr.sum = 0;
```

Пример (окончание)

```
    pthread_mutex_init(&mutexsum, NULL);

    for (i = 0; i < NUMTHRDS; i++){
        pthread_create(&callThd[i], NULL, dotprod, (void *)i);          }

    for (i = 0; i < NUMTHRDS; i++){
        pthread_join(callThd[i], &status);          }

    printf("Sum =  %f \n", dotstr.sum);
    free(a); free(b);
    pthread_mutex_destroy(&mutexsum);
    pthread_exit(NULL);
}
```

Компиляция и выполнение

- $ gcc -pthread pthr.c
- $ ./a.out

  Sum = 400.000000

- $ gcc -pthread pthr.c -o myname.out
- $ ./myname.out

  Sum = 400.000000

Семафоры

- #include <semaphore.h>
- int **sem_init**(
  sem_t * pSem,
  int bShared,
  unsigned uValue);
- int **sem_destroy**(sem_t * pSem);

- int **sem_wait**(sem_t *pSem);
- int **sem_trywait**(sem_t *pSem);
- int **sem_post**(sem_t *pSem);

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Пример

```
#include <pthread.h>
#include <semaphore.h>
#include <iostream>
#define MY_NUM_THREADS 4

sem_t g_Semaphore;
pthread_mutex_t g_Mutex =
PTHREAD_MUTEX_INITIALIZER;
long g_vlNum = 0;
```

```
void *MyThreadProc(void *)
{
    sem_wait(&g_Semaphore);
    pthread_mutex_lock(&g_Mutex);
    ++g_vlNum;
    pthread_mutex_unlock(&g_Mutex);
    std::cout << g_vlNum << " " << std::flush;
    sem_post(&g_Semaphore);
    pthread_mutex_lock(&g_Mutex);
    --g_vlNum;
    pthread_mutex_unlock(&g_Mutex);
    return 0;
}
```

Пример (окончание)

```
int main(){
    int i;
    pthread_t ahThreads[MY_NUM_THREADS];
    sem_init(&g_Semaphore, 0, MY_NUM_THREADS);
    for (i = 0; i < MY_NUM_THREADS; ++i)
        pthread_create(&ahThreads[i], NULL, MyThreadProc, NULL);
    for (i = 0; i < MY_NUM_THREADS; i++)
        pthread_join(ahThreads[i], NULL);
    sem_destroy(&g_Semaphore);
    pthread_mutex_destroy(&g_Mutex);
}
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Компиляция и выполнение

- evgenia@evgenia:~$ g++ -pthread   pthr.cpp
- evgenia@evgenia:~$ ./a.out
- 3 1 2 4 evgenia@evgenia:~$ ./a.out
- 1 1 1 1 evgenia@evgenia:~$ ./a.out
- 3 2 3 1 evgenia@evgenia:~$ ./a.out
- 2 3 1 1 evgenia@evgenia:~$ ./a.out
- 1 1 1 1 evgenia@evgenia:~$ ./a.out
- 1 1 1 1 evgenia@evgenia:~$ ./a.out
- 34 2 1 evgenia@evgenia:~$

Условная переменная

- int **pthread_cond_init** (condition, attr)
- pthread_cond_t cond = PTHREAD_COND_INITIALIZER
- int **pthread_cond_destroy** (condition)

- int **pthread_condattr_init** (attr)
- int **pthread_condattr_destroy** (attr)

- int **pthread_cond_wait** (condition, mutex)
- int **pthread_cond_signal** (condition)
- int **pthread_cond_broadcast** (condition)

Пример

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS  3
#define TCOUNT 10
#define COUNT_LIMIT 12
int     count = 0;
int     thread_ids[3] = { 0,1,2 };
pthread_mutex_t count_mutex;
pthread_cond_t count_threshold_cv;
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Пример (продолжение)

```
void *inc_count(void *t){
    int i;
    long my_id = (long)t;
    for (i = 0; i < TCOUNT; i++) {
        pthread_mutex_lock(&count_mutex);
        count++;
        if (count == COUNT_LIMIT) {
            pthread_cond_signal(&count_threshold_cv);
            printf("inc_count(): thread %ld, count = %d  Threshold reached.\n", my_id,
            count);}
        printf("inc_count(): thread %ld, count = %d, unlocking mutex\n", my_id, count);
        pthread_mutex_unlock(&count_mutex);
        sleep(1);}
    pthread_exit(NULL);
}
```

Пример (продолжение)

```
void *watch_count(void *t){
    long my_id = (long)t;
    printf("Starting watch_count(): thread %ld\n", my_id);
    pthread_mutex_lock(&count_mutex);
    while (count < COUNT_LIMIT) {
        pthread_cond_wait(&count_threshold_cv, &count_mutex);
        printf("watch_count(): thread %ld Condition signal
        received.\n", my_id);
    }
    count += 125;
    printf("watch_count(): thread %ld count now = %d.\n", my_id,
    count);
    pthread_mutex_unlock(&count_mutex); pthread_exit(NULL);
}
```

Пример (окончание)

```
int main(int argc, char *argv[]){
    int i;
    long t1 = 1, t2 = 2, t3 = 3;
    pthread_t threads[3];
    pthread_mutex_init(&count_mutex, NULL);
    pthread_cond_init(&count_threshold_cv, NULL);
    pthread_create(&threads[0], NULL, watch_count, (void *)t1);
    pthread_create(&threads[1], NULL, inc_count, (void *)t2);
    pthread_create(&threads[2], NULL, inc_count, (void *)t3);
    for (i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
     }
    printf("Main(): Waited on %d  threads. Done.\n", NUM_THREADS);
    pthread_mutex_destroy(&count_mutex);
    pthread_cond_destroy(&count_threshold_cv);
    pthread_exit(NULL);
}
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Результат работы

```
$ gcc -pthread pthr.c
$ ./a.out
Starting watch_count() : thread 1
inc_count() : thread 2, count = 1, unlocking mutex
inc_count() : thread 3, count = 2, unlocking mutex
inc_count() : thread 2, count = 3, unlocking mutex
inc_count() : thread 3, count = 4, unlocking mutex
inc_count() : thread 2, count = 5, unlocking mutex
inc_count() : thread 3, count = 6, unlocking mutex
inc_count() : thread 2, count = 7, unlocking mutex
inc_count() : thread 3, count = 8, unlocking mutex
inc_count() : thread 2, count = 9, unlocking mutex
inc_count() : thread 3, count = 10, unlocking mutex
inc_count() : thread 2, count = 11, unlocking mutex
inc_count() : thread 3, count = 12  Threshold reached.
inc_count() : thread 3, count = 12, unlocking mutex
watch_count() : thread 1 Condition signal received.
watch_count() : thread 1 count now = 137.
inc_count() : thread 2, count = 138, unlocking mutex
inc_count() : thread 3, count = 139, unlocking mutex
inc_count() : thread 2, count = 140, unlocking mutex
inc_count() : thread 3, count = 141, unlocking mutex
inc_count() : thread 2, count = 142, unlocking mutex
inc_count() : thread 3, count = 143, unlocking mutex
inc_count() : thread 2, count = 144, unlocking mutex
inc_count() : thread 3, count = 145, unlocking mutex
Main() : Waited on 3  threads.Done.
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Домашнее задание

- Подготовка к тестированию по материалам лекции
- Читать книгу Таненбаум Э., Бос Х. Современные операционные системы, стр. 146-178.
- Подготовка к лабораторной работе №13