Теория и практика многопоточного программирования

ЛЕКЦИЯ 6. МОДЕЛЬ ИСПОЛНЕНИЯ, КРИТИЧЕСКИЕ СЕКЦИИ И ОБЪЕКТЫ БЛОКИРОВКИ

В прошлый раз...

Общие проблемы многопоточности

Проблемы работы с разделяемой памятью

Проблемы примитивов синхронизации

Темы лекции

Математическая модель многопоточной программы

Терминология теории параллельного программирования

Реализация объектов блокировки

В основе модели исполнения: конечный автомат

$A = \langle T, Q, q_o, F, D \rangle$

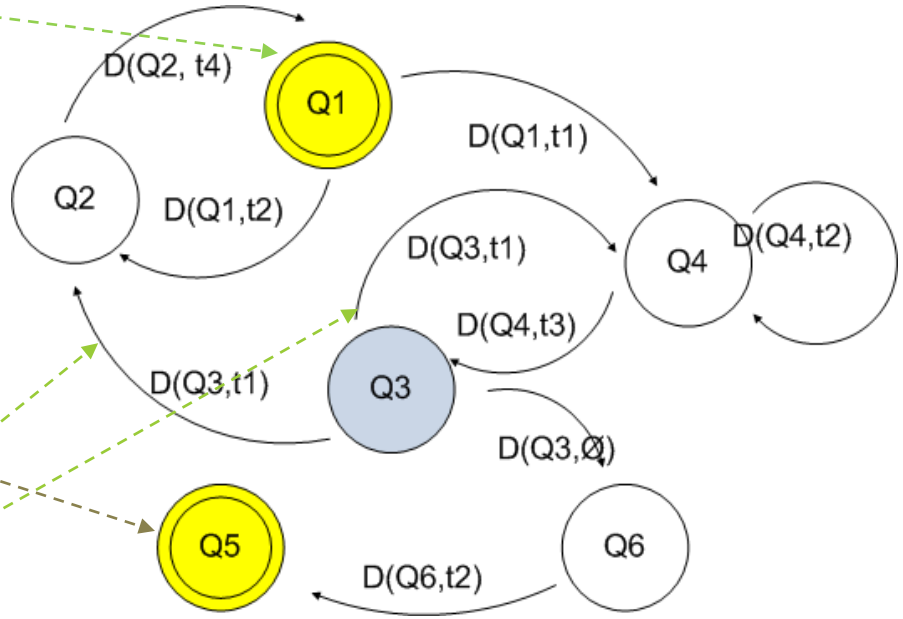
$T^* = T + \{\emptyset\}$

F – подмножество конечных состояний {Q5}

Функция переходов D:

det | $D: Q \times T^* \rightarrow Q$

non | $D: Q \times T^* \rightarrow 2^Q$



Ключевые понятия. Модель

Конечный автомат является моделью **потока** *(thread)* **или всего приложения**. Потоки делят между собой общее время.

Q: **Состояние** (state) – состояние памяти/регистров в ходе исполнения потока. q0 – «точка входа» программы.

D: **Событие** (event) – «появление» команды t_i или выполнение некоторого условия (Ø), при котором происходит **изменение состояния** (*state transition*). Событие считается **моментальным** (*instantaneous*), при этом два события не могут происходить **одновременно** (*simultaneous*). Таковым может быть изменение значения регистра при выполнении атомарной операции инкремента.

Для обозначения отношения **предшествования событий** (*precedence*, «старшинство») используется →.

Например: ***D****(Q3,Ø)* → ***D****(Q6,t2)*

Пример рассуждений

```
1   #include <omp.h>
2   #include <stdio.h>
3   #define N 3
4   int a, b = 0;           //state: состояние переменных
5
6   void foo() {
7       a = N * (N - 1) / 2;    //event: a изменила значение
8       while (a != b);     //empty event: ожидание наступления условия
9       printf("Все потоки стартовали");
10  }
11
12  void bar() {
13  #pragma omp critical
14      b += omp_get_thread_num();  //event: write b
15  }
16
17  int main() {
18  #pragma omp parallel num_threads(N)
19      {
20  #pragma omp master
21          foo();
22          bar();
23      }
24  }
```

Свойства алгоритмов

Неблокирующая синхронизация – отсутствие неопределённых ожиданий:

- **Без препятствий** (obstruction-freedom) – продвигаемся, если не препятствуют другие потоки
- **Неблокируемость** (lock-freedom) – гарантия системного прогресса (если CAS в цикле каждый раз неудачен, значит кто-то преуспел)
  – *допускает циклы while*
- **Свобода от ожиданий** (wait-freedom) – гарантия завершения метода/функции за конечное число шагов
  – *не допускает циклы while*

Ключевы понятия. Блокировка

Обеспечение механизма критических секций через **объект блокировки** (lock):

```
public interface Lock {

        public void lock(); // «взять» (aquire)
        public void unlock(); // «освободить» (release)
}
```

Алгоритмы блокировок. LockOne

Лемма 1. Алгоритм LockOne удовлетворяет условию взаимного исключения.

Доказательство (от противного).

Предположим, $\exists(q,p)[CS_A^q \nrightarrow CS_B^p \ \wedge CS_B^p \nrightarrow CS_A^q]$.

Из кода следует (перед совместным входом):

$$\begin{cases} write_A(flag[A] = true) \rightarrow read_A(flag[B] == false) \rightarrow CS_A \\ write_B(flag[B] = true) \rightarrow read_B(flag[A] == false) \rightarrow CS_B \\ \boldsymbol{read_A(flag[B] == false) \rightarrow write_B(flag[B] = true)} \end{cases}$$

$$write_A(flag[A] = true) \rightarrow read_A(flag[B] == false) \rightarrow$$
$$\rightarrow write_B(flag[B] = true) \rightarrow read_B(flag[A] == false)$$

$$write_A(flag[A] = \boldsymbol{true}) \rightarrow read_B(flag[A] == \boldsymbol{false})$$

Получили противоречие (missing $write_A(flag[A] = false)$)

Алгоритмы блокировок. LockTwo

Из кода следует (перед совместным входом):

$write_A(victim = A) \rightarrow read_A(victim == B) \rightarrow CS_A$

$write_B(victim = B) \rightarrow read_B(victim == A) \rightarrow CS_B$

$write_B(victim = B) \rightarrow read_A(victim == B)$

$write_A(victim = A) \rightarrow write_B(victim = B) \rightarrow read_A(victim == B)$

$\nrightarrow read_B(victim == A)$

Алгоритмы блокировок. PetersonLock.

Взаимное исключение – аналогично LockOne (самостоятельно)

Свобода от зависания.

Без ограничения общности «завис» поток A.

$read_A(victim == A \wedge flag[B] == true)$

Если поток B вызывает *lock(),* то $\boldsymbol{write_B(victim = B)}$

Если поток B вызывает *unlock(),* то $\boldsymbol{write_B(flag[B] = false)}$

Если поток B висит – противоречие по значению ***victim***.

«Честность».

Метод блокировки можно разделить на 2 части:

- «*вход*» (doorway) – часть D_A метода *lock()* с фиксированным количеством интрукций
- «*ожидание*» (waiting) – часть W_A метода *lock()* неограниченной продолжительности

Система потоков обладает свойством «*честности*» (**fairness**) по отношению к последовательности «входов» D_A и D_B процесса блокировки любых двух потоков *A* и *B* с критическими секциями CS_A и CS_B в том случае, когда выполняется следующее утверждение: если $D_A \rightarrow D_B$ то $CS_A \rightarrow CS_B$ для любых A и B из системы.