Архитектура операционных систем

Лекция 1.6

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
}
```

Нарушается условие ограниченного ожидания

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
}
```

Нарушается условие ограниченного ожидания

Недостатки программных алгоритмов

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

L

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

H

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

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

Проблема Producer-Consumer

Producer:

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

Consumer:

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

Информация передается через буфер конечного размера – N

Проблема Producer-Consumer

Решение с помощью семафоров

```
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();
}
```

Мониторы Хора (Hoare)

Структура

Monitor monitor_name {

Описание переменных;

void m_1(…) { … }

void m_2(…) { … }

…

void m_n(…) { … }

Блок инициализации переменных;

}

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

Проблема Producer-Consumer

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();
}
```

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

Проблема Producer-Consumer

Решение с помощью сообщений

Producer:

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

Consumer:

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