Вопросы к проверке знаний

1. Уровни абстракции технологий

Расположите в порядке увеличения степени абстракции технологии высокопроизводительных вычислений:

1. OS Threads, SSE/MMX, C++11 Threads/Java Threads, TBB/Fork-Join framework
2. SSE/MMX, OS Threads, TBB/Fork-Join framework, C++11 Threads/Java Threads
3. SSE/MMX, OS Threads, C++11 Threads/Java Threads, TBB/Fork-Join framework
4. OS Threads, TBB/Fork-Join framework, C++11 Threads/Java Threads, SSE/MMX

2. Закон Амдала

Что показывает закон Амдала:

1. Прирост производительности в зависимости от доли последовательного кода и числа вычислительных элементов
2. Прирост производительности в зависимости от числа вычислительных элементов и используемой технологии распараллеливания
3. Число вычислительных элементов, необходимое для достижения заданного уровня роста производительности
4. Долю последовательного кода, производительность которого невозможно повысить

3. Завершение потоков

Как называются функции, внутри которых, в зависимости от используемой технологии, происходит проверка на необходимость завершения работы потока и по результатам проверки возможно выкидывание исключений или завершение потока с применением стека зарегистрированных функций:

1. Функции завершения потока
2. join-функции
3. Cancellation / interruption points
4. Signal handlers

4. Spin mutex

Чем отличается spin mutex от обычного mutex:

1. Остаётся всегда в user space
2. Остаётся всегда в kernel space
3. Ничем не отличается
4. Позволяет читателям производить операции параллельно

5. Wait/notify

В каком состоянии находится примитив синхронизации (mutex) в приведённом коде на 1, 3 и 4 строках:

```
0: boost::unique_lock<boost::mutex> lock(mutex);
1:
2: while (messageQueue.empty())
3:   conditionVariable.wait(lock);
4:
```

1. Во всех строках захвачен
2. Во всех строках освобождён
3. Захвачен в 1 и 4 строках, в 3 на время вызова *wait* освобождается
4. Захвачен в 1 строке, в 3 и 4 освобождён

6. Ранжирование алгоритмов

Расположите алгоритмы синхронизации в порядке увеличения потенциальной производительности в жёсткой конкурентной среде:

1. Грубая, тонкая, оптимистичная, неблокирующая
2. Тонкая, грубая, оптимистичная, неблокирующая
3. Неблокирующая, оптимистичная, грубая, тонкая
4. Грубая, тонкая, неблокирующая, оптимистичная

7. Выбор алгоритма

Отметьте вид алгоритмов синхронизации, обычно реализуемый на CAS-операциях и позволяющий добиться максимальной независимости исполнения потоков, гарантируя общий прогресс системы:

1. Грубая
2. Тонкая
3. Оптимистичная
4. Неблокирующая

8. Wait-free алгоритм

Выберите корректное определение wait-free класса алгоритмов:

1. Любой поток ожидает, пока хотя бы один другой поток не выполнил свою операцию
2. Любой поток может выполнить свои функции за конечное число шагов, если ни один другой поток не находится в критической секции

3. Система в целом продвигается вперед не зависимо ни от чего
4. Любой поток может выполнить свои функции за конечное число шагов, не зависимо ни от чего

9. Определение типа ошибки

Какая ошибка потенциально может возникнуть в приведённом коде:

```
void f() {
  first_mutex.lock();
  second_mutex.lock();
  ...
}

void g() {
  second_mutex.lock();
  first_mutex.lock();
  ...
}
```

1. Гонка данных (Data Race)
2. Взаимная блокировка (Deadlock)
3. Проблема ABA
4. Инверсия приоритетов

10. Реализация lock-free алгоритмов

Почему lock-free алгоритмы легче реализовать на языке со сборщиком мусора, нежели на нативных языках, например C++:

1. Невозможна инверсия приоритетов
2. Наличие существующих базовых контейнеров в java.util.concurrent
3. Невозможна проблема ABA
4. Наличие CAS-операций

Консенсус

Какое консенснусное число CAS-операций:

1. 1
2. 2
3. 2N-2 (где N-число ячеек памяти, доступных для CAS)
4. Бесконечность

From:
http://wiki.osll.ru/ - **Open Source & Linux Lab**

Permanent link:
http://wiki.osll.ru/doku.php/courses:high_performance_computing:questions?rev=1479592514

Last update: **2016/11/20 00:55**