***14. Резервация мониторов***

Ничего нормального не нашел

[lock reservation](http://www.eecg.toronto.edu/~steffan/workshops/07/cdp/grcevski2.html)

Lock reservation, a powerful optimization for Java locks, is based on the observation that, in Java, each lock tends to be dominantly acquired and released by a specific thread. Reserving a lock for such a dominant thread allows the owner thread of the lock to acquire and release the lock without any atomic read-modify-write instructions.

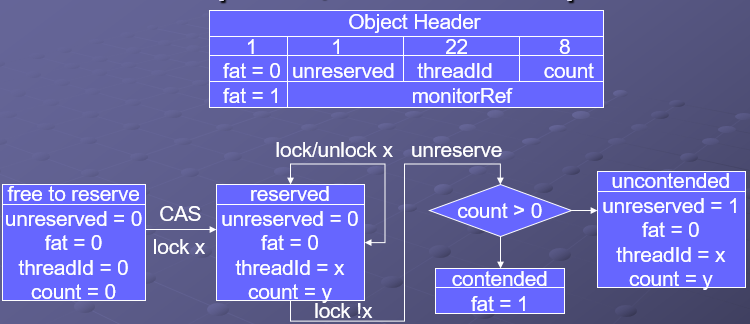
This section presents a new locking algorithm called lock reservation. It exploits the observation that Java locks show thread locality, as discussed in the previous section. The key idea is to reserve locks for threads. When a thread attempts to acquire an object’s lock, one of the following actions is taken:

1. If the object’s lock is reserved for the thread, the runtime system allows the thread to acquire the lock with a few instructions involving no atomic operation.

2. If the object’s lock is reserved for another thread, the runtime system cancels the reservation, and falls back to a conventional algorithm for further processing.

3. If the object’s lock is not reserved, the runtime system uses a conventional algorithm.

Our algorithm can be built on any existing locking algorithm, as long as it uses a lockword, a word in the object header for locking, and allows one bit to be available in the lockword. The bit is used for representing the lock reservation status, and hence named the LRV bit. When the LRV bit is set, the lockword is in the reserve mode, and the structure is defined by our algorithm. When the bit is not set, the lockword is in the base mode, and the structure is de-fined by the underlying algorithm that the runtime system falls back to after canceling the reservation.



* Операция снятия резервации требует остановки нити владельца, является дорогостоящей
* Операция снятия резервации может быть выполнена, если нить владелец не запланирована для исполнения, например, ожидает в wait

*Типы планирования*

* Вытесняющее планирование
  + Возможность остановки в любой инструкции
  + Согласовано с планировщиком многозадачных OS
  + Сложно согласовать с GC
* Кооперативное планирование
  + Остановка в безопасных точках (Yield Point, Safe Point, GC Point)
  + Предпочтительный способ при реализации в ядре OS
  + Требует ожидания перепланировки при остановке нити планировщиком OS, может приводить к задержке
  + Хорошо согласуется с GC

*Безопасные точки*

* Вызов метода
* Создание объекта
* Обратное ветвление
* Синхронизация
* Инструкции генерирующие исключение (Potential Exception Instruction)

*Взаимодействие с GC*

* Расположение ссылок на объекты на стеке и в локальных переменных
  + Требуется при перечислении корневых ссылок
  + Компилятор может использовать регистры
* Карты ссылок (GC Maps) генерируются компилятором
  + При вытесняющем планировании для каждой инструкции процессора
  + При кооперативном планировании для каждой безопасной точки
* Правило Гослинга (Gosling property) – для любой точки корректного Java метода карта ссылок не зависит от пути в графе управления, приведшего в эту точку

*Системные нити*

* 1:1 отображение управляемых и OS нитей
* Простой планировщик через вызовы OS
* Любой способ планирования
  + При кооперативном планировании высокая вероятность конфликта с планировщиком OS
* Требуется синхронизация данных между всеми нитями
* Структура стека определяется OS
  + Ограничение количества нитей

*Зеленые (green, logical) нити*

* Простой вид M:N отображения нитей
  + М – управляемые нити, N – нити OS
  + N ≤ P – количество процессоров
* Кооперативное планирование нитей
  + Планирование по времени может не использоваться. Упрощение системы, но опасность голодания.
  + При реализации планирования по времени усложнение планировщика
  + Уменьшается вероятность небезопасной остановки нитей планировщиком OS.
* Не требуется синхронизация данных между управляемыми нитями ассоциированными с одним процессором
* Невозможность использования процессора при системном вызове
* Любая структура стека
  + Количество нитей не ограничено OS

*Комбинированные M:N нити*

* Сложное отображение нитей
  + М – управляемые нити, N – нити операционной системы
  + N ≥ P – количество процессоров
* При системном вызове создается запасная нить ассоциированная с тем же процессором
* Усложнение синхронизации между нитями OS на одном процессе