3.14

3.14. Lock-free и wait-free программы. Задача о консенсусе, способы решения. Универсальные операции. Конструкция Херлики для реализации lock-free и wait-free структур данных. Базовый формализм Лампорта.

Lock-free и wait-free программы
1. synchronization mechanisms that do not use locks and are called ***lock-free*** [стр. 88]
2. "Lock-free" refers to the fact that a thread cannot lock up: every step it takes brings progress to the system [wikipedia]

1. If lock-free synchronization also guarantees that each operation finishes in a bounded number of steps, then it is called ***wait-free***.
2. "Wait-free" refers to the fact that a thread can complete any operation in a finite number of steps, regardless of the actions of other threads. All wait-free algorithms are lock-free, but the reverse is not necessarily true. [wikipedia]
3. Wait-free алгоритм впускает поток в критическую секцию за время, не зависящее от того, что делают другие процессы.

Задача о консенсусе. [стр. 101]
Проблема консенсуса: система из N потоков, потоки общаются через разделенные объекты (каждый объект имеет свое значение – propose), в итоге потоки должны договориться об общем знании – decide.
Пример проблемы: в связный список разные потоки пытаются добавить элемент – все процессы должны договориться между собой какой элемент вставлять первым.
Консенсусный объект поддерживает следующий интерфейс:
public interface Consensus {
public void propose **(i n t** pid **, int** value) ;
public int decide **(i n t** pid) ;
}

Требования к объектам:

- *Agreement:* 2 корректных процесса не могут выбрать (decide) разные значения.
- *Validity:* Значение выбранное (decided) корректным процессом должно быть одним из предложенных.
- *Wait-free*: Каждый корректный процесс делает выбор за конечное количество шагов.

[стр. 102]
Консенсусное число объекта O – максимальное число процессов, которые смогут решить проблему консенсуса при помощи объекта O.
Объект *универсальный*, если его консенсусное число равно бесконечности.
Бивалентное состояние (bivalent state) – есть два возможных решения, начиная с этого глобального состояния.
Одновалентное – возможно лишь одно решение.
Критическое состояние – все возможные шаги ведут в одновалентные состояния.
Лемма. Любой двух-потоковый консенсусный протокол имеет начальное бивалентное состояние. (если не так – не было бы консенсуса)
Лемма У каждого консенсусного протокола есть критическое состояние
(если бы не было – был бы не wait-free)
Теорема. Read/Write atomic регистры имеют консенсусное число 1.
Док-во – для двух потоков задачу консенсуса не решить.
Способы решения консенсусной проблемы:

3.14

Обычно такие объекты предоставляет hardware. Универсальный объект - **CompSwap** (Compare and Swap):

public class CompSwap {
i n t initValue = 0 ;
public ConipSwap(i n t i n i t v a l u e) {
myvalue = i n i t v a l u e ;
}
public synchronized i n t compSwapOp(i n t prevValue , i n t newvalue) {
i n t oldValue = myvalue;
i f (myvalue == prevvalue)
myvalue = newvalue ;
r e t u r n oldvalue ;
}
}
	public class CompSwapConsensus implements Consensus {
	CompSwap x = **new** CompSwap (- 1) ;
	int proposed []
	public ConipSwapConsensus(i n t n) {
	proposed = new i n t [n] ;
	}
	}
	public void propose(int pid , int v) {
	proposed [pid] = v ;
	public i n t decide (int pid) {
	i n t j = x.compSwapOp(-1, pid) ;
	i f (j == -1)
	else
	return proposed [pid] ;
	return proposed [j] ;
	}}

Конструкция Херлихи для реализации lock-free и wait-free структур данных.[стр 107]
Еще один объект - load-linked and storeconditional (LLSC) register.
Load-Linked – позволяет потоку загрузить значение указателя на объект.
Store-Conditional – позволяет обновить указатель на объект если он не менялся с последнего Load-Linked.
Используем LLSC для реализации параллельной (concurrent) очереди при помощи техники передвижения указателя (*pointer-swinging):*

1. читает указатель, используя Load-Linked
2. делает копию объекта
3. выполняет операцию над копией объекта
4. передвигает указатель объекта на копию, если оригинальный объект не поменялся
5. если оригинальный объект успел измениться, то значит кто-то уже добавил элемент, -> шаг 1.

Wait-free реализация:

1) Все потоки анонсируют свои операции.
2) Читает при помощи load-linked
3) Создает копию
4) Над копией выполняет *все* заанонсированные действия
5) Передвигает указатель (store conditional). Если указатель не менялся, передвигает. Если поменялся, кто-то другой уже выполнил все анонсированные операции.

В книжке есть примеры Java кода этих объектов.
Для больших объектов эта технология не оч подходит – копировать не удобно.

Базовый формализм Лампорта.
Выполнение системы – тройка {S, –>, -->}. S – не более чем счетный набор событий и действий, стрелки – два отношения предшествования, удовлетворяющие аксиомам A1-A5 ниже.
A –> B – A завершится раньше, чем начнется B.
A --> B – выполнение A начнется ранее или в тот же момент, когда B завершится.
Аксиомы (базовый формализм Лампорта)
A1. Отношение –> иррефлексивно, отношение частного порядка

3.14

A2. Если A–>B, то A-->B и B“не –>”A
A3. Если A–>B-->C или A-->B–>C, то A-->C
A4. Если A–>B-->C–>D, то A–>D
A5. для любого A из S, множество всех B: A”не–>”B – конечно.

Аксиома глобального времени: либо A-->B, либо A–>B.

System model [стр. 77]
e– событие, inv(e) – начало события, resp(e) -конец события
История – это (H, $<_H$): H-множество операций, $<_H$ – иррефлексивное, транзитивное отношение.
e$<_H$ f – resp(e) произошло до inv(f).

Process order: *(proc(e) = proc(f) (в одном процессе)) и (resp (e)* occurred before *inv (f))* .
Object order: *(object(e)* n *object(f)* не равно 0) и *(resp(e)* occurred before *inv(f)).*

Опр. История *последовательная*, если $<_H$ полный порядок
Опр. История *легальная*, если удовлетворяет послед. спецификации (sequential specification) всех объектов (пример: For example, if we are considering a read-write register ***x*** as a shared object, then a sequential history is legal if for every read operation that, returns its value as **v**, there exists a write on that object with value **v**, and there does not exist another write operation on that object with a different value between the write and the read operations.).

Последовательная согласованность
История послед. согласованная, если существует последовательная история S эквивалентная H, такая что S – легальная и удовлетворяет process order
Линеаризуемость
История линеаризуема, если существует последовательная история S эквивалентная H, такая что S легальная и сохраняет $<_H$.

Основной источник – книга Concurrent and Distributed Computing in Java, Vijay K. Garg.