interface State {

int size();

long[] current();

void swap(int i, int j);

}

// This is a dummy implementation, useful for

// deducing the overhead of the testing framework.

class NullState implements State {

private long[] value;

NullState(int length) { value = new long[length]; }

public int size() { return value.length; }

public long[] current() { return value; }

public void swap(int i, int j) { }

}

class SynchronizedState implements State {

private long[] value;

SynchronizedState(int length) { value = new long[length]; }

public int size() { return value.length; }

public long[] current() { return value; }

public synchronized void swap(int i, int j) {

value[i]--;

value[j]++;

}

}

class UnsynchronizedState implements State {

private long[] value;

UnsynchronizedState(int length) { value = new long[length]; }

public int size() { return value.length; }

public long[] current() { return value; }

public void swap(int i, int j) {

value[i]--;

value[j]++;

}

}

import java.util.concurrent.atomic.AtomicLongArray;

class AcmeSafeState implements State {

private AtomicLongArray value;

AcmeSafeState(int length) { value = new AtomicLongArray(length); }

public int size() { return value.length(); }

public long[] current() {

long[] temp = new long[value.length()] ;

for(int i=0;i<value.length();i++)

temp[i]=value.get(i);

return temp;

}

public void swap(int i, int j) {

value.getAndDecrement(i);

value.getAndIncrement​(j);

}

}

import java.util.concurrent.ThreadLocalRandom;

import java.lang.management.ThreadMXBean;

class SwapTest implements Runnable {

private long nTransitions;

private State state;

private ThreadMXBean bean;

private long cputime;

SwapTest(long n, State s, ThreadMXBean b) {

nTransitions = n;

state = s;

bean = b;

}

public void run() {

var n = state.size();

if (n <= 1)

return;

var rng = ThreadLocalRandom.current();

var id = Thread.currentThread().getId();

var start = bean.getThreadCpuTime(id);

for (var i = nTransitions; 0 < i; i--)

state.swap(rng.nextInt(0, n), rng.nextInt(0, n));

var end = bean.getThreadCpuTime(id);

cputime = end - start;

}

public long cpuTime() {

return cputime;

}

}

import java.lang.management.ManagementFactory;

class UnsafeMemory {

public static void main(String args[]) {

if (args.length != 4)

usage(null);

try {

var nThreads = (int) argInt (args[1], 1, Integer.MAX\_VALUE);

var nTransitions = argInt (args[2], 0, Long.MAX\_VALUE);

var nValues = (int) argInt (args[3], 0, Integer.MAX\_VALUE);

State s;

if (args[0].equals("Null"))

s = new NullState(nValues);

else if (args[0].equals("Synchronized"))

s = new SynchronizedState(nValues);

else if (args[0].equals("Unsynchronized"))

s = new UnsynchronizedState(nValues);

else if (args[0].equals("AcmeSafe"))

s = new AcmeSafeState(nValues);

else

throw new Exception(args[0]);

dowork(nThreads, nTransitions, s);

test(s.current());

System.exit (0);

} catch (Exception e) {

usage(e);

}

}

private static void usage(Exception e) {

if (e != null)

System.err.println(e);

System.err.println("Usage: model nthreads ntransitions nvalues\n");

System.exit (1);

}

private static long argInt(String s, long min, long max) {

var n = Long.parseLong(s);

if (min <= n && n <= max)

return n;

throw new NumberFormatException(s);

}

private static void dowork(int nThreads, long nTransitions, State s)

throws InterruptedException {

var test = new SwapTest[nThreads];

var t = new Thread[nThreads];

var bean = ManagementFactory.getThreadMXBean();

bean.setThreadCpuTimeEnabled(true);

for (var i = 0; i < nThreads; i++) {

var threadTransitions =

(nTransitions / nThreads

+ (i < nTransitions % nThreads ? 1 : 0));

test[i] = new SwapTest (threadTransitions, s, bean);

t[i] = new Thread (test[i]);

}

var realtimeStart = System.nanoTime();

for (var i = 0; i < nThreads; i++)

t[i].start ();

for (var i = 0; i < nThreads; i++)

t[i].join ();

var realtimeEnd = System.nanoTime();

long realtime = realtimeEnd - realtimeStart, cputime = 0;

for (var i = 0; i < nThreads; i++)

cputime += test[i].cpuTime();

double dTransitions = nTransitions;

System.out.format("Total time %g s real, %g s CPU\n",

realtime / 1e9, cputime / 1e9);

System.out.format("Average swap time %g ns real, %g ns CPU\n",

realtime / dTransitions \* nThreads,

cputime / dTransitions);

}

private static void test(long[] output) {

long osum = 0;

for (var i = 0; i < output.length; i++)

osum += output[i];

if (osum != 0)

error("output sum mismatch", osum, 0);

}

private static void error(String s, long i, long j) {

System.err.format("%s (%d != %d)\n", s, i, j);

System.exit(1);

}

}