

**Thread-based Parallelism and Performance**  
**Due April 7, 2005 in Lecture**

---

**CSE160 Homework Policy:** Cheating WILL be taken seriously. It is not fair to honest students to take cheating lightly, nor is it fair to the cheater to let him/her go on thinking that is a reasonable alternative in life.

- The following is not considered cheating:
  - discussing homework or lab problems in groups.
- The following is:
  - producing the solution to the problem in a group
  - discussing homework or lab with someone who has already completed the problem, or looking at their completed write-up
  - copying another student's solution, or
  - receiving, providing, or soliciting assistance from another student during a test.

**Getting a CSE160 Account:** You should do this right away, as it may take a little while to get the accounts set up. In order to get accounts on the *fwgrid* machines visit the following link [http://fwgrid.ucsd.edu/register\\_project.php](http://fwgrid.ucsd.edu/register_project.php). Then fill up all the necessary information.

- If you are a CSE student and have an existing user name and password use the associated CSE user id in the *user id* field.
- For the field corresponding to *Faculty/Supervisor name* fill in Professor Andrew Chien.
- For the field corresponding to *Project Title* fill in "Student Account for CSE 160".
- For the *Project Website* field fill in <http://csag.ucsd.edu/teaching/cse160s05/>
- For *Project Abstract* just fill in the following text – "Student Account for CSE 160".

For any other doubts contact [snandy@cs.ucsd.edu](mailto:snandy@cs.ucsd.edu)

---

**Thread-based Parallelism and Performance**

Homework Problem 1. Consider the following basic questions about threading and synchronization in Java. Answer each as succinctly as possible. Provide a code segment where requested. It is not a requirement that you run your code for the purpose of the homework, but that is often a good strategy to make sure that you understand things correctly.

1.a. Threads and Parallelism. Are threads the only way to express parallelism/concurrency in Java? (Explain why yes or no)

Write a small section of code that creates a large amount of parallelism (say 100-fold) as quickly as possible.

Explain how long it takes to go from sequential to 100-fold concurrency (in general terms, not exact timings), explain why your method is the fastest.

1.b. Threads and Processor Parallelism. For over twenty years, multithreaded programs have been able to exploit multiple processors in a machine to obtain higher performance (processor parallelism). However, when two threads execute simultaneously (running on two processors), it can be quite different from when they are executed in an interleaved fashion. Write a short program whose purpose it is to detect whether the Java program is being run on a machine with true parallelism. Explain how the program decides whether threads are being run in parallel and announces the result. Partial credit for programs that never announce the wrong answer, but in some cases can't decide.

1.c. Synchronized Methods. Java uses synchronized methods to control concurrent access to object state. Write a program which defines two classes **RightSideUp** and **UpSideDown** to maintain the state of a sailboat. Define operations which update the state of these two objects, appropriately synchronized, to ensure that no one accessing these objects could ever see an inconsistent view (that is both **RightSideUp** and **UpSideDown**) in the same state. Beware of deadlock!

Homework Problem 2. There is a single threaded and a double threaded version of the MergeSort algorithm given in the `/home/cse160/hw1`. The double-threaded version has a bug which prevents it from running properly (you can run it a few times yourself to check it). You are expected to detect the bug and suggest a way to solve it. The student is **not expected to hand in the corrected code** but the solution description should be detailed enough to suggest a working solution.

Note: To run the programs use `java ProgramName DataSize`, where `DataSize` has to be some perfect power of 2.

```
usage: java SingleThreadedMergeSort 8192
usage: java DoubleThreadedMergeSort 16384
```

-\*-

Homework Problem 3.a. Write a program that launches a given number of threads (T). The program also takes as input an integer value (Max). All the threads then try to increment a common integer object (make sure to ensure that access to this object is synchronized). Each thread is assigned a unique id (generated sequentially). All increments to the Integer's value are made in steps of 1 and the thread releases the lock on the Integer object after every increment. The thread that makes the increment taking the object's value to Max is declared the winner.

usage: java Assignment-One-One T Max  
sample usage: java Assignment-One-One 10 10000  
sample output: thread number 7 touched 10000

3.b. Modify the above program so that each thread can choose to *sleep* for a given amount of seconds (S) at most. The program also takes in two additional inputs (N, X), where a thread sleeps for X seconds if it has made N increments without being interrupted (provided it hasn't exhausted its total of S seconds of sleeping time). The threads thus control their own *liveness* (albeit by a crude algorithm). The output required is the same as 1.1.

usage: java Assignment-One-Two T Max S N X  
sample usage: java Assignment-One-Two 10 10000 10 10 1  
sample output: thread number 5 touched 10000

3.c. Modify the program from above to implement your own *sleeping policy* (subject to the fact that a thread cannot sleep for more than S seconds). Take in as input an additional value K ( $K \leq T$ ). Now, make all threads other than the thread with id K implement the same liveness control policy as above (1.2). For the thread with id K implement your own policy in order to ensure that it has a better chance of winning!

usage: java Assignment-One-Two T Max S N X K  
sample usage: java Assignment-One-Two 10 10000 10 10 1 5  
sample output: thread number 7 touched 10000 – sadly thread number 5 did not win!  
sample output: thread number 5 touched 10000 – yes, you had a good policy!

-\*-