

CSc 422 — Homework 2

Due Tuesday, February 24, 2009 *in class*

This assignment is again worth 40 points, divided as indicated. You may discuss the meanings of questions with classmates, but the answers and program you turn in must be yours alone. Please explain your answers clearly and succinctly.

Append a commented listing of your program and the table of timing results to your answers to the first five questions. Also submit your program electronically as described at the end of the assignment. Be sure to follow the programming style described in the handout for Homework 1.

I encourage you to get started sooner rather than later! The problems are more difficult than those on the first assignment, and if you have questions, you need to give us time to answer.

1. [5 points] MPD book, Exercise 3.3, parts (a) and (b).
2. [4 points] MPD book, Exercise 3.8.
3. [5 points] MPD book, Exercise 4.3.
4. [5 points] MPD book, Exercise 4.6.
5. [6 points] MPD book, Exercise 4.31.
6. Write a parallel program that uses the bag-of-tasks paradigm to solve the following problem. Use C and Pthreads or use MPD. Develop your program on Lectura and test it on Voltron. (Remember to recompile on Voltron!) Run the timing tests described below, and turn in (on paper) a table of results as well as your program listing.

There is an online dictionary in `/usr/share/dict/words` on both Lectura and Voltron. It contains 479,827 words on Lectura. The Voltron version is somewhat older (because the OS is older) and contains 479,625 words. The dictionary is used by spell checkers and other programs, and some of the entries aren't actual English words.

Your task is to find all words in the dictionary that are *palindromes*—i.e., words in which the sequence of letters is the same reading left to right or right to left. For example, "dad" and "mom" are palindromes, but "son" and "daughter" are not. Single letters are also palindromes. For simplicity, you may assume that case is significant; e.g., "Dad" would not be a palindrome.

I suggest that you first write a sequential program and then modify it to use the bag-of-tasks paradigm. Your sequential program should have the following phases:

- Read the dictionary file `/usr/share/dict/words` into an array of strings. You may assume that each word is at most 25 characters long.
- Examine each word, one at a time. If it is a palindrome, store the word in a second array of strings.
- After you have examined all words, write out the total number of palindromes to `stdout` and write the palindromes themselves to a file `results.txt`.

The first few words in the dictionary start with numbers (take a look), and the next word is "-a". Skip over these and start with "A", the first word that begins with a letter.

After you have a working sequential program, modify it to use the bag-of-tasks paradigm. Your parallel program should use w worker processes, where w is a command-line argument. Use the workers *just* for the compute phase; do the input and output phases sequentially.

Use 26 tasks in your parallel program, one for each letter of the alphabet. In particular, the first task is to examine all words that begin with "a", the second task is to examine all words that begin with "b", and so on. During the input phase you should build an efficient representation for the bag of tasks; I suggest using an array, where the value in `task[1]` is the index of the first "a" word, `task[2]` is the index of the first "b" word, and so on.

Each worker will obviously be working on a unique task, but if all workers are writing to a single list of palindromes, you would have a critical section. When developing your parallel program, you might find it easiest first to use the single palindrome list from your sequential program (using locks to protect updates to it). However, your final program (the one you turn in) should avoid the need for critical sections by using multiple output buffers, one for each task. (If you first use a single buffer, you might find it instructive to time both programs to measure the synchronization overhead.)

Your parallel program should also time the *compute* phase. Do not time the sequential input and output phases. If you use Pthreads, use the `times` function as in the program `clock.c`. If you use MPD, use the `age` function as in the program `find.mpd`. Read the clock just before you create the workers; read it again as soon as they have finished. The return value from `times()` is in hundredth's of seconds; the return value from `age()` is in milliseconds. Write the elapsed time for the compute phase to the standard output.

To summarize, your parallel program should have the following output:

the total number of palindromes,
the elapsed time for the compute phase,
and the palindromes themselves.

Write the first two items to `stdout`; write the words to a file named `results.txt` in the directory that contains your program.

Timing Tests. Execute your parallel program on Voltron using 2, 3, and 4 workers. Run each test 3 times. Include a table of results with your homework answers; it should contain all the values written to standard output (but not the words themselves) for all 9 test runs. If you use MPD, be sure to set the `MPD_PARALLEL` environment variable to 4 just before you run the timing tests.

Electronic Turnin. Use the `turnin` program on Lectura to submit your *parallel* program. The assignment name is `hw2.program`. The file name should be `palindromes.c` or `palindromes.mpd`.