Homework/Lab 2

due Wednesday, February 19, 2003, 11:59 PM

There are several files to turn in for this assignment. They should all be included in a file named `hw02.tar` that you submit using the `turnin` utility. *Please use the filenames specified* and be sure to include your name in each file.

1. Jordan and Alaghband Problem 1.4, p. 20. Write your answer in a plain text file *hw02.txt* and include it in your tar file. (5 points)

2. (15 points) There is an online dictionary `/usr/dict/words` on bullpen that is used by the `spell` command. Recall that a *palindrome* is a word or phrase that reads the same in either direction, *i.e.*, if you reverse all the letters you get the same word or phrase. A word is *palindromic* if its reverse is also in the dictionary. For example, "noon" is palindromic, because it is a palindrome and hence its reverse is trivially in the dictionary. A word like "draw" is palindromic because "ward" is also in the dictionary.

   Your task is write a C or C++ program to find all palindromic words in the dictionary. You should first write a sequential program and then parallelize it, using the bag-of-tasks paradigm. However, keep your plan for parallelization in mind when designing and implementing the sequential version. You might want to do some things in the sequential program that you will need in the parallel program, such as finding where each letter begins in the dictionary (see below for more on this).

   Your sequential program should have the following phases:

   - Read the file `/usr/dict/words` into an array of strings. You can determine the size of this array ahead of time by running `wc` on the dictionary to see how many words it contains. You may assume that each word is at most 25 characters long.
   - For each word, compute its reverse and do a linear search of the dictionary to see if the reverse of the word is in the dictionary. If so, mark the word and increment your counter of the total number of palindromic words.
   - Print the total number of palindromic words to the screen (`stdout`).

   The first few words in the dictionary start with numbers; you can either skip over them or process them, as you wish. None are palindromic, so this choice will not affect your total count. Some words start with capital letters (and hence the dictionary is not sorted in ASCII order). To keep your program simple, leave the capitals alone and do case-sensitive comparisons.

   The sequential program is something you should have no trouble designing, but as many of you are not expert C or C++ programmers (yet), it might take longer than you expect to implement. Please ask if you run into trouble with the sequential version! You will need to get it done with enough time left to work on the parallelization – which is the whole point of the assignment!

After you have a working sequential program, modify it to use the bag-of-tasks paradigm, implemented using the *pthreads* library. Your parallel program should use W worker threads, where W is a command-line argument. Use the workers just for the compute phase; do the input and output phases sequentially. Each worker should count the number of palindromic words that it finds. Sum these W values during the output phase. This avoids a critical section during the compute phase – you'll need to deal with others, though. Use 26 tasks in your program, one for each letter of the alphabet. In particular, the first task is to examine all words that begin with "a" (and numbers), 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[0]` is the index of the first "a" word, `task[1]` is the index of the first "b" word, and so on. You can also use this array during the search phase to limit the scope of your linear searches. Your parallel program should also time the compute phase. You may use the `timer.c` and `timer.h` code from our class examples. Start the clock just before you create the workers; read it again as soon as they have finished. Write the elapsed time for the compute phase to `stdout`.

To summarize, your program should have the following output:

- total number of palindromic words (to *stdout*)
- the number found by each worker (to *stdout*)
- the elapsed time for the compute phase (to *stdout*)

For the timing tests, execute your parallel program on the four-processor nodes of bullpen (*wetteland* or *rivera*; `ppn=4` in PBS) using using 1, 2, 3, and 4 workers. Run each test 3 times. Submit your job through PBS. You may do all tests in a single PBS script, or create a script for each run and submit them separately. In the file `hw02.txt`, include a table of results; it should contain all the values written to standard output (but not the words themselves) for all 12 test runs, and a brief analysis of the speedup and parallel efficiency you have achieved.

Your submitted tar file should include your `Makefile`, your C source code (including the timer code from class, if you choose to use it), your PBS script(s), a brief `README` file expaining how to run your program, and the `hw02.txt` file. Please do *not* include object files or your executable.

Honor code guidelines: While the program is to be done individually, along the lines of a **laboratory program**, I want to encourage you to ask questions and discuss the program with me, our TA, and with each other, as you develop it. However, no sharing of code is permitted. If you have any doubts, please check first and avoid honor code problems later.

Grading guidelines: Your grade for the program will be determined by correctness, design, documentation, and style, as well as the presentation of your timing results.