CSc 422 — Homework 2

Due Tuesday, February 22, 2005 *in class*

This assignment is again worth 40 points. 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.

There are seven problems. The first six are worth 5 points each. You are to do *any five* of the first six problems—or do all six and we will count only your five highest scores. Everyone is to do the programming assignment, which is worth 15 points.

Append a commented listing of your program and the table of timing results to your answers to the first six 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. MPD book, Exercise 3.5, part (a). *Be sure to read the Errata for this exercise (page 143).*

2. MPD book, Exercise 3.6. Again, be sure to see the Errata for page 143.

3. MPD book, Exercise 3.30. Assume that there are several processes that simultaneously try to do inserts and deletes from the shared queue.

4. MPD book, Exercise 4.4.

5. MPD book, Exercise 4.6.

6. MPD book, Exercise 4.30, parts (a) and (c). For part (a), your answer should be similar to the readers/writers solution in Figure 4.10; be sure to give a global invariant. For part (c), a good way to make your solution fair is to alternate between women and men if both are waiting to use the bathroom.

7. 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 Parallel. 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/dict/words` on both Lec and Par. It contains 25,143 words (and is used by the `spell` command). Your task is to find all words in the dictionary that have unique letters—i.e., words in which no letter occurs more than once.

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 file `/usr/dict/words` into an array of strings. You may assume that each word is at most 25 characters long. Declare an integer array of the same length and initialize it to zeroes; this array will be used to record the lengths of words that contain unique letters.
- Examine each word, one at a time. If it has unique letters, then set the corresponding entry in the second array to the length of the word.

- After you have examined all words, go back through the array of word lengths. Count the total number of words with unique letters, and write those that have at least six unique letters to a file named `results`. At the end of the program, write the total count to standard out. (While debugging, you might want to write all words with unique letters to the `results` file.)

The first few words in the dictionary start with numbers (take a look!). Skip over these and start with "a", the first word that begins with "a". Some words are proper nouns and hence start with capital letters. Also skip over these. In short, you are to consider only those words that consist of lower-case letters.

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. Each worker should count the number of words with unique letters that it finds. Sum these `W` values during the output phase. (This avoids a critical section during the compute phase!)

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.

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 words with unique letters,
> the number found by each worker,
> the elapsed time for the compute phase,
> and the words that contain at least 6 unique letters.

Write the first three items to standard out; write the words to a file named `results` in the directory that contains your program.

Timing Tests. Execute your parallel program on Par 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 `words.c` or `words.mpd`.