Programming Assignment 1

Early Due Date: Thurs. 2/11 11:59pm, Carmen Time Zone
Due Date: Sun. 2/14 11:59pm, Carmen Time Zone
Final Due Date: Thurs. 2/18 11:59pm, Carmen Time Zone

Please read all instructions carefully. You will be graded based upon meeting all requirements stated in this assignment.

Background

The producer - consumer problem is a classic programming exercise in computer science. It has both high practical benefit as well as a straight-forward implementation. This problem is composed of two parts. First, a "producer" creates some sort of workload, storing instructions or intermediate work in a queue. Second, a "consumer" reads this queue, performs the desired work and produces appropriate output.

In this lab, we will build a serial version of producer consumer that is designed to be easily parallelizable. We will connect with the owens login nodes at Ohio Supercomputer for compilation and testing of our program. In lab2, we will test both this program, as well as a parallel version, on the owens cluster.

Problem Statement: a serial producer - consumer

Using a collection of invertible functions (provided), implement a serial version of producer consumer. This program will serve as a reference and also provide a basis for future multi-threaded implementations.

- You will be provided with the following four functions in a separate .o file, which you will need to link with your .c program to produce an executable a.out file.

  uint16_t transformA(uint16_t input_val)
  uint16_t transformB(uint16_t input_val)
  uint16_t transformC(uint16_t input_val)
  uint16_t transformD(uint16_t input_val)

main program

Your main program will create all required data structures and control the execution of two subroutines: producer() and consumer().

Main program requirements:

- other than temporary debugging statements, your main program should perform no I/O.
- in main(), declare an array of the following type in which to store your work queue:

```
struct work_entry
{
    char cmd;
    uint16_t key;
};
```

  For this version of your program, use a work queue size of 50. For your future design consideration, this value will change in later program versions.

- alternately run producer() and consumer() until all data is processed. producer() will read from the input data file and create entries in the work queue. consumer() will extract entries from the work queue and create the program output.

producer()

The producer function will: read input commands, in the form of *cmd / key* pairs, from standard input; encode the *key* using the requested transform function ("*cmd*") on the *key* provided; and, create an associated entry in the work queue.

producer() requirements:

- read all input from standard input (do not open files from within your program).
- for each line of the input file:
  - validate that the line contains a valid *cmd* / *key* pair as follows:
    - valid values for *cmd* are { A, B, C, D, X }. Input lines containing other values for *cmd* should be discarded.
    - for *cmd* values { A, B, C, D }, valid values for *key* are [0 - 1000] (note inclusive interval). Extra leading zeroes may or may not appear in the *key* field. Input lines for *cmd* values { A, B, C, D } which contain invalid values for *key* should be discarded.
    - the X *cmd* indicates end-of-data. Lines containing an X *cmd* may or may not contain a value for *key*.
  - for each valid *cmd / key* pair, other than *cmd* X, the producer should:
    - call the corresponding transformβ() function (where $\beta \in$ { A, B, C, D }) to create an encoded key. The *key* will be the parameter to transformβ(), which will $return$ the encoded *key*.
    - create a work buffer entry containing the current value of *cmd* along with it's encoded *key*.
- producer() should $return$ to main() upon filling the work queue or upon reading a *cmd* of X.

(note: be sure that you understand UNIX standard I/O before writing your program. Failure to use standard input and standard output correctly in your program will result in a score of 0 for this assignment.)

consumer()

The consumer function will: extract *cmd / key* pairs from the work queue; decode the *key* using the requested transform function ("*cmd*"); and, output the decoded *key* to standard output.

consumer() requirements:

- extract work items from the work queue;
- for each work item extracted from the work queue:
  - call the corresponding transformβ() function (where $\beta \in$ { A, B, C, D }) to decode the key.
  - print (on standard output):
    - a sequence number, starting with 1 for the first valid *cmd / key* pair read from the work queue, and incrementing by 1 for each vaid *cmd / key* pair (cumulative over multiple consumer() invocations).
    - the queue position (0 - 50) of the current work item being processed;
    - the *cmd* for the current work item
    - the encoded *key* (from the queue)
    - the decoded *key* (computed in the consumer)
    - a newline.
- consumer() should $return$ to main() upon emptying the work queue or upon reading a *cmd* of X.

(note: be sure that you understand UNIX standard I/O before writing your program. Failure to use standard input and standard output correctly in your program will result in a score of 0 for this assignment.)

Input Data Format

Each line of the data file contains the following:

- char *cmd*
- white space
- uint16_t *key*
- newline

There may exist additional blank lines anywhere within the data.

There may exist other lines after a line containing an 'X' command. Those lines should be ignored.

There may exist other data on a line after a valid *cmd / key* pair. The valid *cmd / key* pair should be processed and the extraneous data ignored.

Test files are currently available on Carmen in the Files/labs folder, and will be up-loaded to owens once our space allocation has been initialized. In the meantime, it is fine to upload the small test files to your local directory on owens.

Instrumentation

- Measure the overall run-time of your program using the Unix time(1) utility to report elapsed clock, user and system times.
- Using the Unix time(2) system call, measure
  - the total run-time of your producer module
  - the total run-time of your consumer module
  - the total run-time of your program

  Note that you will need to measure each run of producer() and each run of consumer(), accumulating the run times into a single total for each module.

Program Output

Provide the output as specified in the consumer() instructions. Your output should be neat and easily readable (although it will be long). It is suggested you make an effort to align your columns well.

A suitable example **format** for your output would resemble:

```
1       Q:0     A       0       100
2       Q:1     B       1       101
3       Q:2     C       2       102
4       Q:3     D       3       103
5       Q:4     A       4       104
.
.
.
```

Program Creation &Testing

1. Using the Ohio Supercomputer Center account and access code provided in Carmen (see announcements), create a user account for yourself which is associated with this course.
   - for account creation support contact the OSC help desk at oschelp@osc.edu or (614) 292-1800, or toll free at (800) 686-6472.
   - for general information and user instructions the OSC web site, www.osc.edu, has a considerable amount of reference material.
2. Using your new account, log into owens using ssh and your preferred terminal emulation software (bitvise, putty, iTerm, Terminator, etc.). This will place you on the owens "login nodes" (you are not yet running on the cluster).
3. On the login nodes, you may use vi, $emacs$ or another visual editor to create your programs. Alternatively, you can write your program on your personal computer, and then upload your source code to owens. (Please contact OSC support for assistance in uploading files to owens.)
4. Still on the login nodes you can continue to make minor edits, compile your program and test using the smaller test data files provided. Continue to do this until your programs completes those smaller data files (PC_data_x1 and PC_data_t00100) successfully.
5. Still on the login nodes, compile your program using the icc compiler with optimizer level 3 (-O3) turned on. Then, using both time(1) and time(2), perform a short benchmark run using test file PC_data_t01000. A reasonable serial implementation should finish in 4 minutes or less. Note that the login nodes are a time-sharing system, so your performance may vary. It is reasonable to make 3 - 5 runs and take the best time.
6. If your serial program takes longer than 8 minutes to complete the benchmark, then please schedule a code review during office hours.

Note that we will be performing all program testing on owens using the Intel C compiler, icc. If you choose to develop your code on your personal computer and then upload to owens, you must compile and test it successfully on owens. Only successful runs on owens compiled with icc will be counted for grading purposes.

Submission

Be sure to follow all submission requirements:

- within your home directory on owens create a (sub)directory called “cse5441_lab1.”
- place **only** your .c files, .h files (if any) and your makefile in this directory. These files collectively are called your “program files.”
  - your makefile should build your program when “make” is typed at the prompt from the directory containing your program files.
    - when testing your submitted program, we will provide copies of transform.o in your build directory
  - your makefile should build a single executable called “lab1_<osc id>_<osu id>,” where:
    - <osc id> is your OSC login id.
    - <osu id> is your Ohio State id.
  - use the Intel compiler (“icc”) only. Your compilation may generate a warning regarding transform.o, but should compile without any other warnings.
  - do not submit the transform.o files, or any other binary files.
- Submit your cse5441_lab1 directory from an Owens login node using the OSC submit system:
  - `https://www.osc.edu/resources/getting_started/howto/howto_submit_homework_to_repository_at_osc`.
  - submit this assignment to “lab1”