CSE 5441 Spring '24

Assignment #2 – Pthreads Programming

Please read these instructions carefully. Your grade is based upon meeting all requirements stated in this assignment.

Background and Problem Statement

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.

The producer thread reads the sequence of numbers and feeds that to the consumers. Consumers pick up a number, do some "work" with the number, then go back for another number.

Program Requirements

Start with the skeleton program provided and then re-architect it as needed. There are really two problems here: managing the bounded buffer and synchronizing it. I suggest writing and test your bounded buffer implementation first before implementing synchronization.

- Your final source codes MUST be named prod_consumer_mutex.c and prod_consumer_condvar.c
  - What to do if vim gives you errors about the swap file being present?
  - When your vim crashes without closing things properly, it leaves behind a hidden .swp file. In your case, it is probably named .prod_consumer_mutex.c.swp and/or .prod_consumer_condvar.c.swp.
  - Please note the dot in front of the file name. If you delete that swp file, you will not get the error.
- prod_consumer_mutex.c MUST use a mutex to wait when the buffer is empty or full.
- prod_consumer_condvar.c MUST use mutexes and condition variables i.e. using pthread_cond_wait()/pthread_cond_signal to wait when the buffer is empty or full.
- Please find the optimal value of producer and consumer threads that yields the best performance. Find the point at which adding more consumer threads does not significantly improve the performance.
- You may read the input from either a serial or a parallel region within your program. All input should be read from stdin as showed in the run_script.
- You program should support one "producer" thread which will perform populate a shared data structure like a stack or a queue.
- Limit the maximum size of the bounded buffer to 10.
  - The simplest way to implement this would be with a stack with one pointer as below.
  - But you are free to choose any other data structure you prefer. The data structure can be either declared globally or locally to the main function.

- int buffer[MAX_BUF_SIZE] = {0};
- int location = 0;

- You program should support having multiple “consumer” threads, which will take work from the shared data structure, perform the dummy computations.
  - The consumers MUST consume ALL the data the producers put in the shared data structure. Nothing should be left in the shared data structure once all the consumers have exited. If consumers do not read all the data, you will lose points.
  - We use a usleep to simulate this work.
    - The skeleton code calls usleep(10 * number) after each extraction
    - However, this must be changed to usleep(10000 * number) and re-tested with this change before submission. Failure to do this will be considered a sub-optimal implementation, and you will lose points.
- There are two helper functions provided that are used to print to stdout.
  - print_insertion takes the value being inserted and the insertion index/location
  - print_extraction takes the consumer number, the value being extracted, and the extraction index/location
  - The order of output NEED NOT be the same as the order of the input file.
  - Please use these helper functions inside buffer_insert and buffer_extract. These will be used by the scripts to validate your solution’s functionality.

Input Files Provided

The following files have been provided to you. All files are available on Owens in the /fs/ess/PAS2661/Lab1-Assignment directory. Please copy these to your home directory before starting to work on them.

- prod_consumer.c – The skeleton file
- run_script.sh – A script to run the final program and collate the output
- validate.py – A helper script to help validate the output of your program
- Input files – longlist and shortlist

Testing Instructions

- The script will load the necessary modules (module load gcc-compatibility/8.4.0). Use the same module to get the correct gcc compiler.
  - run_script.sh should already do this.
  - Your program should compile without warnings.
  - Points will be deducted for any warnings
- Logging in to Owens using SSH

  ssh -X -C <username>@owens.osc.edu

- When compiling and making short (less than a minute) test runs, you can use the login nodes.

- Note that the Owens HPC system only has a limited number of compute cores on each node. If you exceed this, you will see very poor performance as different threads will be competing for CPU resources. Please keep this in mind while choosing the number of threads.
- For final benchmark runs or when running with the larger value of usleep, please do it on an interactive allocation as described below.
- Getting an interactive allocation

  salloc -N <number_nodes> -A PAS2661 -p <partition> --time=<time_in_minutes>

  e.g. To request for one full compute node from the serial partition for 30 minutes, you can use the following command. I am just using 30 as an example. Please use higher values as appropriate.

  -bash-4.2$ salloc -N 1 -n 28 -A PAS2661 -p debug --time=30

  salloc: Pending job allocation 17868816

  salloc: job 17868816 queued and waiting for resources

  salloc: job 17868816 has been allocated resources

  salloc: Granted job allocation 17868816

  salloc: Waiting for resource configuration

  salloc: Nodes o0646 are ready for job <- That is the node which was allocated for you

  bash-4.2$ hostname

  owens-login03.hpc.osc.edu <- You're still on the head node when you get the control back

  bash-4.2$ ssh o0646 <- This will move you to the node that was allocated

  <….snip….>

  -bash-4.2$ hostname Now you're on the node that was allocated.

  o0646.ten.osc.edu

Report Requirements

Provide a 1–2-page report which includes the following (note that accuracy and brevity are prized, needlessly lengthy reports may lower your score).

- You will submit the PDF report on Carmen.
- In your PDF report, you should address the following points

- Provide a brief explanation describing how your changes ensure use of shared memory.
- Provide a chart or graph which illustrates the scalability (or lack thereof) of your solution.
  - Y-axis of the graph will represent time taken for execution of the program as reported by the run_script.sh for a given number of consumer threads
  - X-axis of the graph will represent the number of consumer threads.
- Explain why the values you chose gave the best performance
- Describe any unusual or unexpected results your found.

Code Submission Instructions

- The run_script.sh file will compile your program and run it for different combinations of producers and consumers with different input files.
- On successful completion, the run_script.sh file will create a sub-directory named "cse5441-pthreads-lab" and place all necessary files for submission there.
- Submit your cse5441-pthreads-lab directory from an Owens login node using the OSC submit system
  - More details about how to submit your assignment from OSC is available at the following website: https://www.osc.edu/resources/getting_started/howto/howto_submit_homework_to_repository_at_osc
- Submit this directory to "assignment-2"
  - You can use the following command to do the submission
    - **/fs/ess/PAS2661/CSE-5441-SP24/submit**
  - Note that the system will automatically close at the specified deadline for submission. You will not be able to submit your files after that.
  - Here is a sample set of steps to submit.

```
./run_script.sh

Build prod_consumer_mutex ….

<…snip…>

-rwxr-xr-x 1 osc0577 PAS2661 14576 Feb  8 11:50 a.out
drwxr-xr-x 2 osc0577 PAS2661  4096 Feb  8 11:51 cse5441-pthreads-lab
-rw-r--r-- 1 osc0577 PAS2661   205 Mar 15  2022 longlist
-rw-r--r-- 1 osc0577 PAS2661  9865 Feb  8 11:24 prod_consumer_condvar.c
-rw-r--r-- 1 osc0577 PAS2661  9864 Feb  8 11:31 prod_consumer_mutex.c
-rw-r--r-- 1 osc0577 PAS2661  9797 Feb  8 11:31 prod_consumer.c
-rwxr-xr-x 1 osc0577 PAS2661  3981 Feb  8 11:50 run_script.sh
-rwxr-xr-x 1 osc0577 PAS2661  3981 Feb  8 11:50 validate.py
```

-rw-r--r-- 1 osc0577 PAS2661 39 Mar 15 2022 shortlist

-bash-4.2$ /fs/ess/PAS2661/CSE-5441-SP24/submit

****

Hello. This script will submit your assignment assigned by Hari Subramoni to OSC

Note:

Before you run this script, please create one directory which includes all the files you want to submit

The previous submission of the same assignment from you will be replaced by the new submission

Enter the name of assignment and press [ENTER]

>>>lab1-assignment

Enter the absolute path of the directory which includes all the files of your assignment assignment-2 and press [ENTER]

You will submit the current directory if you simply press [ENTER]

>>> /fs/ess/PAS2661/CSE-5441-SP24/Submissions/lab1-assignment

Your assignment lab1-assignment is submitted successfully. Thank you!

Rubrics

Note to students: I have attempted (to the best of my abilities) to cover all possible cases and create a comprehensive rubric. I am and have been updating these based on feedback/questions I have received over the duration of this course. Even after this, it could still not cover all the questions you may have.

The student should have submitted two programs – prod_consumer_mutex.c and prod_consumer_condvar.c.

1. 70 points for the correct program without any data validation errors
    a. For a correct program, the points will be divided as follows
        i. 50 points for the correctness of the implementation (25 for each program)
        ii. 20 points for the quality of the implementation (10 for each program)
    b. If you have validation errors, you will lose a minimum of 25 points per program.
    c. If the program does not run for the grader, a minimum of 25 points will be deducted per program.
        i. Questions of the type “the program ran fine for me, but not for the grader”,

"the program did not hang for me, but it hung for the grader" etc may be considered on a case-by-case basis.

d. If you forget to submit your program(s) because you did not follow the instructions listed above, you will lose all 25 points that are meant for the program. No excuses.

2. 15 points for getting approximately linear scaling/improvements in performance with the input longlist on the Owens batch partition.
    a. The program should approximately show linear scaling/improvements in compute time as reported by the run_script.sh.
    b. If the improvement is less than 80% of linear scaling, the student will lose 15 points for the program.
3. 10 points for the report and the clarity with which the student has described the changes
4. 5 points if the program cleans up all resources it allocates
    a. The program should cleanup all resources it allocated.
    b. If it does not – i.e., if there are memory leaks, then the student will lose points based on the percentage of resources they failed to clear up.
5. Late submissions will lose points
    a. The due date will not be extended
    b. 5 points will be deducted for every 24 hours of delay for a maximum allowable delay of 48 hours.
    c. Beyond 48 hours of delay, the student will receive no points for the assignment.

Academic Integrity

Collaborating or completing the assignment with help from others or as a group is NOT permitted

Copying or reusing previous work done by others is not permitted. You may reuse the work you did as part of this class.