CS 610 Semester 2022–2023-I: Assignment 2

21st August 2020

Due Your assignment is due by Sep 3, 2022, 11:59 PM IST.

General Policies

- You should do this assignment ALONE.
- Do not plagiarize or turn in solutions from other sources. You will be PENALIZED if caught.

Submission

- Submission will be through mooKIT.
- Submit a PDF file with name "<roll-no>.pdf". Include the solution to Problem 1 in the PDF, and explain your implementations, results, and other issues (if any) for Problems 2–4. We encourage you to use the LaTeX typesetting system for generating the PDF file.
- Name each source file for Problems 2–4 "<roll-no-probno>.cpp" where "probno" stands for the problem number (e.g., "22111045-prob2.cpp").
- You will get up to TWO LATE days to submit your assignment, with a 25% penalty for each day.

Evaluation

- Show your computations where feasible and justify briefly.
- Write your code such that the EXACT output format is respected.
- We will evaluate the implementations with our OWN inputs and test cases, so remember to test thoroughly.
- We may deduct marks if you disregard the listed rules.

Problem 1 **[10 marks]**

Consider the following loop nest.

```
1 for i = 1, N-2
2  for j = i+1, i+N-2
3   A(i, i-j) = A(i, i-j-1) - A(i+1, i-j) + A(i-1, i-j-1)
```

List all flow, anti, and output dependences, if any, using Delta test. Assume all array subscript references are valid.

Problem 2 **[20 marks]**

You are given an integer array *nums* of length n which represents a random permutation of all the integers in the range $[0, n-1]$. The number of inversions is the number of different pairs (i, j) where $0 <= i < j < n$, such that $nums[i] > nums[j]$. Count the total number of inversions in parallel using p threads.

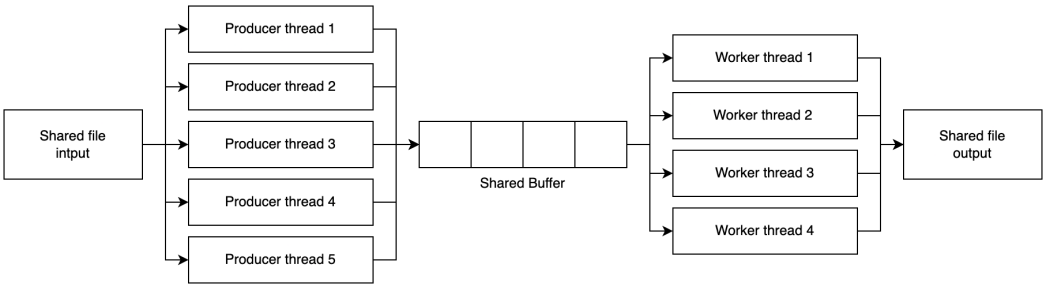
Your implementation should take two command line parameters: n and p. A function will be provided that initializes the array *nums* with random integers in the range $[0, n-1]$.

Problem 3 **[50 marks]**

The producer and consumer problem can be modeled as a synchronization problem. We will practice implementing one scenario using Pthreads. You are free to use any Pthread feature.

Consider an input file "input.txt" that contains integer numbers, each number is on a separate line. There are `P` producer threads. Each producer thread reads a number from a new line in the input file and writes that number into a shared buffer. The shared buffer can hold up to `B` numbers. There are `W` worker threads that pick a number from the shared buffer (i.e., the slot becomes empty) and check whether the number is prime. If the number is not prime, the worker thread does nothing. If the number is prime, the worker thread writes the number on a new line in a text file called "prime.txt". All the worker threads write to the same shared output file "prime.txt".

Your implementation should avoid synchronization problems and should take three command line parameters: `P`, `B`, and `W`.



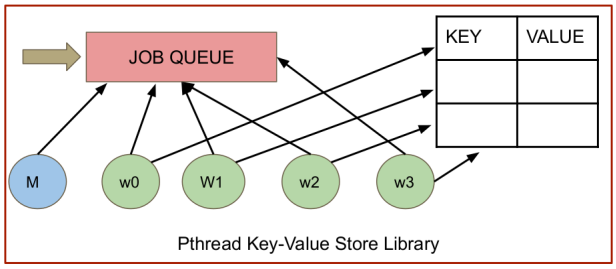
Take care of the following points.

- Two producer threads should not read a number from the same line in the input file.
- A number written to the shared buffer by a producer thread should not be overwritten by other threads.
- Two worker threads should not read the same number from the same slot of the shared buffer.
- A worker thread should not overwrite a prime number written by another worker thread in the file "prime.txt".
- A producer thread blocks until there is an empty slot in the shared buffer to write.
- A worker thread blocks until there is a non-empty slot in the shared buffer that can be read.
- The implementation will run till there are no more numbers in "input.txt". When the input file is processed, one or more producer threads will signal the worker threads that there is no more new work and the worker threads can terminate after the current batch of work is processed.

Problem 4 **[50 marks]**

You need to implement a multithreaded key-value store library using Pthreads. The store library should support Insert(key, value), Update(key, value), Delete(key), and Find(key) operations on the key-value store. Remember that the functions can be called in parallel by concurrent threads.

The following figure shows details of the key-value store. The library consists of four worker threads (W0, W1, W2, W3) and a manager thread (M). The worker threads pick operations from a "Job Queue" and performs them on the key-value store. These operations are Insert(key, value), Update(key, value), and Delete(key). The manager thread enqueues operations in the Job Queue. The worker threads indicate success or failure status of operations at corresponding locations in the Job Queue. Success is denoted by value 1 and failure by value 0. The Manager thread later consumes the status as a result of the "check status" call from the program and marks the Job Queue entry free. The library should ensure mutual exclusion when (i) worker threads operate at the same location in the key-value store, and (ii) manager and worker threads when performing enqueue and dequeue operations on the Job Queue.



The library provides the following APIs for client programs.

- `int enqueue (struct operation *op)`: This creates an entry in the Job Queue by the manager thread.

```
1 struct operation {
2  uint8_t type; // 0 - insert, 1 - update, 2 - delete
3  uint32_t key;
4  uint64_t value;
5 };
```

  Attributes `type` specifies the operation type and key, `value` specifies the corresponding (key,value) pair for an Insert or Update operation. The `value` field is irrelevant for a Delete operation.

  On success, `enqueue` returns the location id (between 0 and *max* queue size - 1) in the Job Queue. It returns -1 on an error, such as Job Queue full.

- `int check_status(int jobloc)`: This returns the status of the previous enqueue operation at job location `jobloc`. The manager thread uses the `jobloc` value and returns the operation status from the Job Queue. The manager thread also removes the corresponding entry from Job Queue after this call.

  On success, `check_status` returns the success (value 1) or failure (value 0) of operation submitted at `jobloc`. It returns -1 on an error such as operation is not complete or status is not available.

- `uint64_t find(uint32 t key)`: This returns the value corresponding to the key. The manager thread reads the value from key-value store and returns the value. The manager thread has to ensure mutual exclusion with worker threads if an operation is ongoing at that key location in the key-value store.

  On success, `find` returns the value at a key in the key-value store. It returns -1 on error such as key not present.

Design constraints:

- The size of the key-value store depends on insert/delete operations
- There is only one manager thread and four worker threads
- The size of the Job Queue is 8.

You are free to use any Pthread feature.