Le Uyen Nguyen

100 171 8086

CSE-3320.001

Professor Bakker

Programming Assignment 2: Threads

I. Part 1

Compilation: gcc -pthread part1.c

./a.out <file_name>

1. Explain the problem and identify evaluation metrics for experiments.
   - Part 1 focuses on finding the occurrences of a substring s2 in a given string s1. In the sequential approach, the program evaluates the given string from the beginning to end in order. As the given string's length increases, the running time increases. To solve this issue, exploiting concurrency using multithreading program is a better approach. To evaluate performance, we can compare and analyze the statistics of the running time taken by both single-threaded and multithreaded programs.
2. Explain your choice of threading libraries.
   - For this assignment, I choose to use pthread library. The first and foremost reason for this decision is that I am familiar with pthread more than other libraries. Following that, most of threading libraries are built upon pthread, therefore, it may improve the code's performance. Even though pthread requires more hand typing resulting in extra codes, it allows programmers more flexibility on passing arguments. Next, orthogonality is another advantage of pthread library since it does not provide

multiple functions to do the same task. For example, there is only one function to create thread. This will lead to less unambiguity on the code itself.

3. Explain the design of the experiment and develop programs for evaluation.
   - Based on the hint given in the instruction of this assignment, we can implement the multithreaded program by dividing the given string by the number of threads. Therefore, each thread will individually and concurrently work on different part of the string. The occurrences of the substring will be added to the global variable after each thread finishes its task. This will speed up the running time since the string is broken up into many shorter strings, all of which are evaluated at the same time.
4. Detail your collected experimental results.
   - On the input file 'shakespeare.txt' and the substring 'the'

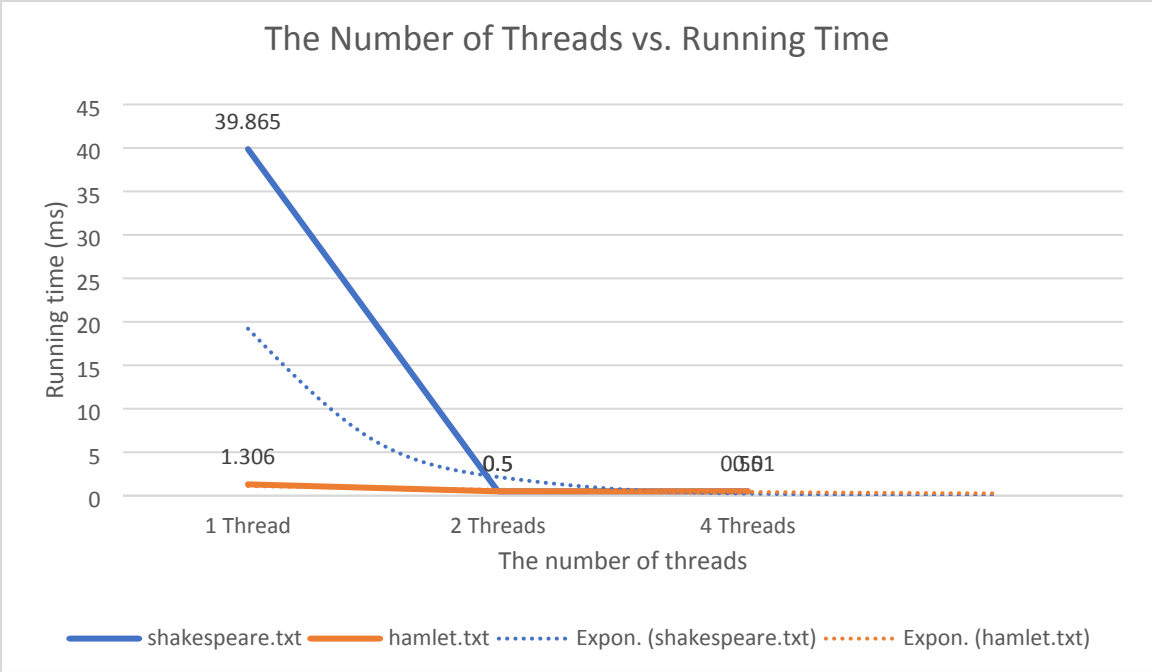
	1 Thread	**2 Threads**	**4 Threads**
Running time (ms)	39.865	0.500	0.500

   - On the input file 'hamlet.txt' and the substring 'Hamlet'

	1 Thread	**2 Threads**	**4 Threads**
Running time (ms)	1.306	0.500	0.501

5. Analyze, graph, and interpret experimental results and draw conclusions.
   - Taking the data from 'shakespeare.txt' with the substring 'the', the single-threaded program runs in 39.865 ms; meanwhile, the multithreaded program with either 2 or 4

threads finishes within 0.5 ms. Since performance improvements is based on physical concurrent execution rather than threading per se, there is no significant difference between running the program with 2 or 4 threads. However, it is clear that the parallelism significantly improves the program performance in both input files and substrings compared to the sequential solution.



II. Part 2

Compilation: gcc -pthread part2.c

./a.out <file_name>

1. Explain the problem and identify evaluation metrics for experiments.
   - Part 2 mentions the producer and consumer problem in parallel programming. There will be two threads represented by producer and consumer concurrently insert and take out items from a finite-size buffer. Nevertheless, since it is a limited-space buffer, if producer tries to insert item when there is no space available or if the consumer pulls item out when there is nothing left, the CPU cycles will be wasted.

Therefore, this algorithm requires the use of condition variables to signal when the producer can put the item in and when the consumer can take item out.

2. Explain your choice of threading libraries.
    - For part 2, I again choose pthread library. Apart from all above-mentioned reasons in the question 2 of Part 1, the simplicity and orthogonality of pthread helps to reduce the chance of losing signal between threads. Additionally, the context switch between threads can be faster when using pthread because it is constructed in user space, and thence, does not need to deal with the kernel overhead when implementing process switching. This becomes helpful for the producer/consumer algorithm since it uses the condition variables for synchronization.
3. Explain the design of the experiment and develop programs for evaluation.
    - For the producer/consumer problem in Part 2, the condition variable is needed to prevent wasting CPU cycles. The global variable representing the buffer capacity is used to manage when to issue the waiting signal. To demonstrate, the producer thread locks the mutex and ensures that the buffer has space available to put the items in. The capacity is incremented after each insertion in the producer thread. If the capacity reaches its limits, the producer then calls pthread_cond_wait() to wait until there is a signal of available space in the buffer and to unlock the mutex so that the consumer thread can work on it. Otherwise, after the producer has filled in the item, it calls pthread_cond_signal() to inform the waiting consumer. On the other hand, the consumer thread concurrently waits for the signal of item available in the buffer to be taken out. Once it is waked up, the consumer acquires the mutex and guarantees the buffer is not empty. The capacity is decremented as one item is pulled from the

buffer. Following this, pthread_cond_wait() is called when the capacity equals 0, representing the empty buffer; differently, the consumer calls the pthread_cond_signal() to wake up the producer.

- Since the producer thread reads input from file, it constantly checks if the file has ended. The global variable is used to keep track of this condition. When the producer reaches end-of-file symbol, it sets the global variable to true and stops reading from file. At the same time, the consumer thread also observes this variable; once it is flagged, the consumer thread will exit.

4. ~~Detail your collected experimental results.~~
5. ~~Analyze, graph, and interpret experimental results and draw conclusions.~~