University of Colorado at Colorado Springs
Spring 2020 Operating Systems Project 2 - POSIX Thread Programming

Instructor: Yanyan Zhuang
Total Points: 100
Out: 3/6/2020
Due: 11:59 pm, Sunday, 3/29/2020

Introduction

The purpose of this project is to practice `Pthread` programming by solving various problems. The objectives of this project is to:

1. Get familiar with `Pthread` creation and termination.
2. Learn how to use mutexes and conditional variables in `Pthread`.
3. Learn how to design efficient solutions for mutual exclusion problems.

Project submission

For each project, please create a zipped file containing the following items, and submit it to Canvas.

1. A report that includes (1) the (printed) full names of the project members, and the statement: **We have neither given nor received unauthorized assistance on this work**; (2) the name of your virtual machine (VM), and the password for the `instructor` account (details see *Accessing VM* below); and (3) a brief description about how you solved the problems and what you learned. The report can be in txt, doc, or pdf format.
2. The `Pthread` code and files that contain your test cases.

Accessing VM. Each team should specify the name of your VM that the instructor can login to check the project results. Please create a new user account called `instructor` in your VM, and place your code in the home directory of the instructor account (i.e., `/home/instructor`). Make sure the `instructor` has the appropriate access permission to your code. In your project report, include your password for the `instructor` account.

Tasks

Task 1 (30 points)

Given two character strings `s1` and `s2`. Write a `Pthread` program to find out the number of substrings, in string `s1`, that is exactly the same as `s2`. For example, if `number_substring(s1, s2)` implements the function, then you have `number_substring(‘‘abcdab", ‘‘ab") = 2`, `number_substring(‘‘aaa", ‘‘a") = 3`, and `number_substring(‘‘abac", ‘‘bc") = 0`. The size of `s1` and `s2` (`n1` and `n2`), as well as their data are the input by the user, via a file. Assume that `n1` is at least twice as long as `n2`.

Attached code `substring_sequential.c` is a sequential solution of the problem. `read_f()` reads the two strings from a file named `string.txt`, and `num_substring()` calculates the number of substrings. Write a parallel program using `Pthread` based on this sequential solution.

HINT: Strings `s1` and `s2` are stored in a file named `string.txt`. String `s1` is partitioned among `NUM_THREADS` threads to concurrently search for matching with string `s2`. After a thread finishes its work and obtains the number of local matchings, this local number is added into a global variable showing the total number of matched substrings in string `s1`. Finally, this total number is printed out. You can find an example of the file `string.txt` in the attached source code.

MORE HINT: If you partition `s1` among threads, you will miss those `s2`'s at the partition boundaries. Instead of partitioning the data, can you partition the task(s)?

Task 2 (30 points)

Use ***condition variables*** to solve a producer-consumer problem. Here we have two threads: one producer and one consumer. The producer reads characters one by one from a string stored in a file named `message.txt`, then writes sequentially these characters into a circular queue. Meanwhile, the consumer reads sequentially from the queue and prints them in the same order. Assume a buffer (queue) size of 6 characters. Write a `Pthread` program using condition variables.

Task 3 (40 points)

Read attached code `list-forming.c` and modify the program to improve its performance. In this program there are `num_threads` threads. Each thread creates a data node and attaches it to a global list. This operation is repeated `K` times by each thread. The performance of this program is measured by the program runtime (in microsecond). Apparently, the operation of attaching a node to the global list needs to be protected by a lock and the time to acquire the lock contributes to the total run time. Try to modify the program in order to reduce the program runtime.

Your tasks

1. Implement a modified version of `list-forming.c` and name it `my_list-forming.c`.
2. Verify that your program achieves better performance than the original version by using different combinations of `K` and `num_threads`. Typical values of `K` could be 200, 400, 800, ... Typical values of `num_threads` could be 2, 4, 8, 16, 32, 64, ... Draw diagrams to show the performance trend.

3. In the report, explain your design and discuss the performance results.

HINTS:

1. Since the problem does not require a specific order of the nodes in the global list, there are two ways to add nodes. First, a node could be added to the global list immediately after it is created by a thread. Alternatively, a thread could form a local list of `K` nodes and add the local list to the global list in one run. Will the choice in how to add nodes make a difference? Why?

2. The original program uses `pthread_mutex_trylock`. Will the use of `pthread_mutex_lock` make a difference? Why?

3. Each thread is pinned to a specific CPU (according to the thread id). Will letting threads run on all CPUs make a difference? Why?

Instructions

1. Power off your VM and change to VM setting to use 4 virtual CPUs.

2. Verify that you VM has 4 virtual CPUs:

   ```
   $ cat /proc/cpuinfo
   ```

   You should see information about 4 CPUs (processor: 0-3).

3. To compile the program:

   ```
   $gcc list-forming.c -o list-forming -pthread -D_GNU_SOURCE
   ```

Resources

GDB tutorial: `https://web.eecs.umich.edu/~sugih/pointers/summary.html` This is a great tool when you need to debug segmentation faults.