ITP30002 Operating System

Homework 3

Multithreaded TSP Solver

Date assigned:
May 19 (Tue)

Due date:
9 PM, May 26 (Tue)

Overview

- You are asked to write `mtsp.c`, a multithreaded version of a parallel Traveling Salesman Problem (TSP) solver
  - the program runs multiple threads, one of which is working as a user interface, another as a job producers and the others as job consumers
  - construct the program according to the given functional and design requirements

- Submission
  - Deadline: 9 PM, May 26 (Tue)
  - Late submission: no late submission will be accepted
  - Deliverables: (1) write-up, and (2) source code
  - Submission site: Hisnet

Notes

- We use the same TSP data as used for Homework 2
  https://github.com/hongshin/OperatingSystem/tree/hw2

- Recommend to use Peace `http://peace.handong.edu` or an equivalent environment
  - Ubuntu 16.04.6 LTS Kernel 4.15.0, GCC 5.4.0, Intel Xeon CPU 2.2 GHz 40 cores, 32 GB RAM
  - Your program should be built and executed successfully because TAs will test your program using Peace

Your Assignment

- Construct `mtsp.c` that receives a TSP instance and then finds a shortest route using concurrent executions of children processes

- Requirements
  - Your program must provide all expected functionalities
    - Input
    - Interactive user interface
  - You must implement the program by following the given design and applying proper programming features/mechanisms
    - Producer-consumer with bounded buffer

Program Design Requirement (1/2)

- Main thread (User interface thread)
  - Read the command-line inputs and creates a producer thread and consumer threads to start parallel solving of the given TSP instance
    - The program creates a single process running multiple threads
  - In a middle of solving, the main thread receives commands from the user and generates the responses.
- Producer thread
  - The program runs one producer thread that creates subtasks until all task gets done
  - The producer thread delivers subtasks to consumer threads via a bounded buffer
    - a subtask is specified by a prefix of routues
    - a subtask is to explore all routes (permutations) having a certain prefix
    - the number of routes to explore in a subtask should be 11!
    - each subtask is to explore a unique set of routes (no overlapped with other subtasks)

Program Design Requirement (2/2)

- Consumer threads
  - The program runs *N* numbers of consumer threads concurrently, and *N* is initially given from the command line argument
  - A consumer thread takes a subtask from the bounded buffer and processes the subtask, and then takes another subtask and repeat this job until the termination

Functional Requirement (1/2)

- Input
  - Receive (1) a filename of a TSP instance data, and (2) the initial number of consumer threads
    - E.g., `$ ./mtsp gr17.tsp 8`
  - A TSP data file with N cities (City 0 to City N-1) consists of N lines where each line contains N non-negative integers for $13 \leq N \leq 50$.
    The j-th integer at the i-th line is the weight between City i to City j
  - The limit of children processes at a time is in between 1 and 8 (inclusive)
- Output
  - When a user raises a termination signal (i.e., Ctrl+C) or all subtasks are done completely, print out the following information to the standard output and then terminates the program:
    - the best solution (a route and its length) upto the point
    - the total number of checked/covered routes upto the point

Functional Requirement (2/2)

- Interactive user command
  - The user can give the following three commands via the standard input in a middle of program execution
    - `stat` : print the best solution up to the moment, and the number of checked routes
    - `threads`: print the information of all consumer threads including thread ID, the number of subtasks processed so far, the number of checked routes in the current subtask
    - `num` *N*: change the number of consumer threads into *N* ($1 \leq N \leq 8$)
      - use `pthread_cancel()` to stop a running thread
      - if a consumer thread stops before completing an assigned subtask, the subtask must be assigned to another consumer

Evaluation

- Criteria
  - fulfillment of requirements 50%
  - novelty of analysis and discussion 30%
  - clarity in technical description 20%

- Write-up
  - up to 2 pages in the given template
  - elaborate your design of synchronization that ensures multiple threads can work correctly and efficiently on shared data structures
  - show that your program fulfills the requirements
  - discuss issues and ideas as you had in doing this homework
  - submit a PDF file of your write-up