Systems Software/Programming – Lab Manual

Lab 9 – Concurrency Control

Write C program for each of the problem below. After executing each of the C program, you will need to capture the output in text file format.
C program file will be named as StudentID_Lab9_x.c
Text file with output captured will be names as StudentID_Lab9_x.txt

Note: None of the below problems require thread synchronization mechanisms. These assignments will teach how to divide the computation work among different threads so that it can be done with improved run time.

Problem 1: Write a C program StudentID_Lab9_1.c using POSIX thread to compute square root of integers from 0 to 99 and return an array of double sroot[100] with computed value of square roots:

- Main thread will create 10 threads and stores threadID in an array. Main thread must wait till all the threads are completed
- Each of the threads will calculate square root of 10 integers by calling sqr_root function depending on which thread it is
    - E.g. thread 0 (first thread) will calculate square root of 0 to 9, thread 1 (2nd thread) will calculate square root of 10 to 19 and so on.
    - Once square root is of an integer is calculated it will stored in an array at an offset pointed by the input value e.g. if thread 1 has finished calculating square root of 15 then it will place calculated value at sroot[15].
- While main thread is waiting for all the thread to finish, main thread should display a short message to the user and then display the results of the computation when they are ready.

Problem 2: Compare serial versus parallel algorithm runtime for matrix summation.
Parallel Algorithm:
Write a C program StudentID_Lab9_2.c using POSIX thread for summation of matrix mat[ROWS][COLS] by a number of threads provided by user.

- Main thread accepts from user input number threads to use (num_threads). num_threads must be less than ROWS
- Main thread create number of threads as given in num_thread.
- Each thread will:
    - Each thread will be assigned specified number of rows of a matrix to calculate the sum. i.e. number of rows assigned to each thread will be ROWS/num_threads. if ROWS value is not multiple of num_threads then you need to handle that situation by assigning additional rows to some threads. e.g. ROWS = 10 and num_thread=4 then 2 rows each will be first allocated to all 4 threads but then we are left with 2 more rows which should be assigned to 2 out of 4 threads.
    - Local calculated sum value must be returned back from thread to main thread
- Main thread will access the sum value returned from each thread and display the final total sum of the entire matrix.

Serial Algorithm:
Write the serial algorithm in StudentID_Lab9_2.c for summation of matrix. For large matrices (e.g. 500x500 etc) compare the time difference between parallel versus serial algorithm.

Get the runtime information for both Serial and Parallel version by adding the following code in your programs:
You can measure the time it took to run the program using following code:

```
/* this function returns the time of day in microsecond */
uint64_t get_gtod_clock_time ()
{
    struct timeval tv;
    if (gettimeofday (&tv, NULL) == 0)
            return (uint64_t) (tv.tv_sec * 1000000 + tv.tv_usec);
    else
            return 0;
}

/* At the beginning of main function declare variables */
uint64_t start_time_value, end_time_value, time_diff;
```

```
    /* Get the start time */
    start_time_value = get_gtod_clock_time();

    -------- your code ------------------------

    /* Get the end  time */
    end_time_value = get_gtod_clock_time();

/* Time difference */
time_diff = end_time_value - start_time_value;
/* display the difference */
printf("%"PRIu64"\n", time_diff);
```

Compare Runtime:

Note the different in output for time_diff value between Serial and Parallel version an algorithm for different values of num_threads and ROWS/COLS