Synchronization

In the present laboratory exercise, please use the common synchronization mechanisms to solve synchronization problems in multiplex applications based on the POSIX threads standard.
The mechanisms to be used are (a) mutexes, semaphores, and condition variables defined by POSIX, (b) atomic operations, as defined by the hardware, and exported to the programmer via built-in of the GCC compiler.

1.1 Synchronization to existing code

Familiarize yourself with the use of POSIX threads by studying the pthread - test.c template given to you.
The simplesync.c program is running, which is as follows: After initializing a variable val = 0, it creates two threads that run at the same time: the first thread increases the value of the val variable by N times, the second decreases it N times by 1. The threads do not synchronize their execution.
The following are required:

• Use the provided Makefile to compile and run the program. What do you notice? Why?

• Study how two different executable simplesync - atomic, simplesync - mutex are generated from the same simplesync.c source code file.

• Extend the code of simplesync.c so that the execution of the two threads in the executable simplesync - mutex is synchronized using POSIXmutexes. Confirm the correct operation of the program.

• Extend simplesync.c code so that the execution of the two threads in the executable simplesync - atomic is synchronized using individual GCC functions. Confirm the correct operation of the program.

Questions

1. Use the time (1) command to measure the execution time of executables. How does the execution time of executables synchronize compare to the execution time of the original program without synchronization? Why?
2. Which synchronization method is faster, the use of individual functions or the use of POSIX mutexes? Why?
3. Which processor commands translate the use of individual GCC functions into the architecture for which you are compiling? Use the CS parameter of the GCC to generate the Assembly intermediate code, together with the -g parameter to include source code information (e.g., ".loc 1 63 0"), which may make it easier for you. See the make command output for how to compile simplesync.c.
4. Which commands translate the use of POSIXmutexes into the architecture for which you are compiling? Give an example of compiling the pthread_ mutex_lock () function in Assembly, as in the previous query.

1.2 Parallel calculation of the Mandelbrot set

You are given a program that calculates and extracts the terminal images of the entire Mandelbrot (Fig. 1).

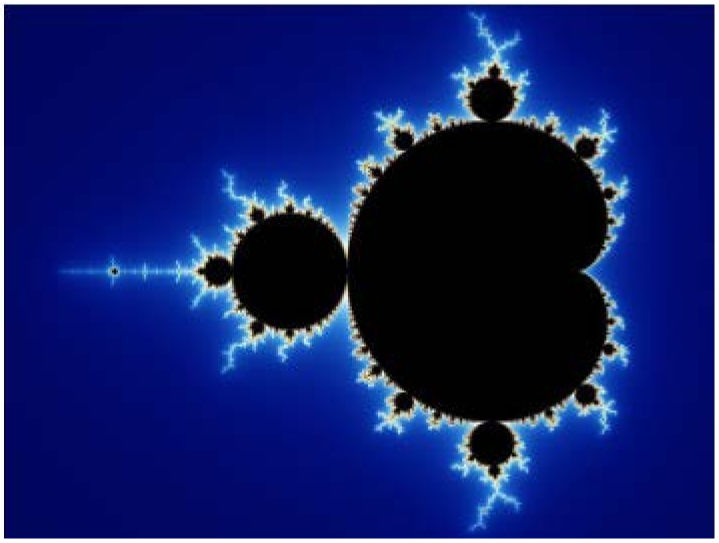

Figure 1

The Mandelbrot set is defined as the set of points at the complex level, for which the sequence z n + 1 = z n 2 + c is blocked. A simple algorithm for its design is as follows: We start by mapping the drawing surface to an area of the complex plane. For each pixel we take the corresponding complex c and repeatedly calculate the sequence z n + 1 = z n 2 + c, z 0 = 0 until | z n | > 2 or exceed a predefined value. The number of repetitions required corresponds to the color of the specific pixel. In mandel.c you are given a program that calculates and designs the whole Mandelbrot text terminal, using colored characters. For its function it is based on the library mandel - lib. {C, h}. The library implements the following functions:

• mandel_iterations_at_point (): Calculates the color of a point (x, y) based on the above algorithm.
• set_xterm_color (): sets the color of characters that are exported to the terminal.

The output of the program is a block of characters, column x_chars and row y_chars. The program repeatedly compute_and_output_mandel_ line () to print the block line by line.

Request the extension of the mandel.c program so that the calculation is distributed in NTHREADS POSIX threads, indicative value 3. The distribution of the computational load is done in order: For n threads, the i-th (with i = 0,1,2, ..., n - 1) undertakes the series i, i + n, i + 2 × n, i + 3 × n, ...

The required message synchronization is done by semaphores provided by the POSIX standard, including the header file <semaphore.h>. See more in the manual pages sem_overview (7), sem_wait (3), sem_post (3). The NTHREADS value will be given at runtime as an argument to the command line.

Note: If your terminal is left with the wrong color in the text due to the execution of the program [e.g. purple characters on a black background], use the reset command to return it to its original setting.

Questions

1. How many semantics are needed for the synchronization scheme you are implementing?
2. How long does it take to complete the serial and parallel program with two calculation threads? Use the time (1) command to time the execution of a program, e.g., time sleep 2. To make sense of the measurement, try it on a machine with a dual-core processor. Use the cat / proc / cpuinfo

command to see how many computing cores a machine has.
3. Does the parallel program you created show acceleration? If not, why not? What is the problem with the synchronization scheme that it has optimized? Note: How big is the critical section? Does it need to contain both the calculation phase and the output phase of each line generated?
4. What happens in the terminal press Ctrl-Cell program is running? What situation is left, in terms of the color of the letters? How could you extend your mandel.c to ensure that even if the user presses Ctrl-C, the terminal will return to its previous state?