Inge5SE – Parallel Programming 10/2023 – LAB2 : PThreads Synchronisation

Alexandre Berne, Etienne Hamelin

aberne@omnesintervenant.com, ehamelin@omnesintervenant.com

0. Know your hardware

During this course, you will measure the performances of many programs running under Linux. You'd better use a native Linux machine, or a Mac, or if not available, use Windows' Subsytem for Linux (WSL, https://doc.ubuntu-fr.org/wsl) or Oracle Virtualbox.

For your performance measurements to be actually useful, **always** check that:

- code are evaluated on the same machine for the whole lab,
- your machine runs at least 4 CPU cores (esp. in Virtualbox),
- your laptop is plugged in, and battery charged (otherwise, the OS might activate power-saving mode),
- minimize background load (quit all games, interaction-heavy web page, etc.),
- measure at least twice, to ensure that measurements are more or less constant.

The code is provided in the "source" directory; please modify only in the "work" directory; place your report (as a PDF file) in the "report" directory. When you're ready, modify the "submit.sh" script with your names, run it: it will put your work and report in a .tar.gz archive file. Submit the tar.gz file on the boostcamp interface.

Q1: *What is your OS/hypervisor system configuration?*
How many CPU physical and logical cores does your PC run? How much RAM memory?
Use the commands lscpu, lsmem, cat /proc/cpuinfo, and cat /proc/meminfo to answer.

1. Shared variables

1.1 Adding numbers in different threads

The first exercise will help us to understand the problem when using shared variables in parallel programming (race condition)

In the sum_value_threads.c code, we can find 4 threads incrementing a global variable.

Q2: Read the original code in the sequential folder, compile it and run it on your target. Is the displayed result correct? Why?

Q3: Measure the values of real, user, sys, $\frac{user+sys}{real}$ *of the original (in the sequential folder) code execution.*

Q4: Using thread synchronisation functions, modify the code sum_value_threads.c (in the work folder) to solve the problem identified.

Q5: Give the values of real, user, sys, $\frac{user+sys}{real}$ *of the modified code. Compare the execution times with those of Q3. Explain the difference between those measurements.*

1.2 Road to prime numbers

In the second exercise, we will count the number of prime numbers from 2 to 10000000. The code prime_numbers_threads.c in the folder is empty and contains only a skeleton of the architecture to respect. The thread number will vary from 1 to 20.

Q6: The function is_prime() will check if the number in argument is a prime number or not. Write this function and test it from 2 to 10000000 in the main() function to validate (expected : 664579)

Q7: Give the values of real, user, sys, $\frac{user+sys}{real}$ *of the sequential code execution.*

Q8: Using the arguments argc and argv in the main() function, get the number of threads from user. Write the thread_runner() function.

Q9: Trace the curve of ***real****, as a function of the number of threads* $real = f(n_{th})$

Q10: Using Amdahl's law, estimate the fraction of "perfectly parallel" code as a function of the number of threads and plot those results.

2. Producer/consumer problem

In a big bakery, several bakers are producing bread loaves, and several consumers are buying bread. They all use the same basket to store the loaves ready to be sold.

Today is a busy day, with million loaves to produce and to sell! Can we sell all loaves and pay for the wheat?

In source/3-prod-cons you will find an implementation of the famous "producer-consumer" problem, where a queue, or FIFO (first-in, first-out) structure, is used to store items. Several producers write on the queue, while several consumers read from the queue.

Q11: Build, then run, the prod-cons program using the "make time" command. Explain the difference between the total produced and consumed.

Q12: In the work directory, change the implementation of the fifo module so that concurrent accesses to shared data are protected with a mutex. Verify that your implementation is now correct, and try to improve performance.