CS4540 – Operating Systems

Assignment 5. Process Synchronization

Introduction

Monte Carlo methods (or *Monte Carlo experiments* [or *Monte Carlo simulations*]) are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results; typically one runs simulations many times over in order to obtain the distribution of an unknown probabilistic entity.

[… A] *simulation* is a fictitious representation of reality, a *Monte Carlo method* is a technique that can be used to solve a mathematical or statistical problem, and a *Monte Carlo simulation* uses repeated sampling to determine the properties of some phenomenon (or behavior). […D]istinctions [between these terms] are not always easy to maintain. [http://en.wikipedia.org/wiki/Monte_Carlo_method]

A Monte Carlo simulation provides an interesting way of calculating π. The algorithm works as follows:

1) Suppose you have a circle inscribed within a square, as shown in Figure 1. The radius of this circle is 1.
2) Generate a series of random points as simple (x, y) coordinates. These points must fall within the Cartesian coordinates that bound the square. Of the total number of random points that are generated, some will occur within the circle.
3) Estimate π by performing the following calculation:
   π = 4× (number of points in circle) /
   / (total number of points)

   Note that the above formula is derived from the following facts:
   a) (circle_area) / (square_area) $=_{approx..}$ (number of points in circle) / (number of points in square)
   b) For r = 1, (circle_area) / (square_area) = $\pi r^2 / (2r)^2 = \pi / 4$.
      Hence: π = 4 * (circle_area) / (square_area)

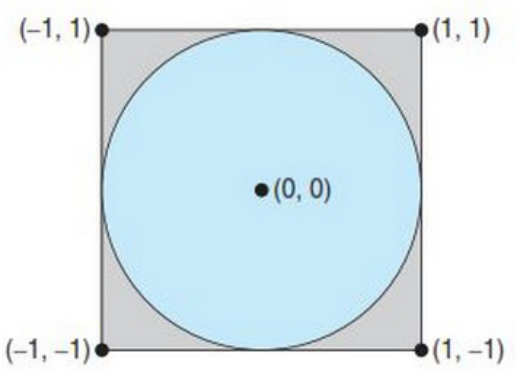

Figure 1. Calculating π with the Monte Carlo simulation.

Problem Specification

Write a multithreaded version of the above algorithm that creates ten threads, each of which: (i) generates a predefined number of random points, (ii) determines if the points fall within the circle, and (iii) counts the number of points that fall within the circle.

Each thread will have to update the shared global count *circleCount* of all points that fall within the circle. Protect against race conditions on updates to *circleCount* by using mutex locks.

When all thread have exited, the parent thread will calculate and output the estimated value of π.

Extra Credit (10%)

Print (with proper labels) the estimated value of π for *each* thread before it exits. This will allow us to see different estimates of π: (i) as calculated individually by each thread, and (ii) as estimated by all threads together.

Hints and Implementation Requirements

1) Recall that in an x–y Cartesian coordinate system, the circle centered at the point (0, 0) and with the radius r is the set of all points (x, y) such that: $x^2 + y^2 = r^2$. A point (x, y) falls within the circle iff $x^2 + y^2 \leq r^2$.
2) Use the constant NR_PTS for the total number of points; set it to 1,000,000.
3) Use the constant NR_THREADS determining the number of threads (used to generate random points and count the number of points that occur within the circle).
4) Note that mutex locks are available in Pthreads.

SLC Report Requirements

For each program write a full SOFTWARE LIFE CYCLE (SLC) report (analogous to the SLC report presented in class).

Please note that your reporting job is easier due to the following simplifications:

1) The PROBLEM SPECIFICATION section (Step 1) should include just a copy of the given problem(s).
2) The PROGRAM STRUCTURE DESIGN section (Step 2) will have a few modules to name (Substep *2.1. Modules and Their Basic Structure*), and a few modules to provide pseudocode for (Substep *2.2. Pseudocode for the Modules*).
3) The sections for RISK ANALYSIS (Step 3), VERIFICATION (Step 4), REFINING THE PROGRAM (Step 7), PRODUCTION (Step 8) and MAINTENANCE (Step 9) can include just 1-2 sentences (analogous in the SLC report presented in class).
4) In the CODING section (Step 5) you will probably have just 2-3 code refinement levels (the first will just add function headers and trailers, as shown in class).
5) Due to the simple structures of the programs, the TESTING section (Step 6) should be pretty easy. But document it well.
6) Using mono-spaced fonts like (Courier) for source code enhances code readability and quality. Please use only this type of font in your report for the source code.

Coding, Running and Submission Requirements

1) Follow *C Code Style Guide* and use the proper programming style it requires, including comments, blank lines, indentations, spaces, etc.
2) All programs must be compiled and executed on Ubuntu running within the Oracle VirtualBox.
3) Remember about *Assignment Submission Instructions* (guidelines), to be followed for each program of this assignment.
4) In addition to what *Assignment Submission Instructions* require, provide a *makefile*, and, if needed a README file explaining how to compile and run your program.

Submission Checklist

For each program you need to submit:

1) the SLC report (including 9 SLC steps, with as many pseudocode and code refinements as needed);
2) *makefile*, and, if needed a README file explaining how to compile and run the program;
3) the files created by the *script* command (including program output to the terminal, if any);
4) the output files (if any in addition to the output to the terminal)

Submit your complete assignment package (all files) via Elearning as a *zip* file. Please use **<hw#_lastname.zip>** as the naming convention for zipped file.

-------- Good luck! -----