

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Operating System Principle |
| **Semester**: | 2nd semester of the academic year **2019-2020** |
| **Major**: | Software Engineering |
| **Class**: | 2018 |
| **Student Name**: |  |
| **Student ID:** |  |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Process synchronization and deadlock | | | |
| Date | | May, 2020 | Type | | □Confirmatory  √ Design  √ Comprehensive |
| 1. **Objective & Requirements**    1. Understand the concept of process synchronization; understand the concept of deadlock    2. Learn how to use the synchronization mechanisms provided in Linux and pthread library, i.e. the mutex lock and semaphore    3. Can use mutex lock and/or semaphore to solve real synchronization problems | | | | | |
| 1. **Experimental environment (**platform and software**)**   VirtualBox+Ubuntu | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results) 2. Tasks for this lab   An interesting way of calculating π is to use a technique known as Monte Carlo, which involves randomization. This technique works as follows: Suppose you have a circle inscribed within a square, as shown in the following figure. (Assume that the radius of this circle is 1.)    • First, 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.  • Next, estimate π by performing the following calculation:  π = 4 × (number of points in circle) / (total number of points)  Write a multithreaded version of this algorithm that creates several separate threads to generate a specified total number of random points. These threads will count the number of generated points and points that occur within the circle and store those counted numbers in global variables. When the number of generated points reach the specified upper bound, each thread will exit, and the parent thread will calculate and output the estimated value of π. Additional requires are as follows:   1. You must protect against race conditions on updates to the shared global variables by using mutex locks or binary semaphores 2. Each global variable has an associated mutex lock, and your program should try to maximize the degree of thread concurrency, by keeping each critical section as short as possible 3. Experiment with different numbers of random points generated. As a general rule, the greater the number of points, the closer the approximation to π. 4. Experiment with both single-threaded and multi-threaded versions of the monte carlo algorithm for calculating pi. With the same total number of random points, compare the time cost and approximation precision through sufficiently many experiments. Explain whatever difference you may discover. 5. Please provide your procedure to perform the tasks and source codes. | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems） | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |