KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN MẠNG MÁY TÍNH VÀ TT DỮ LIỆU

Hướng dẫn Lab 6.1 – Cấu hình CPU

Kiến trúc đa nhân sẽ khó phát huy được hiệu quả nếu như hệ điều hành không thể khai thác toàn bộ nhân cân bằng và liên tục; cũng như một chương trình mang tính chất tuần tự sẽ không thể tăng tốc hoàn thành.

Trong LAB này, bài toán tìm số PI theo phương pháp Monte Carlo được giới thiệu, trong đó số điểm sinh ngẫu nhiên càng lớn sẽ tăng độ chính xác của kết quả. Nhiều tiểu trình được sinh ra và tham gia tính toán, cũng như phân bố chạy trên nhiều nhân sẽ rút ngắn thời gian hoàn thành.

Mục tiêu	**Lý thuyết liên quan**	**Tài nguyên**
Kiểm tra cấu hình CPU	Ch5: CPU Scheduling	https://github.com/Trantin84/LAB_IntroOS (mã nguồn ví dụ). Sử dụng image Ubuntu 16 / 18 Project "PI – Monte Carlo"
Bài toán Monte Carlo		
Đánh giá hiệu suất		

Yêu cầu sinh viên: Hiểu lý thuyết Lập lịch CPU đa nhân. Hiểu và thực thi các đoạn mã đã cung cấp. Phân tích và Áp dụng cho các bài toán đa nhiệm. Hiểu bài toán ước lượng giá trị PI.

Đánh giá sinh viên: Hỏi đáp các vấn đề lý thuyết. Kỹ năng thực hành. Bài tập.

Yêu cầu nộp bài: các tập tin mã nguồn .c và tập tin khả thực thi .out của các Yêu cầu trong buổi thực hành và Bài tập cuối hướng dẫn trong thời gian cho phép của giảng viên.

Preferences

[1] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, [2018], Operating System Concepts, 10th edition, John Wiley & Sons, New Jersey.

Programming Problems of Chapter 4 and Chapter 6.

[2] Wikipedia, [2021], Monte Carlo method,

Access https://en.wikipedia.org/wiki/Monte_Carlo_method

[3] Linux manual page, [2021], on sched_setaffinity(2)

Access https://man7.org/linux/man-pages/man2/sched_setaffinity.2.html

[4] Techmint, [2021], 9 Useful Commands to Get CPU Information on Linux

Access https://www.tecmint.com/check-linux-cpu-information/

[5] Linux manual page, [2021], on gettid(2)

Access https://man7.org/linux/man-pages/man2/gettid.2.html

Thông tin hệ thông.

Yêu cầu 1: Kiểm tra cấu hình hệ thống qua CLI.

Hướng dẫn: Thực thi các lệnh sau đây: (có thể lược giãn đối số để hiển thị toàn bộ thông tin).

```
$ lscpu
$ lscpu | egrep -i 'core.*:|socket'
        Thread(s) per core:    2
        Core(s) per socket:    2
        Socket(s):             1
```

* Để thay đổi số nhân và số luồng tính toán của CPU, với máy ảo, cần phải tắt máy ảo, cấu hình CPU và khởi động lại.

* Với máy chủ, tham khảo Yêu cầu 3 để cấu hình số nhân CPU được tham gia tính toán.

Yêu cầu 2: Thực thi chương trình tính giá trị số PI và khảo sát thời gian thực thi với số tiểu trình thay đổi: 1, 2, 4, hoặc nhiều hơn.

Hướng dẫn: Tải về tập tin ***LAB_IntroOS/LAB_6/task1_1.c***, biên dịch và thực thi. Lưu ý rằng khi so sánh về thời gian chạy, cần thực hiện trên cùng một máy tính và tránh thao tác hay chạy chương trình khác cùng lúc đó.

Trong lời gọi sau đây, tiến trình sẽ tạo ra 10 tiểu trình (tham số argv[1]) và mỗi tiểu trình thử 1.000.000 điểm (tham số argv[2]).

- Dòng 40 sẽ lặp lại n_thread lần thực thi dòng 41 để tạo ra số tiểu trình đã yêu cầu khi gọi thực thi (giá trí đối số argv[1]) và yêu cầu mỗi tiểu trình sinh ra argv[2] điểm ngẫu nhiên.

- Dòng 43 sẽ tạo ra n_thread lời gọi đồng bộ (dòng 44) để kết thúc các tiểu trình đã tạo ra sau khi chúng hoàn tất nhiệm vụ.

- Dòng 58 sinh số gieo theo thời gian hệ thống.

- Dòng 65 sinh ra giá trị toạ độ trục hoành của điểm ngẫu nhiên, thuộc đoạn [-1, 1].

- Tương tự, dòng 66 sinh ra giá trị toạ độ trục tung, cũng thuộc đoạn [-1, 1].

- Dòng 68 tính khoảng cách điểm ngẫu nhiên đếm Gốc toạ độ. Do có sử dụng hàm sqrt() nên cần thư viện math.h và đối số -lm khi biên dịch.

- Dòng 69: Nếu điểm này nằm trong vòng tròn Monte Carlo thì đếm nó vào counter.

```
40    for (int i = 0; i < n_thread; i++)
41      pthread_create( & tid[i], NULL, runner, (argv[2]));
43    for (int i = 0; i < n_thread; i++)
44      pthread_join(tid[i], NULL);
58    srand((unsigned int) time(NULL));
65      x = -1 + ((float) rand() / (float)(RAND_MAX)) * 2;
66      y = -1 + ((float) rand() / (float)(RAND_MAX)) * 2;
68      distance = sqrt(x * x + y * y);
69      if (distance <= 1.0) counter++;
```

```
$ gcc –o pi.out task1_1.c –lpthread -lm
$ ./pi.out 10 1000000
   PI = 3.1412
```

Yêu cầu 3: Trong bài toán Ước lượng giá trị số PI tại Yêu cầu 2, cài đặt tính ái lực (affinity) để cho các tiểu trình có tid chẵn chạy trên nhân CPU_0, còn tiểu trình có tid lẻ chạy trên nhân CPU_1.

Hướng dẫn: Tải về tập tin ***LAB_IntroOS/LAB_6/task1_1affinity.c***, biên dịch và thực thi. Khi thực thi cần cấp quyền bằng lệnh sudo.

```
$ gcc –o pi.out task1_1affinity.c –lpthread -lm
$ sudo ./pi.out 10 1000000
   PI = 3.1412
```

- Dòng 24 là thư viện chứa lời gọi định thời.

- Dòng 63 khai báo tập chứa luồng tính CPU (Thread of CPU / Hệ điều hành nhìn thấy "CPU Threads" chứ không thấy "CPU Cores".

- Dòng 64 là xoá trắng tập CPU.

- Dòng 65 là thêm CPU #0 vào tập. Lưu ý trong System Monitor hay Task Manager thì CPU được đánh số từ 1, 2, …. còn trong lập trình thì đánh số từ 0.

- Dòng 65 là thêm CPU #1 vào tập.

- Dòng 67 là gác tập CPU cho tiểu trình tid. Từ đó tiểu trình này sẽ được định thời CPU chỉ trên tập chỉ định.

```
24  #include <sched.h>
63    cpu_set_t set;
64    CPU_ZERO( & set);
65    if ((tid % 2) == 0) CPU_SET(0, & set);
66    else CPU_SET(1, & set);
67    if (sched_setaffinity(tid, sizeof(set), & set) == -1)
```

Bài tập

Sinh viên nộp tập tin MSSV_PI_Baocao.docx sau khi hoàn tất các lần chạy khảo sát. Sinh viên có thể thử nghiệm trên nhiều máy tính, mỗi máy tính là một bảng báo cáo như sau:

CPU Clock: _____ GHz	**1 thread * 1 triệu điểm**		**2 thread * 1 triệu điểm**		**4 thread * 1 triệu điểm**	
	Giá trị PI	*Time*	*Giá trị PI*	*Time*	*Giá trị PI*	*Time*
1 luồng CPU						
2 luồng CPU						
4 luồng CPU						

Đọc thêm “Phương pháp Monte Carlo”

Một cách giá trị π khá thú vị là sử dụng kỹ thuật Monte Carlo, liên quan đến ngẫu nhiên. Kỹ thuật này hoạt động như sau: Giả sử bạn có một vòng tròn bán kính là 1 nội tiếp trong một hình vuông cạnh là 2, như thể hiện trong hình sau:

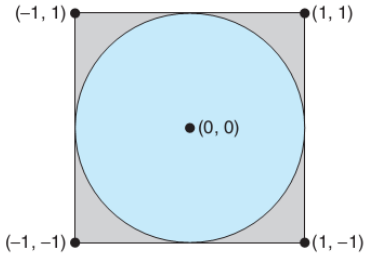
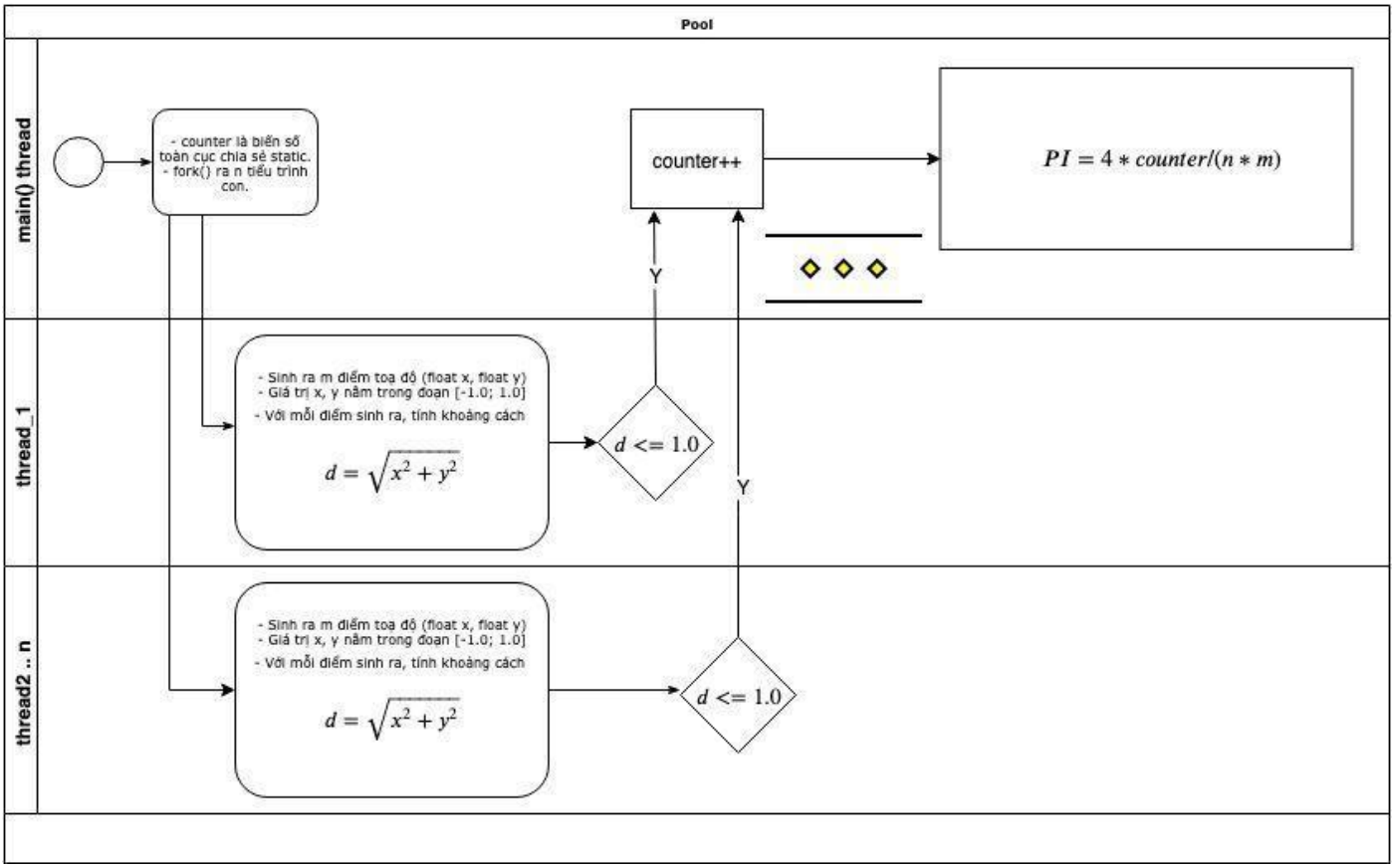


Figure 4.25 Monte Carlo technique for calculating π.

- Đầu tiên, tạo một chuỗi các điểm ngẫu nhiên dưới dạng tọa độ (x, y) đơn giản. Những điểm này phải nằm trong tọa độ Descartes bị ràng buộc hình vuông. Trong tổng số điểm ngẫu nhiên được tạo, một số sẽ nằm trong vòng tròn.
- Tiếp theo, ước tính π bằng cách thực hiện phép tính sau: π = 4 × (số điểm trong vòng tròn) / (tổng số điểm)

Chương trình cần tạo ra n tiểu trình và mỗi tiểu trình sẽ sinh ra m điểm, cũng chính tiểu trình sẽ tính khoảng cách d và cập nhật vào biến counter (là tổng số điểm nằm trong hình tròn, biến toàn cục chia sẻ). Xem lưu đồ kèm theo.


Pool
main() thread
- counter là biến số toàn cục chia sẻ static.
- fork() ra n tiểu trình con.
counter++
$PI = 4 * counter/(n * m)$
Y
thread_1
- Sinh ra m điểm toạ độ (float x, float y)
- Giá trị x, y nằm trong đoạn [-1.0; 1.0]
- Với mỗi điểm sinh ra, tính khoảng cách
$d = \sqrt{x^2 + y^2}$
$d <= 1.0$
Y
thread2 .. n
- Sinh ra m điểm toạ độ (float x, float y)
- Giá trị x, y nằm trong đoạn [-1.0; 1.0]
- Với mỗi điểm sinh ra, tính khoảng cách
$d = \sqrt{x^2 + y^2}$
$d <= 1.0$