FACULTY OF INFORMATION TECHNOLOGY

>>> COURSE MATERIAL <<<

INTRODUCTION TO OPERATING SYSTEM

Course ID 502047

Lab Part 5 – Programming Project

Mục tiêu	**Lý thuyết liên quan**	**Tài nguyên**
Đo thời gian	Ch4	Github
Lập lịch CPU	Ch5: CPU Scheduling	
Sắp xếp bằng Thread	Ch4: Threads and Concurrency.	
Ước lượng số PI	Ch4 và 5.	

Yêu cầu nộp bài: các tập tin mã nguồn .c và tập tin khả thực thi .out của các "ví dụ" và bài tập cuối hướng dẫn.

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, 5

[2] Greg Gagne , [2019], GitHub OS-BOOK OSC10e, Westminster College, United States

Access https://github.com/greggagne/osc10e in September 2019.

https://www.geeksforgeeks.org/merge-sort-using-multi-threading/

1. Lập lịch CPU

Bài tập này hiện thực một số thuật toán lập lịch tiến trình đã giới thiệu tại lớp lý thuyết. Bộ lập lịch sẽ được chỉ định một nhóm tác vụ được xác định trước và sẽ lập lịch cho các tác vụ dựa trên thuật toán lập lịch đã chọn. Mỗi tác vụ được gán một độ ưu tiên và khoảng thời gian cần dùng CPU. Các thuật toán lập lịch sau đây sẽ được thực hiện:

- Đến trước được phục vụ trước (FCFS), sắp xếp các tác vụ theo thứ tự mà chúng yêu cầu CPU.

- Tác vụ ngắn nhất trước tiên (SJF), sắp xếp các tác vụ theo thứ tự độ dài của các nhiệm vụ.

- Lập lịch ưu tiên (Priority), lên lịch các tác vụ dựa trên mức độ ưu tiên.

- Lập lịch xoay vòng (RR), trong đó mỗi tác vụ được chạy trong một lượng thời gian nhất định.

- Ưu tiên với xoay vòng, sắp xếp các tác vụ theo thứ tự ưu tiên và tiếp tục sử dụng lập lịch xoay vòng cho các tác vụ có mức độ ưu tiên như nhau.

Giả định trong bài tập này, mức độ ưu tiên nằm trong khoảng từ 1 đến 10, trong đó giá trị số cao hơn biểu thị mức độ ưu tiên tương đối cao hơn. Đối với lập lịch xoay vòng, độ dài của lượng thời gian là 10 mili-giây.

1.1 Hiện thực

Mã nguồn .c đã được đính kèm.

- Các tập tin hỗ trợ này đọc lịch trình của các tác vụ, chèn các tác vụ vào một danh sách và gọi trình lập lịch CPU.

- Lịch trình của các tác vụ có dạng [tên tác vụ] [độ ưu tiên] [thời điểm đến] [CPU burst], với định dạng ví dụ sau:

T1, 4, 0, 20

T2, 2, 0, 25

T3, 3, 0, 25

T4, 3, 0, 15

T5, 10, 0, 10

Trong đó, tác vụ T1 có mức độ ưu tiên 4 và chuỗi CPU là 20 mili giây đến lúc 0, v.v.

1.2 Chi tiết hiện thực

Tập tin driver.c đọc trong lịch trình của các tác vụ, chèn từng tác vụ vào danh sách được liên kết và gọi trình lập lịch xử lý bằng cách gọi hàm schedule(). Hàm schedule() thực thi từng tác vụ theo thuật toán lập lịch đã chỉ định và thực thi lệnh gọi hàm run() đã được định nghĩa trong tập tin CPU.c.

Hàm schedule() được hiện thực khác nhau trong các tập tin cần bổ sung bao gồm

schedule_fcfs.c

schedule_sjf.c

schedule_rr.c

schedule_priority.c

schedule_priority_rr.c

Một tập tin Makefile được sử dụng để xác định thuật toán lập lịch cụ thể sẽ được driver.c gọi. Ví dụ: để xây dựng bộ lập lịch FCFS, hãy nhập

```
>make fcfs
```

và thực thi bộ lập lịch (sử dụng lịch trình của các tác vụ schedule.txt) như sau:

```
>./fcfs schedule.txt
```

Tham khảo tệp README trong tải xuống mã nguồn để biết thêm chi tiết. Trước khi tiếp tục, hãy chắc chắn làm quen với mã nguồn được cung cấp cũng như tập tin Makefile.

Trong mã nguồn cung cấp, giải thuật FCFS đã chạy với dữ liệu giả định rằng Arrival Time = 0 và các tiến trình đến theo thứ tự từ trên xuống trong tập tin thông tin schedule.txt

```
trantin@SONY-UBUNTU:~/tttin/project5$ make fcfs
gcc -Wall -c schedule_fcfs.c
gcc -Wall -o fcfs driver.o schedule_fcfs.o list.o CPU.o
trantin@SONY-UBUNTU:~/tttin/project5$ ./fcfs schedule.txt
Running task = [T1] [0] [0] [20] from 0 to 20.
Running task = [T2] [0] [0] [25] from 20 to 45.
Running task = [T3] [0] [0] [25] from 45 to 70.
Running task = [T4] [0] [0] [15] from 70 to 85.
Running task = [T5] [0] [0] [20] from 85 to 105.
Running task = [T6] [0] [0] [10] from 105 to 115.
Running task = [T7] [0] [0] [30] from 115 to 145.
Running task = [T8] [0] [0] [25] from 145 to 170.
trantin@SONY-UBUNTU:~/tttin/project5$
```

Yêu cầu 1.1. Tính thời gian quay vòng trung bình, thời gian chờ trung bình cho thuật toán FCFS bằng cách chỉnh sửa bổ sung hàm schedule() trong tập tin schedule_FCFS.c

Yêu cầu 1.2. Giả sử thời gian đến của mỗi tiến trình khác nhau và khác 0. Hãy tiếp tục chỉnh sửa hàm schedule() trong tập tin schedule_FCFS.c. Gợi ý: Trong tập tin schedule_FCFS.c một mảng các Task đã được dùng để chứa thông tin tiến trình theo thứ tự "First come". Hãy sắp xếp lại mảng đó với tiêu chí "Arrival Time First", rồi tính toán và xuất ra theo thứ tự mảng.

Yêu cầu 1.3: hiện thực tập tin schedule_priority.c. Gợi ý: tương tự yêu cầu 2, hãy sắp mảng các Task lại theo tiêu chí "Độ ưu tiên", nếu cùng độ ưu tiên thì dùng tiêu chí phụ là FCFS.

Yêu cầu 1.4: hiện thực tập tin schedule_sjf.c. Gợi ý: tương tự yêu cầu 2, hãy sắp mảng các Task lại theo tiêu chí "Burst", nếu cùng độ ưu tiên thì dùng tiêu chí phụ là FCFS.

Yêu cầu 1.5: hiện thực tập tin schedule_rr.c. Gợi ý: Mảng các Task cần cập nhật lại Burst của mỗi tác vụ sau mỗi lần đáp ứng CPU, tác vụ nào có Burst giảm còn 0 thì xóa khỏi mảng Task.

Yêu cầu 1.6: hiện thực tập tin schedule_priority_rr.c. Gợi ý: Copy lại Yêu cầu 3 và bổ sung: nếu có nhiều tác vụ cùng độ ưu tiên thì sao chép chúng và một Mảng khác và chạy RR trên mảng đó.

2. Bài toán ước lượng giá trị số PI

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:

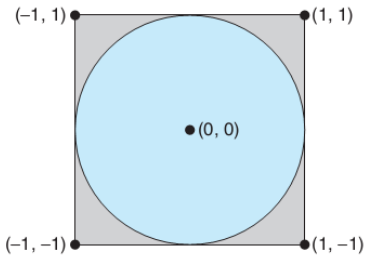


Figure 4.25 Monte Carlo technique for calculating π.

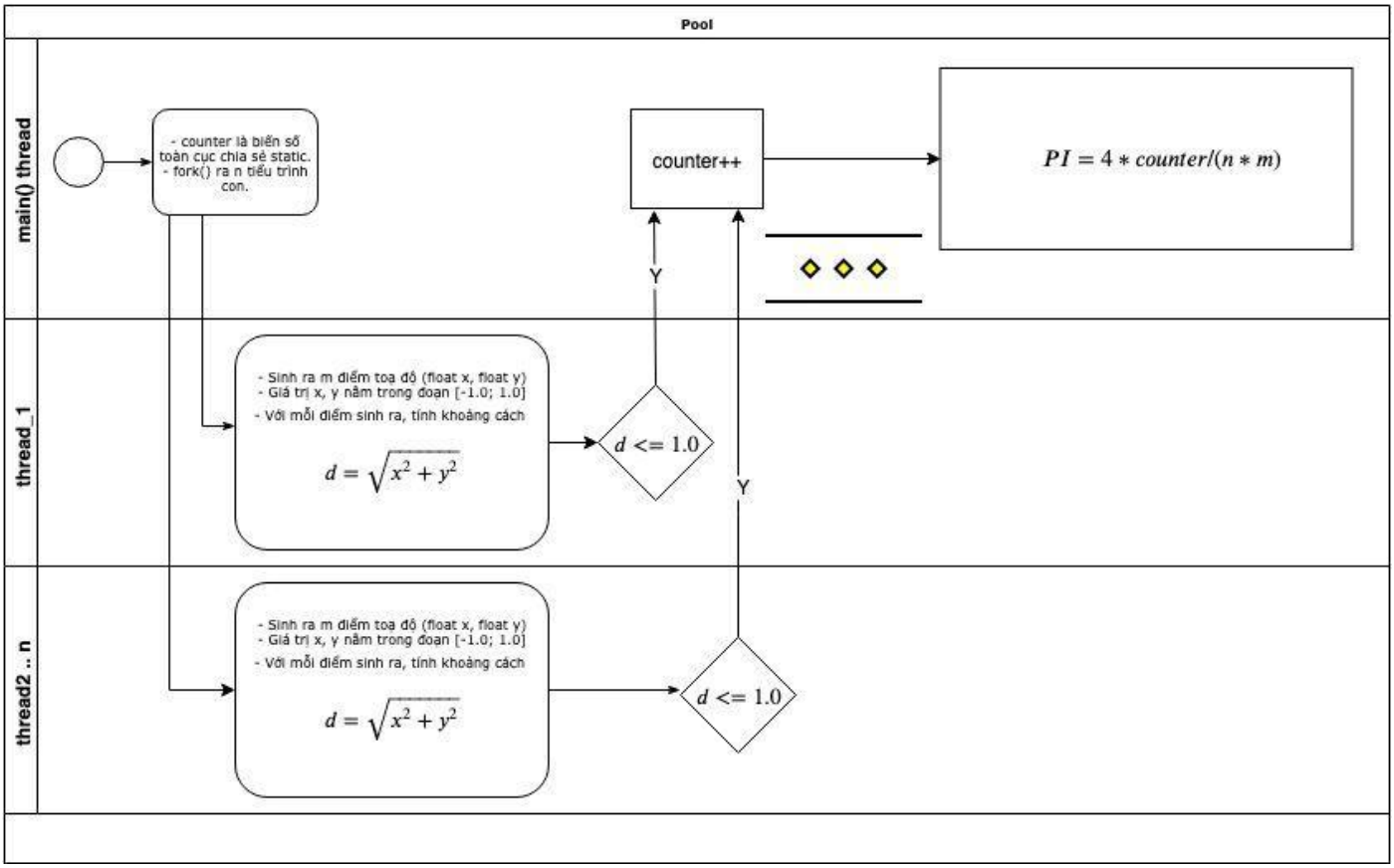
- Đầu tiên, tạo một chuỗi các điểm ngẫu nhiên dưới dạng tọa độ (x, y) đơn giản. Những điểm này phải nằm trong tọa độ Descartes bị ràng buộc hình vuông. Trong tổng số điểm ngẫu nhiên được tạo, một số sẽ nằm trong vòng tròn.
- Tiếp theo, ước tính π bằng cách thực hiện phép tính sau: π = 4 × (số điểm trong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

Mỗi tiểu trình cũng cần ghi giá trị các điểm sinh ra và một tập tin m_point.txt.

Lời gọi

```
>./pi.out 100 1000
PI = 3.1412
```


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
$d = \sqrt{x^2 + y^2}$
$d <= 1.0$

3. Ứng dụng sắp xếp bằng tiểu trình

Merge Sort is a popular sorting technique which divides an array or list into two halves and then start merging them when sufficient depth is reached. Time complexity of merge sort is O(nlogn).

Threads are lightweight processes and threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.
Multi-threading is way to improve parallelism by running the threads simultaneously in different cores of your processor. In this program, we'll use 4 threads but you may change it according to the number of cores your processor has.
Examples:

```
Input :  83, 86, 77, 15, 93, 35, 86, 92, 49, 21,
         62, 27, 90, 59, 63, 26, 40, 26, 72, 36
Output : 15, 21, 26, 26, 27, 35, 36, 40, 49, 59,
         62, 63, 72, 77, 83, 86, 86, 90, 92, 93


Input :  6, 5, 4, 3, 2, 1
Output : 1, 2, 3, 4, 5, 6
```

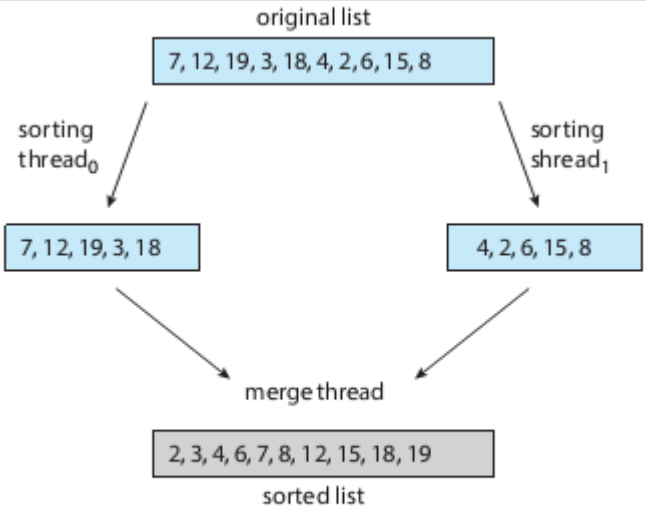


Figure 4.27 Multithreaded sorting.