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

Hướng dẫn Lab 6.5 – Giải thuật định thời RR

Một cách để đáp ứng tốt mọi tiến trình, đó là xoay vòng phục vụ. Hệ thống cần có năng lực chiếm quyền "preemptive" và quan sát toàn diện hàng chờ. RR cũng là một trong những lựa chọn hàng đầu của các hệ điều hành do tính đáp ứng là tiêu chí cần thiết trong phục vụ người dùng.

Mục tiêu	**Lý thuyết liên quan**	**Tài nguyên**
Lập lịch CPU	Ch5 - Programming Projects Scheduling Algorithms	https://github.com/Trantin84/LAB_IntroOS Sử dụng image Ubuntu 16 / 18 Project "CPU Scheduler"
Giải thuật RR		
Bài tập RR		

Yêu cầu sinh viên: Hiểu và thực thi các đoạn mã đã cung cấp. Áp dụng cho các bài toán liên lạc giữa các tiến trình.

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.

[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

Yêu cầu 1: Hiện thực giải thuật RR với giả định mọi tiến trình đến tại thời điểm 0. Cho thời gian lượng tử (hay lát cắt thời gian) t_quantum là hằng số cho trước.

Hướng dẫn: Để hiện thực giải thuật RR, chúng ta cần một hàng chờ có thông số "thời gian còn lại cần chạy CPU (gọi là t_remain)" với cài đặt ban đầu chính là giá trị burst của từng tiến trình, và sẽ giảm một lượng t_quantum mỗi khi tiến trình được gọi chạy. Lưu ý rằng nếu thời gian còn lại này nhỏ hơn t_quantum thì lần chạy cuối chỉ cần lượng này, và đồng hồ bộ định thời cũng dịch đi cùng giá trị (chứ không phải là t_quantum).

Hãy sao chép LAB 6.3 thành LAB 6.5 và chỉnh sửa/bổ sung các chi tiết sau:

Tạo mới tập tin RR.c

	FCFS.c	RR.c
9 12	`void * FCFS(void * param) {` `  //SORTING ARRIVAL TIME`	`void * RR(void * param) {` `  //DO NOT SORTING`

Bổ sung vào RR.c

Mảng chứa thời gian cần CPU còn lại.	`int t_remain(process);` `t_remain[i] = task[i].burst;`
Tổng thời gian cần CPU còn lại.	`int total_remain = sum all t_remain[]`
Tiến trình đầu tiên được phục vụ	`int next_P = 0;`
Giải thuật chạy khi còn tiến trình chưa hoàn tất.	`while(total_remain > 0)`
TH1: tiến trình xong rồi.	`(SKIP)`

TH2: tiến trình cần ít hơn quantum	`time += t_remain[i];`
	`total_remain -= t_remain[i];`
	`t_remain[i] = 0;`

TH3: tiến trình cần nhiều hơn quantum	`time += t_quantum;`
	`t_remain[i] -= quantum;`
	`total_remain -= quantum;`

Đi đến tiến trình tiếp theo	`next_P = (next_P +1) MOD process`

Chỉnh sửa tập tin driver.c

`pthread_create( & tid[0], NULL, FCFS, NULL);`

thành `pthread_create( & tid[0], NULL, RR, NULL);`

* SV có thể tạo thành một tiểu trình mới `pthread_create( & tid[1], NULL, RR, NULL);` để chạy cả 2 giải thuật, khi đó, cần sao chép mảng task cho từng tiểu trình, vì mảng task ban đầu đã bị thao tác thay đổi bởi mỗi tiểu trình được gọi. Cũng cần lưu ý khi chạy đồng thời nhiều giải thuật, các dòng in thông báo ra màn hình có thể bị xen kẽ.

Chỉnh sửa tập tin task.h

- Thêm các nguyên mẫu hàm vừa bổ sung nếu có.

- Thêm các biến số toàn cục nếu có.

```
$make clean
$make rr
$./rr data.txt
```

Yêu cầu 2: Hiện thực giải thuật RR với giả định mọi tiến trình đến tại các thời điểm khác nhau và khác 0.

Hướng dẫn: Giải thuật RR xoay vòng phục vụ các tiến trình "đã đến", tức là t_arrival <= time. Trong Yêu cầu 1, chúng ta cho rằng mọi tiến trình đã đến. Với Yêu cầu mới, có thể dùng 1 trong 2 cách:

1. Tạo một mảng flag[process] = {false} và ở mỗi đầu vòng lặp phục vụ, chúng ta kiểm tra tiến trình đến chưa bằng cách kiểm t_arrival <= time cho tất cả tiến trình và cập nhật mảng cờ này. Tiến trình đang đến lượt có cờ là false thì cũng bị bỏ qua (không làm gì cả, chỉ tăng chỉ số next_P như TH1 ở Yêu cầu 1.
2. Hiệu quả hơn cách 1, chúng ta có thể đặt điều kiện t_arrival <= time là điều kiện bổ sung (&&) để một tiến trình được phục vụ.

* Trong trường hợp CPU đang rảnh mà các tiến trình vẫn chưa đến, bộ định thời sẽ tăng dần thời gian time (time++) cho đến khi có tiến trình đến. Đoạn thời gian này gọi là Idle time.

Bài tập

Nộp các phiên bản ở các Yêu cầu nêu trên cùng các tập tin dữ liệu.

Phụ lục

Bài tập 5.4 Xem tập các tiến trình sau đây, với thời gian cần chạy ở cột Burst Time được cho ở đơn vị mili giây.

Process	Burst Time	Priority
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

Cho rằng các tiến trình đến theo thứ tự P1, P2, P3, P4, P5, tại thời điểm 0.0

Bài tập 5.5 Các tiến trình sau đây được lập lịch với giải thuật RR

Process	Priority	Burst Time	Arrival
P1	40	20	0
P2	30	25	25
P3	30	25	30
P4	35	15	60
P5	5	10	100
P6	10	10	105