BỘ THÔNG TIN VÀ TRUYỀN THÔNG

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 2

TÌM HIỂU VÀ HIỆN THỰC GIẢI THUẬT ROUND-ROBIN CÓ TÍNH ĐẾN THỜI GIAN CHUYỂN NGỮ CẢNH BẰNG JAVA

Giảng viên : **NGUYỄN NGỌC DUY**

Sinh viên thực hiện : **TRƯƠNG CHÍ TÀI**

MSSV : **N19DCAT067**

Lớp : **D19CQAT01-N**

Hồ Chí Minh, tháng 11 năm 2021

Mục lục

I. Giới thiệu:

- Lý do chọn đề tài: Hiện nay nơi đâu chúng ta đều có thấy được mọi người sử dụng đồ công nghệ, mà nhắc tới đồ công nghệ thì đối với những nhà nghiên cứu và làm việc về lĩnh vực này như chúng ta không thể không nhắc đến hệ điều hành, và điều độ CPU là một trong những công việc quan trọng nhất của hệ điều hành để ta có thể tối ưu hóa được hệ thống, nên em chọn giải thuật Round-Robin, một trong những giải thuật lâu đời được sử dụng trong các hệ điều hành truyền thống để nghiên cứu và nắm được các cơ bản để có thể tự nghiên cứu và phát triển các hệ điều hành hiện nay.
- Mục đích và nhiệm vụ: Tìm hiểu và hiện thực giải thuật Round-Robin có tính đến thời gian chuyển ngữ cảnh bằng ngôn ngữ Java.
- Giải thuật Round-Robin (RR) là một trong những giải thuật điều độ CPU để quyết định tiến trình nào trong hàng chờ thực thi sẽ được cấp CPU để thực thi. Bên cạnh giải thuật Round-Robin thì có các giải thuật điều độ CPU khác như First-Come First-Served (FCFS), Shortest-Job-First (SJF), Shortest-Remaining-Time-First (SRTF),...Và trong bài tiểu luận này chúng ta sẽ tìm hiểu về thuật toán điều độ quay vòng (Round-Robin).

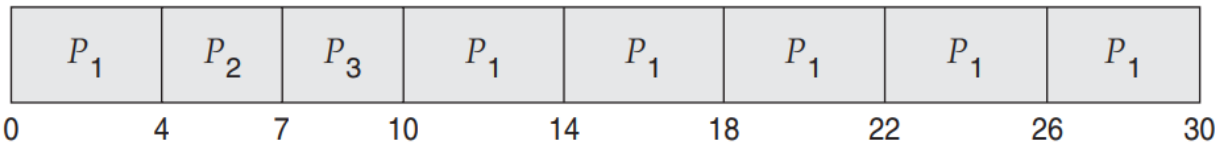
II. Điều độ quay vòng (Round-Robin):

1. Điều độ quay vòng là gì?

- Tên thuật toán điều độ quay vòng là dựa theo nguyên tắc quay vòng, nguyên tắc này có nghĩa là mỗi người sẽ được chia cho một điều gì đấy một cách công bằng trong mỗi vòng.
- Là thuật toán lâu đời nhất và đơn giản nhất thường được sử dụng cho hệ điều hành đa nhiệm và là phiển bản sửa đổi của FCFS, được thiết kế đặc biệt cho hệ thống chia sẻ thời gian. Ngược lại với FCFS có cơ chế không phân phối lại (non-pre-emptive) thì RR có cơ chế phân phối lại (pre-emptive) bằng cách sử dụng ngắt đồng hồ để hệ thống có thể chuyển đổi giữa các tiến trình. Hệ thống định nghĩa một khoảng thời gian nhỏ gọi là lượng tử thời gian (time quantum) hay lát cắt thời gian (time slice) có chiều dài khoảng từ vài mili giây tới vài trăm mili giây tùy vào cấu hình cụ thể. Tiến trình sẽ lần lượt được cấp CPU trong những khoảng thời gian như vậy trước khi bị ngắt và CPU được cấp cho tiến trình khác.
- Giống như FCFS, tiến trình sẵn sàng được xếp vào hàng đợi sao cho tiến trình đến sau được thêm vào cuối hàng. Khi CPU được giải phóng, hệ điều hành đặt thời gian của đồng hồ bằng độ dài lượng tử, lấy một tiến trình ở đầu hàng đợi và cấp CPU cho tiến trình. Sau khi được cấp CPU, tiến trình chuyển sang trạng thái chạy. Nếu tiến trình kết thúc chu kỳ sử dụng CPU trước khi hết thời gian lượng tử, tiến trình sẽ giải phóng CPU và trả lại quyền điều khiển cho hệ điều hành. Trong trường hợp ngược lại, khi hết độ dài lượng tử, đồng hồ sẽ sinh ngắt. Tiến trình đang thực hiện phải dừng lại và quyền điều khiển chuyển cho hàm xử lý ngắt của hệ điều hành. Hệ điều hành thực hiện việc chuyển đổi ngữ cảnh và chuyển tiến trình về cuối hàng đợi sau đó chọn một tiến trình ở đầu và lặp lại quá trình trên.
- Sau đây chúng ta hãy cùng xét một ví dụ để có nhìn một cách tổng quan nhất để rút ra các điểm mạnh và điểm yếu của giải thuật Round Robin:

Process	Burst Time
P_1	24
P_2	3
P_3	3

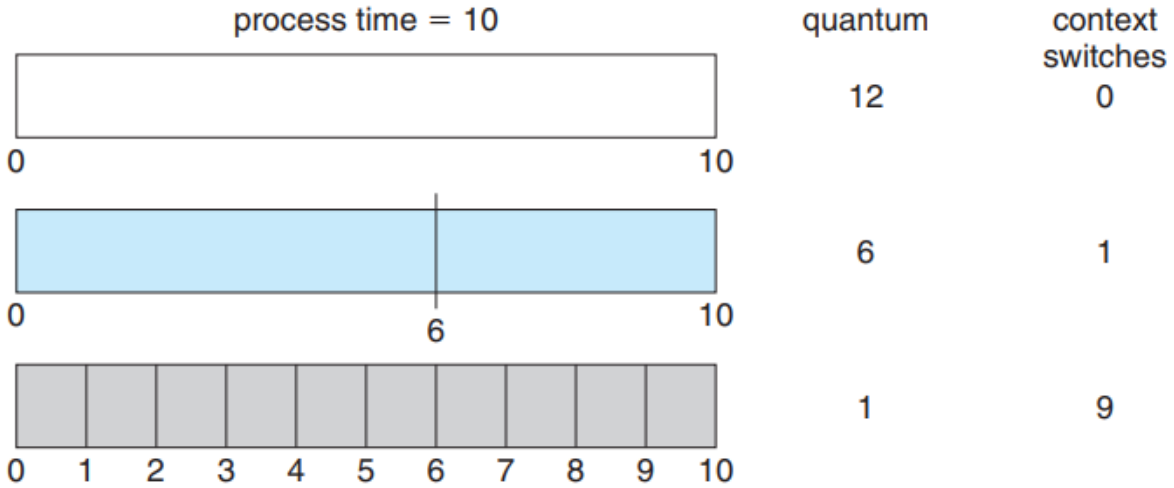
Xét với thời gian quantum q=4 milliseconds, thì P1 sẽ thực hiện được 4 ms và sẽ cần thêm 20 ms nữa để thực hiện xong nhiệm vụ nên P1 sẽ được phân phối lại, CPU được gửi lại cho P2 đang nằm trong hàng đợi. P2 chỉ cần 3 ms nên sẽ thực hiện xong trước khi thời gian quantum kết thúc và như đã đề cập ở trên thì CPU sẽ được giải phóng và trả lại quyền điều khiển cho OS. Tiếp tục CPU được gửi cho P3 để xử lí. Mỗi tiến trình đề được xử lí 1 lần rồi thì CPU sẽ trở về lại P1 và thực hiện tương tự. Và ta có kết quả như sau:



Chúng ta hãy tính toán thời gian chờ đợi trung bình cho tiến trình này. P1 chờ 6 ms (10 - 4), P2 chờ 4 ms và P3 chờ 7 ms.

Do đó, thời gian chờ trung bình là 17/3 = 5,66 ms.

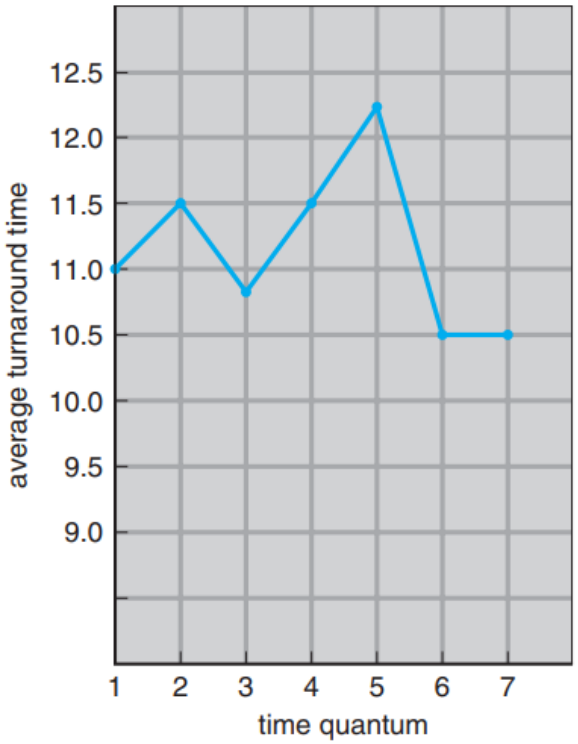
- Xét 10 ms đầu với thời các thời gian quantum khác nhau thì hãy xem sự khác biệt số lần chuyển ngữ cảnh của các trường hợp:



→ Với ví dụ ta vừa xét, ta thấy được số lần chuyển ngữ cảnh sẽ tỉ lệ nghịch với độ lớn thời gian quantum.

- Nếu lượng tử là 12 đơn vị thời gian, tiến trình kết thúc trong ít hơn 1 thời gian lượng tử. Tuy nhiên, nếu q là 6 đơn vị thời gian, quá trình đòi hỏi 2 lần cần CPU, dẫn đến 1 lần chuyển đổi ngữ cảnh. Nếu lượng tử thời gian là 1 đơn vị thời gian, thì dẫn đến 9 lần chuyển ngữ cảnh sẽ xảy ra, làm chậm việc thực hiện các tiến trình.

- Do đó, chúng ta muốn lượng tử thời gian lớn đối với thời gian bối cảnh. Nếu thời gian chuyển đổi ngữ cảnh là khoảng 10% của lượng tử thời gian, thì khoảng 10% thời gian CPU sẽ được sử dụng trong việc chuyển bối cảnh. Trong thực tế, hầu hết các hệ thống hiện đại đều có thời gian quantum khác nhau, từ 10 đến 100 ms. Thời gian cần thiết cho việc chuyển đổi ngữ cảnh thường ít hơn 10 ms. Do đó, thời gian chuyển đổi ngữ cảnh là một phần nhỏ của thời gian quantum.
- Thời gian quay vòng (turnaround time) cũng phụ thuộc vào độ lớn nhỏ của thời gian quantum. Nhìn vào hình sau:



process	time
P_1	6
P_2	3
P_3	1
P_4	7

- Ta thấy được thời gian quay vòng trung bình của một tập các tiến trình sẽ cải thiện nếu thời gian quantum tăng. Nói chung, thời gian quay vòng trung bình có thể được cải thiện nếu hầu hết các tiến trình có thể thực hiện xong trong 1 lần đầu tiên được cấp CPU. Ví dụ, ta có 3 tiến trình với mỗi tiến trình có burst time là 10 đơn vị thời gian và thời gian quantum q=1 đơn vị thời gian thì thời gian quay vòng trung bình sẽ là 29, nếu như thời gian quantum q=10 đơn vị thời gian thì thời gian quay vòng trung bình sẽ giảm xuống còn 20. Nếu thời gian chuyển ngữ cản được thêm vào thì thời gian quay vòng trung bình sẽ tăng thậm chí nhiều hơn đối với một thời gian quantum nhỏ hơn. Mặc dù ta cần thời gian quantum cần lớn để xử lí với các lần chuyển ngữ cảnh, nhưng nó không nên quá lớn, nếu không RR sẽ trở thành FCFS. Một quy tắc là 80% của các CPU burst nên ngắn hơn thời gian quantum.
- Từ ví dự trên ta hãy cùng đến với những đặc điểm, điểm mạnh và điểm yếu của giải thuật này.

2. Đặc điểm:

- Round-Robin là thuật toán có phân phối lại (pre-emptive).
- CPU sẽ chuyển đến tiến trình tiếp theo sau 1 khoảng thời gian cố định được gọi là time quantum hay time slice.
- Tiến trình được phân phối lại sẽ được thêm vào cuối hàng đợi.
- Round-Robin là kiểu lai được gọi là clock-driven (khác event-driven).
- Là một trong những thuật toán lâu đời nhất, công bằng nhất và dễ thực hiện nhất.
- Được sử dụng rộng rãi trong các hệ điều hành truyền thống.
- Thời gian đợi trung bình thường là dài.
- Không một tiến trình nào được cấp CPU hơn 1 time quantum trong một dòng (trừ trường hợp chỉ có tiến trình đấy có khả năng chạy).
- Nếu có n tiến trình và time quantum là q, thì mỗi tiến trình chờ không quá (n-1)×q đơn vị thời gian cho đến lần time quantum tiếp theo.
- Nếu thời gian lượng tử lớn quá lớn thì dẫn đến thì chính sách RR tương tự chính sách FCFS đối với tiến trình.
- Điều độ quay vòng cho phép cải thiện thời gian đáp ứng của tiến trình so với FCFS nhưng vẫn có thời gian chờ đợi trung bình tương đối dài.

3. Điểm mạnh:

- Không gặp vấn đề đói tài nguyên hay hiệu ứng đoàn xe.
- Tất cả các tiến trình đều được cấp phát CPU công bằng.
- Đối xử với các tiến trình không với bất kì ưu tiên nào.
- Phương pháp này không phụ thuộc vào burst time. Nên rất dễ thực hiện trên các hệ thống.
- Mang lại hiệu suất tốt nhất về thời gian phản hồi trung bình.
- Cho phép OS sử dụng phương pháp chuyển ngữ cảnh để lưu lại trạng thái của tiến trình được phân phối lại.
- Nếu biết tất cả các tiến trình đang ở trong hàng đợi chạy thì ta có thể tính được thời gian phản hồi tệ nhất của trường hợp các tiến trình này.

4. Điểm yếu:

- Nếu thời gian lượng tử của OS thấp thì hiệu năng CPU sẽ giảm vì tốn rất nhiều thời gian chuyển ngữ cảnh (Context switching).
- Phương pháp này tốn nhiều thời gian chuyển ngữ cảnh.
- Hiệu năng phụ thuộc rất nhiều vào thời gian lượng tử.
- Không thể thiết lập ưu tiên cho các tiến trình nên không thể dành ưu tiên cho các nhiệm vụ đặc biệt.
- Thời gian lượng tử càng thấp thì số lần chuyển ngữ cảnh càng cao hơn.
- Để tìm ra chính xác thời gian lượng tử là nhiệm vụ khó khăn trong hệ thống này.

5. Hiện thực giải thuật:

- Tạo 1 class Process gồm các thuộc tính arr (thời gian đến), bt (burst time), total_wt (tổng thời gian chờ.

- Khởi tại mảng Process arr[] và re_arr[] để copy từ arr để tính toán.
- Tạo biến q (quantum time) và cs (context-switch time).
- Khởi tạo một hàng đợi readyQueue để chứa các chỉ số của các tiến trình đang chờ được cấp CPU.
- Tạo một biến t (thời gian) và gán bằng arr[0].arr (vì các tiến trình nhập vào thời gian luôn lớn hơn hoặc bằng tiến trình trước).
- Tạo một hàm updateQueue(Queue<Integer> readyQueue,Process arr[],int n,int t) để update các tiến trình nào có arr bằng t và bt >0 thì sẽ được thêm vào hàng đợi chờ được cấp CPU.
- Tạo hàm roundRobin(Process arr[],int n,int q,int cs) để thực hiện xoay vòng các tiến trình. Trong hàm này, ta updateQueue và kiểm tra hàng đợi có rỗng không:
  + Nếu có ➔ kết thúc vòng lặp và hiển thị kế quả.
  + Nếu không: lấy phần tử đầu trong readyQueue gán bằng tam. Kiểm tra arr[tam].bt >q:
    - Đúng: thì cập nhật các thuộc tính của Process và update hàng đợi readyQueue trong thời gian tiến trình đó thực hiện.

```
re_arr[tam].bt-=q;
re_arr[tam].total_wt+=(t-re_arr[tam].arr);
re_arr[tam].arr=t+q;
int ti=t+1;
t=t+q+cs;
thuTu+="p"+tam+"->";
for (;ti<t;ti++)
{
   updateQueue(readyQueue,re_arr,n,ti);
}
```

    - Không đúng: cũng cập nhập thuộc tính của Process nhưng có nhưng phép toán khác, xem dưới đây để rõ hơn:

```
re_arr[tam].total_wt+=(t-re_arr[tam].arr);
re_arr[tam].arr=t+re_arr[tam].bt;
int ti=t+1;
t+=(re_arr[tam].bt+cs);
re_arr[tam].bt=0;
thuTu+="p"+tam+"->";
for (;ti<t;ti++)
{
   updateQueue(readyQueue,re_arr,n,ti);
}
```

- Sau khi readyQueue rỗng ta hiển thị kết quả ra màn hình gồm: thời gian đến, burst time, tổng thời gian chờ (total_wt), thời gian kết thúc (finish), thời gian quay vòng (turn around

time) của các tiến trình, trung bình thời gian chờ và trung bình thời gian quay vòng của tất cả tiến trình.

III. Kết luận:

- Tên thuật toán điều độ quay vòng là dựa theo nguyên tắc quay vòng.
- Giải thuật Round-Robin (RR) là một trong những giải thuật điều độ CPU.
- Là phiên bản sửa đổi của giải thuật FCFS và được thiết kế đặc biệt cho hệ điều hành đa nhiệm.
- Là giải thuật lâu đời, công bằng nhất, dễ nhất và được sử dụng rộng rãi trong các hệ điều hành truyền thống.
- Là giải thuật có phân phối lại (pre-emptive).
- Nếu ta biết được tổng số tiến trình đang trong hàng đợi chờ được CPU ta có thể tính được trường hợp thời gian phản hồi tệ nhất.
- Dành rất nhiều thời gian cho việc chuyển ngữ cảnh.
- Là kiểu lai được gọi là clock-driven.
- Nếu thời gian lượng tử lớn quá lớn thì dẫn đến thì chính sách RR tương tự chính sách FCFS đối với tiến trình.
- Điều độ quay vòng cho phép cải thiện thời gian đáp ứng của tiến trình so với FCFS nhưng vẫn có thời gian chờ đợi trung bình tương đối dài.
- Hiệu năng phụ thuộc rất nhiều vào thời gian lượng tử.

IV. Tài liệu tham khảo:

1. A. Silbeschatz, P.B. Galvin, G. Gagne. Operating system concepts. 9th edition. John Wiley & Sons. 2013.
2. Từ Minh Phương. Bài giảng hệ điều hành. Học viện Công nghệ Bưu chính Viễn thông 2013.
3. Thầy Nguyễn Ngọc Duy. Giáo trình hệ điều hành. Học viện Công nghệ Bưu chính Viễn thông cơ sở TP. Hồ Chí Minh.