**What is the advantage of the scheduling strategy used in this assignment in comparison with other scheduling algorithms you have learned?**

*(Việc thực hiện chiến lược định thời như trong Bài tập lớn lần này có những ưu điểm gì so với việc thực hiện những chiến lược định thời khác mà bạn đã học?)*

Trả lời:

Cho đến thời điểm hiện tại, chúng ta đã học được 06 giải thuật định thời chính khác nhau, bao gồm: First\_come\_First\_serve (FCFS), Shortest\_Job\_First (SJF), Round\_Robin (RR), Priority (Prio), Multilevel\_Queue (MLQ), Multilevel\_FeedBack\_Queue (MLFQ). Tuy nhiên, nhằm nâng cao chất lượng và cải thiện mức độ khách quan của việc so sánh giữa các giải thuật, chúng ta xét các phiên bản nâng cao của các giải thuật. Và sau khi xem xét và loại bỏ các giải thuật sơ khai, ta có 03 giải thuật tốt nhất hiện tại: Shortest\_Remaining\_Time\_First (SRTF), Round\_Robin (RR), Multilevel\_FeedBack\_Queue (MLFQ). Chúng ta sẽ tiến hành so sánh 03 giải thuật trên với giải thuật được sử dụng trong khuôn khổ Bài tập lớn.

Sơ đồ mô phỏng mối liên hệ giữa các giải thuật:

A close-up of a computer code

Description automatically generated

Trong dung lượng phần định thời của Bài tập lớn học kỳ 232, ta xem xét giải thuật multilevel\_queue (priority + roundrobin) kết hợp cơ chế **slot**, ta gọi giải thuật này là Multilevel\_Slot\_Queue (MLSQ).

|  |  |  |
| --- | --- | --- |
| **Chiến lược định thời** | **Ưu điểm** | **Nhược điểm** |
| Shortest\_ Remaining\_Job\_First (SRJF) | * *Thời gian đáp ứng được tối ưu hóa (optimal response time):* lần lượt thực thi các tiến trình có thời gian sử dụng CPU ngắn nhất đến dài nhất, quá trình này cũng làm giảm thời gian chờ trong hàng đợi ready (Average waiting time) xuống mức thấp nhất. * *Starvation được giảm thiểu cho các tiến trình với thời gian thực thi CPU ít:* các tiến trình có CPU burst ngắn không bị cản trở bởi các tiến trình có CPU burst dài. | * *Không thực tế:* vì hầu như không thể biết trước lượng thời gian sử dụng CPU trong tương lai một cách chính xác. * *Phương pháp giải quyết:* nhìn về quá khứ để ước lượng thời gian sử dụng CPU trong tương lai. * *Starvation vẫn có thể xảy ra với các tiến trình có thời gian thực thi CPU dài.* * *Tăng nguy cơ overhead:* do đặc tính preemptive, cần phải kiểm tra và thực hiện chuyển ngữ cảnh thường xuyên. |
| Round\_ Robin (RR) | * *Nâng cao tính tương tác của hệ thống:* mọi tiến trình đều có lượng thời gian sử dụng CPU như nhau trong một chu kỳ, nâng cao tính công bằng. Đáp ứng rõ tiêu chí của multi-tasking operating system. * *Dễ hiện thực:* kế thừa tính giản đơn của giải thuật FCFS, chỉ bổ sung công tác chuyển ngữ cảnh khi cần thiết. * *Không xuất hiện Starvation:* các tiến trình lần lượt sử dụng CPU trong 1 time quantum (time slice) cố định, sau đó nhường lại cho các tiến trình khác, do đó không có tiến trình nào bị “bỏ rơi”. Nếu biết trước được time quantum và số lượng tiến trình trong ready-queue, có thể xác định chính xác thời gian tối đa một tiến trình phải chờ để có thể tiếp tục sử dụng CPU. | * *Average waiting time tương đối lớn nếu số lượng tiến trình nhiều:* càng nhiều tiến trình trong ready-queue, thời gian tối đa để một tiến trình có thể tiếp tục sử dụng CPU càng lớn, thời gian chờ đợi trong ready-queue tỷ lệ thuận với số tiến trình hiện có. * *Tăng nguy cơ overhead:* do đặc tính preemptive, cần phải kiểm tra và thực hiện chuyển ngữ cảnh thường xuyên. * *Vấn đề cài đặt time quantum:* nếu time quantum không đủ bé, hiệu suất của RR tiệm cận với FCFS, nếu time quantum không đủ lớn, hiệu suất của RR giảm rõ vì thời gian chuyển ngữ cảnh lớn so với thời gian thực thi với CPU. |
| Multilevel\_Feedback\_Queue (MLFQ) | Là sự kết hợp giữa RR và Prio:   * *Cân bằng average waiting time và đặc tính interactive:* các level có độ ưu tiên khác nhau làm giảm thời gian chờ đợi giữa các tiến trình, các level có cùng độ ưu tiên có cơ hội sử dụng CPU như nhau làm tăng tính tương tác của hệ thống. * *Hạn chế Starvation:* các tiến trình có độ ưu tiên thấp áp dụng phương pháp aging (định tuổi) để tăng độ ưu tiên, trở thành level có độ ưu tiên cao hơn, từ đó tiến trình chắc chắn sẽ được thực thi. | * *Phức tạp hóa công tác hiện thực:* phát triển từ MLQ, thừa hưởng trọn vẹn những công đoạn khó nhằn vốn có, đồng thời phải cung cấp cơ chế switching giữa các level. * *Tăng nguy cơ overhead:* do cần phải theo dõi và điều chỉnh các thông số biến động của tiến trình |
| Multilevel\_Slot\_ Queue (MLSQ) | * *Các tiến trình có độ ưu tiên cao sẽ được ưu ái*: tuy MLFQ vẫn có sự ưu ái, nhưng so với MLSQ việc sử dụng cơ chế slot thể hiện rõ hơn về mức độ quan trọng của các tiến trình có độ ưu tiên cao. Nâng cao tính thực dụng của chương trình vì đảm bảo đủ khả năng đáp ứng được các nhu cầu quan trọng nhất. * *Cân bằng average waiting time và đặc tính interactive:* các level có độ ưu tiên khác nhau làm giảm thời gian chờ đợi giữa các tiến trình, các level có cùng độ ưu tiên có cơ hội sử dụng CPU như nhau làm tăng tính tương tác của hệ thống. | * *Nguy cơ Starvation tiềm tàng:* việc không hỗ trợ phương pháp aging để thay đổi độ ưu tiên các tiến trình làm giảm tính linh hoạt của giải thuật, đồng thời tăng nguy cơ “đói” cho các tiến trình vốn dĩ có độ ưu tiên thấp. Tuy nhiên với cơ chế slot, đã giảm thiểu được nguy cơ này. (phân tích rõ ở phần sau) * *Không tối ưu hóa hiệu suất:* việc sử dụng độ ưu tiên cố định cho tiến trình có thể dẫn đến lãng phí tài nguyên CPU vì làm tăng average waiting time. Ví dụ: một tiến trình ngắn bị đặt vào hàng đợi level có độ ưu tiên thấp khiến nó dành nhiều thời gian trong ready\_queue hơn là thực thi với CPU. |

**Phân tích cơ chế slot trong MLSQ:**

Việc sử dụng slot trong MLSQ có những tác dụng sau:

* *Đặt ra sự tách biệt về tầm quan trọng giữa các tiến trình có độ ưu tiên thấp và các tiến trình có độ ưu tiên cao:* Các tiến trình với độ ưu tiên thấp sẽ được cấp phát ít slot để thực thi hơn và ngược lại, các tiến trình với độ ưu tiên cao sẽ được cấp phát nhiều slot để thực thi hơn. Bởi vốn dĩ, trong công thức **slot = MAX\_PRIO – priority** đã thể hiện rõ mối tương quan tuyến tính ngịch giữa biến slot và priority, rằng: độ lớn priority càng lớn (độ ưu tiên thấp) thì slot càng ít. Một số sự so sánh nhỏ như sau với chủ thể là MLSQ:
  + So với SRTF: Trong SRTF, thời gian còn lại của mỗi tiến trình được sử dụng để quyết định tiến trình nào sẽ được chạy tiếp theo. Trong khi đó, MLSQ sử dụng slot cố định, không phụ thuộc vào thời gian còn lại mà chỉ dựa trên mức ưu tiên.
  + So với RR: RR chia thời gian CPU thành các time slice đồng đều cho mỗi tiến trình. Trái lại, MLSQ sử dụng slot cố định dựa trên mức ưu tiên, cho phép các tiến trình ưu tiên cao hơn chạy nhiều hơn trong một đơn vị thời gian.
* *Giảm thiểu nguy cơ Starvation:* mặc dù trong bảng so sánh việc sử dụng MLSQ sẽ gây ra Starvation bởi tính không linh hoạt về độ ưu tiên giữa các tiến trình, thế nhưng cơ chế slot đã cung cấp một giải pháp giảm thiểu nguy cơ đó. Bằng cách cấp phát một lượng slot sử dụng CPU cố định cho mỗi level khác nhau, khi đó nếu một tiến trình có độ ưu tiên cao mà slot đã hết thì nó buộc phải nhường CPU lại cho các tiến trình ở level thấp hơn.
* *Giảm độ phức tạp trong công tác hiện thực*: khác với MLFQ cần phải cung cấp cơ chế switching level giữa các tiến trình, MLSQ không thực hiện điều đó, mà chỉ cần cung cấp thêm duy nhất một thuộc tính trong mỗi hàng đợi là **slot**. Việc sử dụng slot được ví như phần bù cho tính không linh hoạt của giải thuật, chính slot sẽ tạo ra sự linh hoạt đó nhưng theo cách riêng của mình.

Như vậy, MLFQ buộc tiến trình thực hiện nội biến đổi để giành được quyền sử dụng CPU, trong khi MLSQ chủ động thực hiện ngoại biến đổi mang quyền sử dụng CPU cho tiến trình. MLSQ không chỉ kế thừa được những lợi điểm của MLFQ (tức kế thừa được mặt tốt của cả RR và Prio), mà còn có một sử cải biến hết sức mới lạ, và sự mới lạ này đã mang lại một góc nhìn khác của công việc định thời, đó là: các tiến trình không cần phải tự thay đổi bản thân để có thể giành lấy quyền thực thi với CPU, mà chỉ cần thực hiện một thay đổi nhỏ với thuộc tính của mình và quyền thực thi CPU tự khắc sẽ đến.