3: Thread, Scheduler & Kernel

Phan Minh Thong

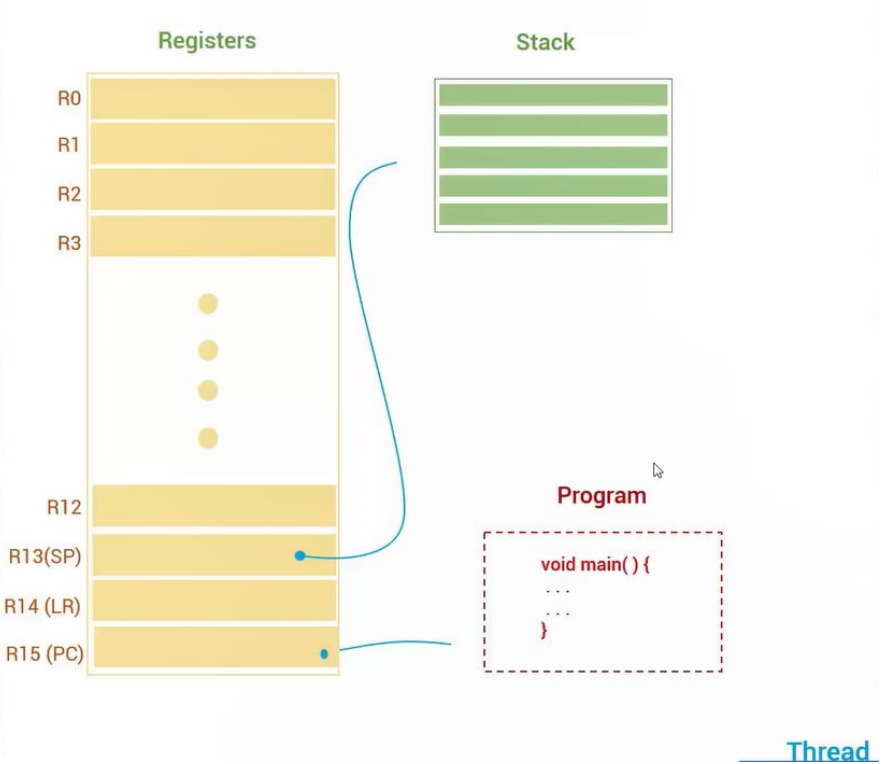
Thread

Về cơ bản, thread là quá trình thực thi của 1 Task.

Một thread gồm:

- 1 bộ thanh ghi.
- Bộ nhớ Stack.
- Chương trình để thực thi.

Khi một Thread chạy, nó có quyền sử dụng toàn bộ tài nguyên của vi điều khiển.

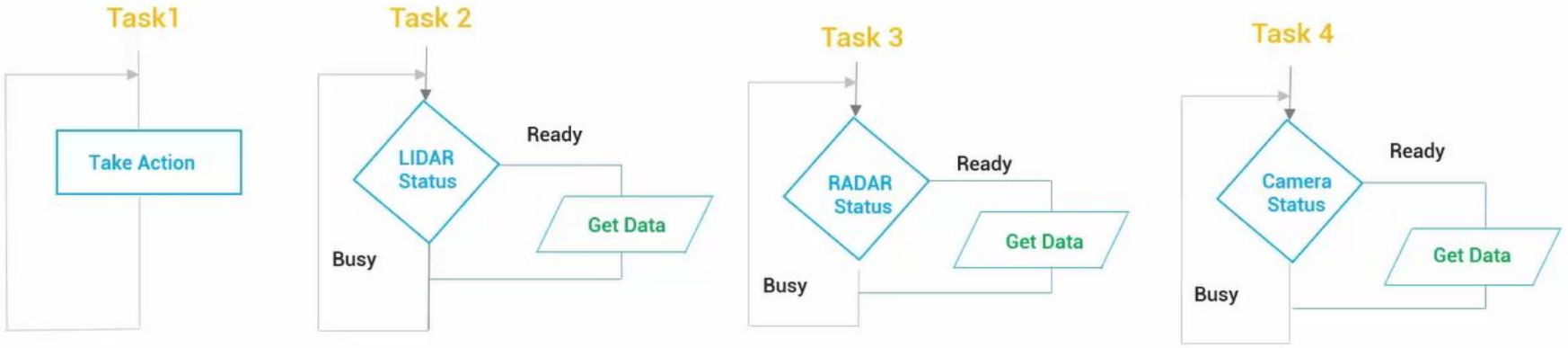


Thread

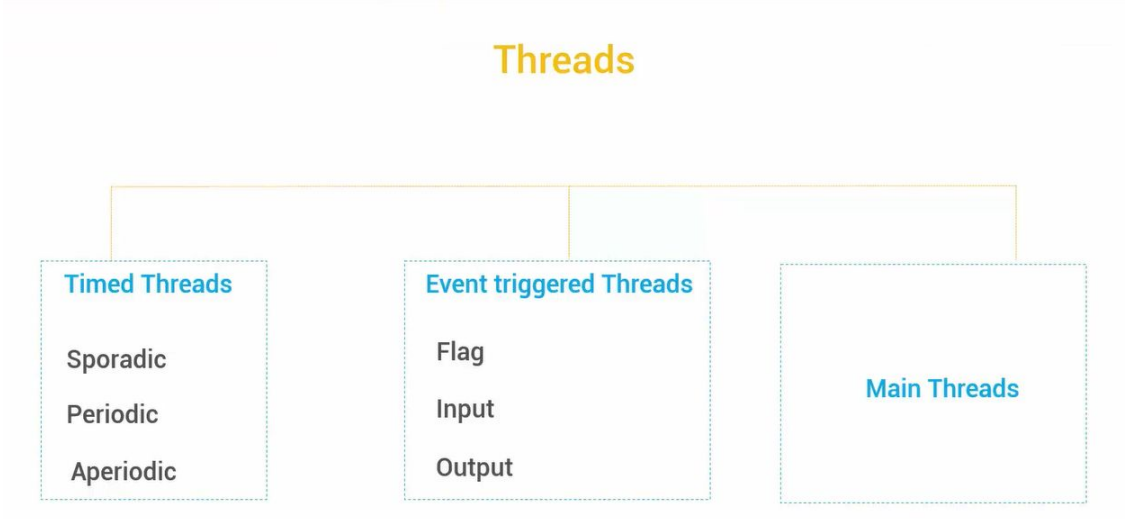
Với ví dụ ở bài trước:

- Quá trình thực thi Task Take Action là Thread Take Action.
- Quá trình thực thi Task LIDAR là Thread LIDAR.
- Tương tự với các task còn lại.

Như vậy, một chương trình có thể có nhiều thread.



Phân loại Thread



● Các tác vụ chạy khi có sự cố. ● Các tác vụ chạy định kì. ● Các tác vụ không đoán trước được.	● Các tác vụ được khởi chạy khi có 1 cờ nào đó được set hay reset. ● Các thread chạy khi có input/output.	● Các tác vụ chạy khi hệ thống rảnh.

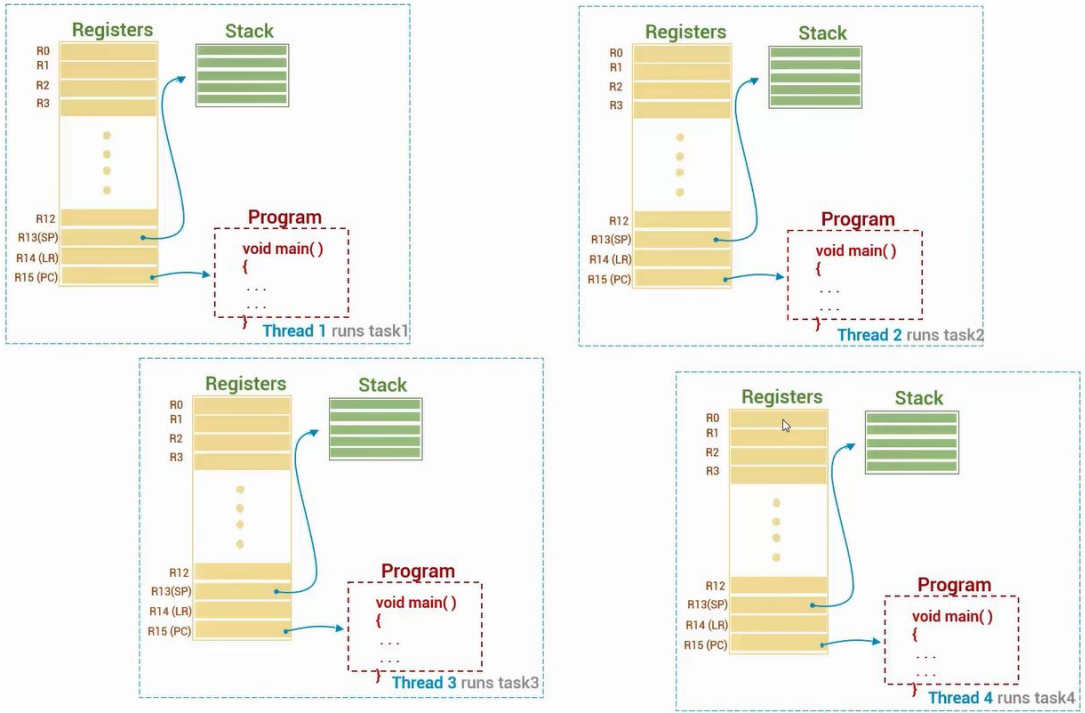
Multi Thread in RTOS

Mỗi Thread sẽ cần 1 bộ thanh ghi, Stack và Chương trình riêng để hoạt động.

→Nếu có đủ các thành phần trên cho mỗi Thread, tức là vi điều khiển có đủ 4 core với 4 Register Bank, các Thread có thể chạy song song và độc lập.

⇒STM32 chỉ có 1 core với 1 Register Bank duy nhất.

⇒Phải liên tục chuyển đổi việc sử dụng thanh ghi giữa các Task.

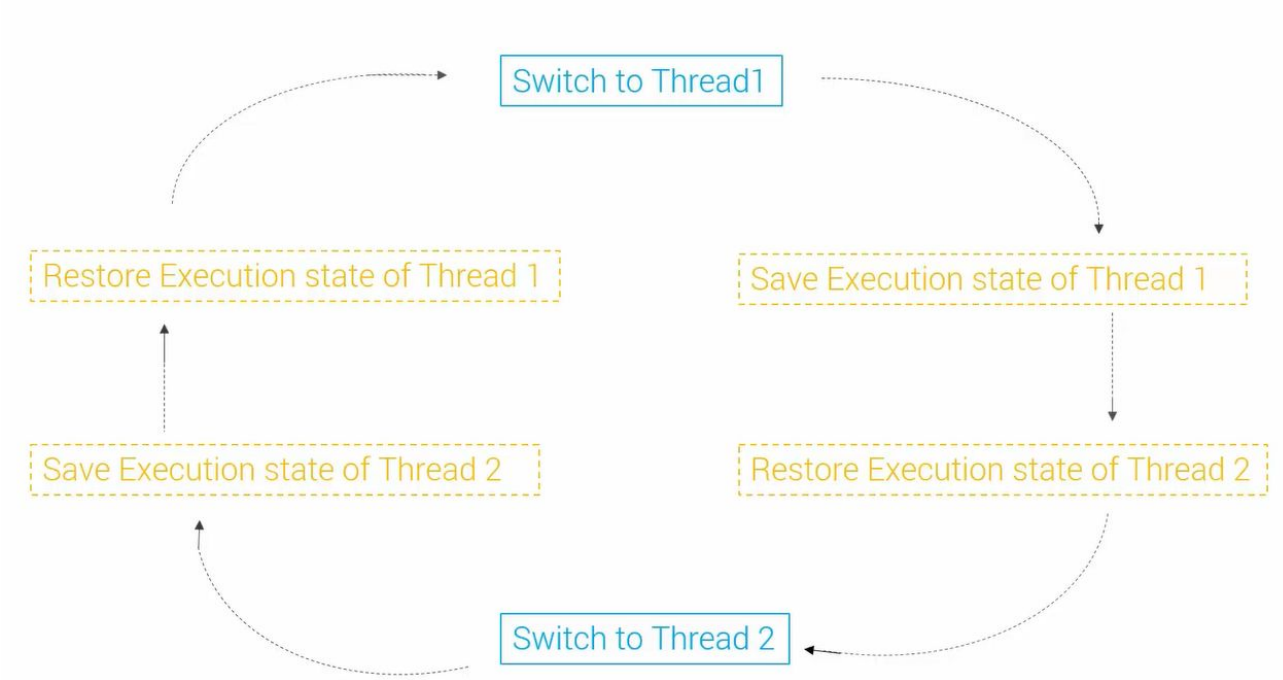


Thread Switching Process

Khi một Thread chạy, có có quyền sử dụng toàn bộ thanh ghi để tính toán.

Việc chuyển đổi thanh ghi cho các Thread là quá trình Thread Switch.

Mô tả theo sơ đồ:

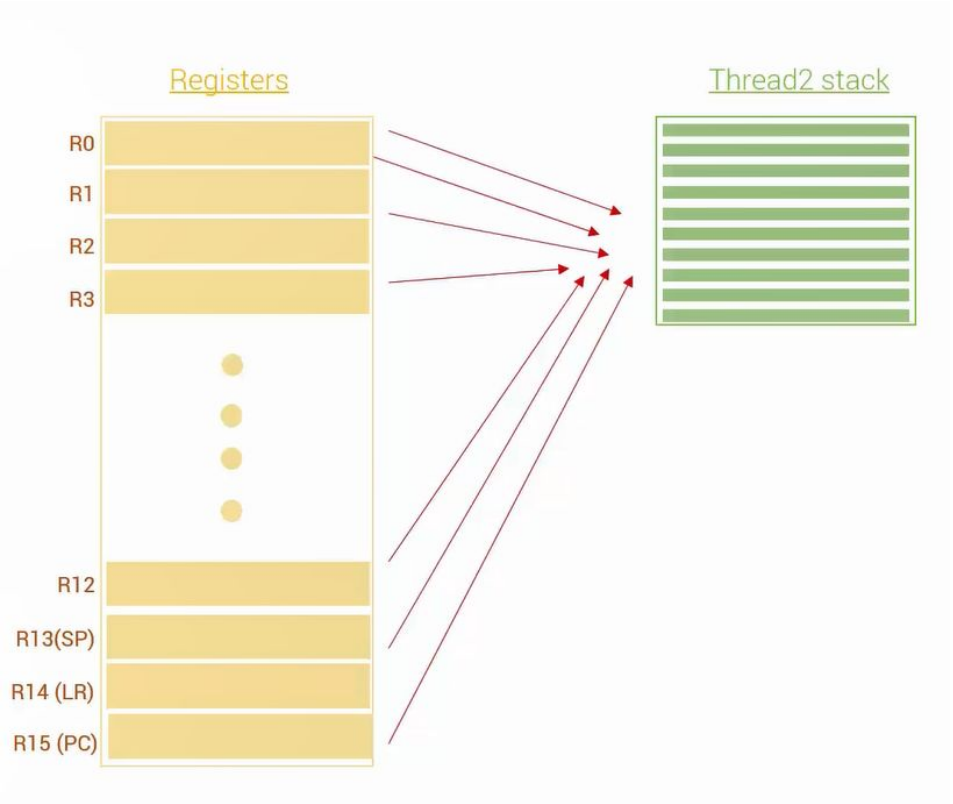


Save Execution State of a Thread

Save Execution State là việc lưu lại trạng thái của 1 Thread.

1 vùng nhớ Stack sẽ được tạo trong RAM, và dùng riêng cho mỗi thread.

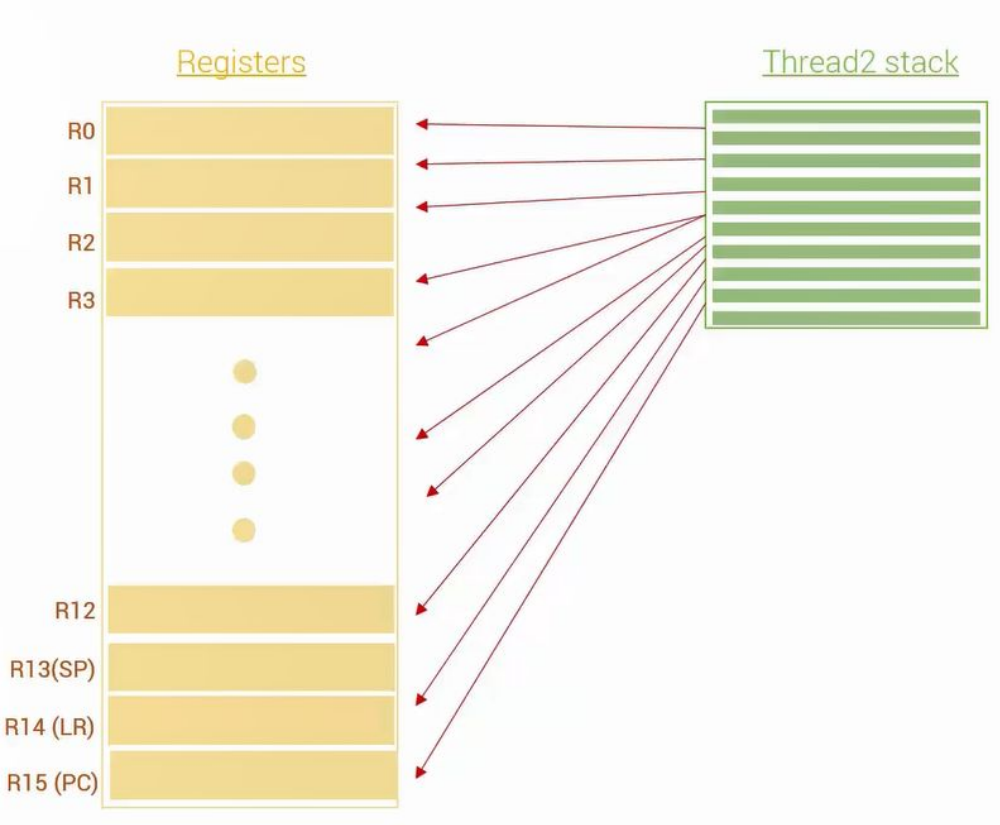
Việc lưu lại trạng thái chính là lưu giá trị các thanh ghi trong core vào Stack.



Restore Execution State of a Thread

Restore Execution State là việc khôi phục lại trạng thái của 1 Thread.

Việc khôi phục lại trạng thái chính là lấy các giá trị đã lưu trong RAM khi save, lưu các giá trị này vào các thanh ghi tương ứng trong core.



Thread Control Block (TCB)

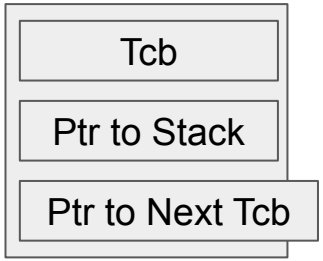
Để lưu được thông tin 1 Thread thì cần có 1 cấu trúc dữ liệu.

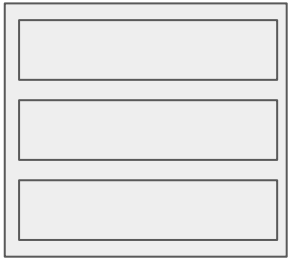
TCB là 1 Struct chứa thông tin của mỗi Thread. Gồm 2 thông tin chính:

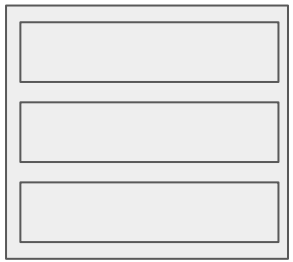
- Pointer to Stack.
- Pointer to next Thread.

Ngoài ra còn có các biến thông tin Thread:

- Status
- ID
- Priority
- …







Thread Scheduler

Thread Scheduler đảm nhiệm việc dừng 1 thread đang chạy và chọn 1 thread khác cho CPU thực hiện dựa trên các thuật toán.

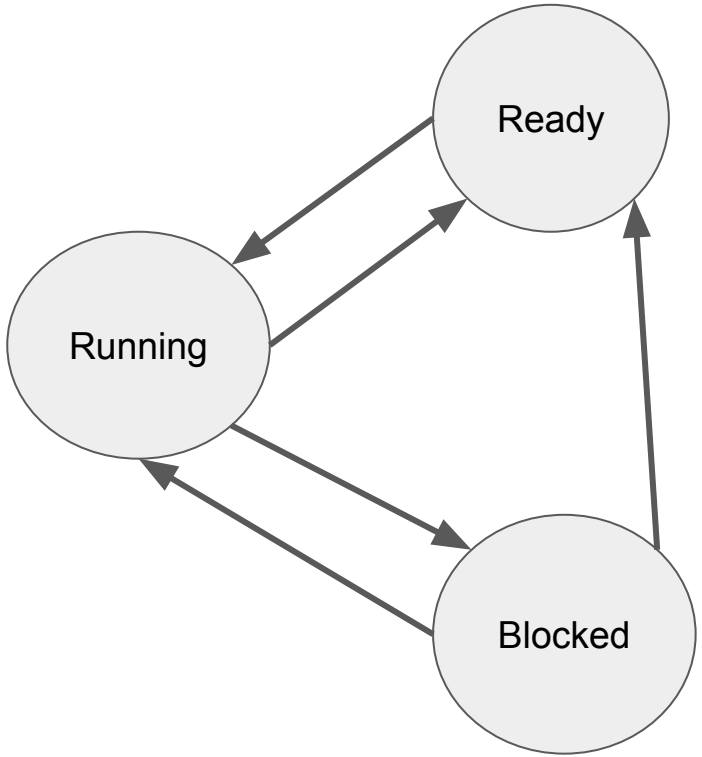
⇒Là 1 hàm có chức năng chạy các thread theo thứ tự lần lượt từng thread một.

Thread thường có 3 trạng thái chuyển đổi lẫn nhau:

- Running: Thread đang được thực thi bởi CPU.
- Ready: Thread sẵn sằng được thực thi bởi CPU.
- Blocked: Thread đang đợi 1 sự kiện nào đó để được kích hoạt.

Tại 1 thời điểm chỉ có 1 Thread ở trạng thái Running, các Thread còn lại sẽ chờ ở Ready và Blocked.

- **Ready→Running** khi Thread đang Running hoàn tất hoặc hết thời gian thực thi, hoặc thread Ready có độ ưu tiên cao hơn.
- Tương ứng, **Running→Ready** khi thread Ready có ưu tiên cao hơn, hoặc thread Running hết thời gian thực thi.
- **Running→Blocked** khi đang thực thi đến điểm cần 1 sự kiện nào đó xảy ra.
- **Blocked→Running** khi sự kiện kích hoạt xảy ra, và thread có độ ưu tiên cao hơn cả Ready và Running.
- **Blocked→Ready** khi sự kiện kích hoạt xảy ra nhưng độ ưu tiên thấp hơn hoặc ngang với Ready hay Running.



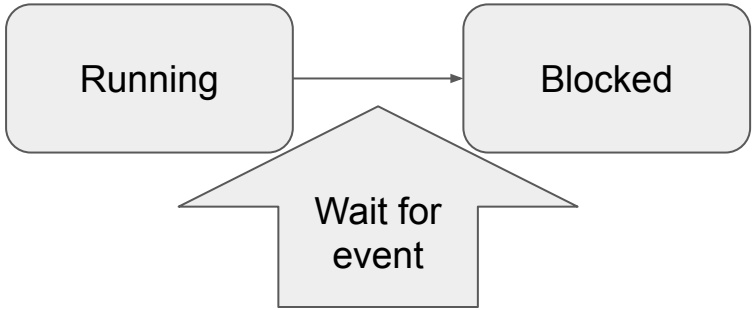
Preemption

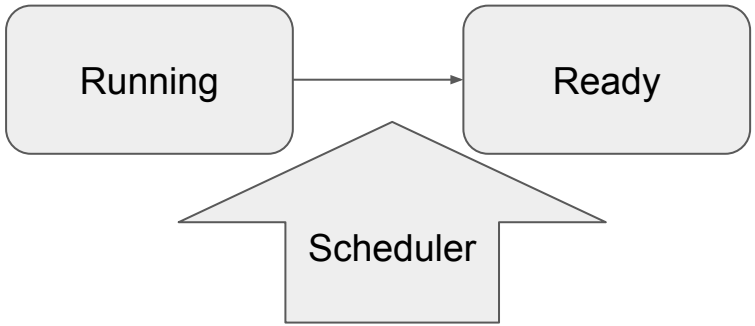
Khi 1 thread chuyển từ Running sang Blocked, nó tự hoãn chính nó vì phải đợi 1 event.

Khi 1 thread chuyển từ Running sang Ready, tức là nó bị Scheduler bắt buộc nhường quyền thực thi cho thread khác, bị Preempt bởi Scheduler. Nhằm mục đích:

- Đảm bảo tính công bằng các thread.
- Giúp thread hoàn thành deadlines.

Cần 1 ngắt để tạo Preemption.





Mỗi Scheduler sẽ có thuật toán riêng để quyết định trạng thái của các Thread. Có thể được phân loại gồm:

- **Static & Dynamic:**
  - Static: Độ ưu tiên của thread được xác định trước khi hệ thống chạy.
  - Dynamic: Độ ưu tiên của thread được xác định trong quá trình chạy.
- **Preemptive & Non-Preemptive:**
  - Preemptive: Thread có thể bị hoãn do 1 thread khác(có độ ưu tiên cao hơn) hoặc do ngắt.
  - Non-Preemptive: Khi 1 thread chạy, nó có quyền hoàn tất thực thi trước khi CPU thực thi thread khác, trừ khi nó chủ động nhường CPU.

Scheduler có thể kết hợp giữa các loại trên.

Scheduler Criteria

Một Scheduler được đánh giá dựa trên các tiêu chí:

- Throughput: Số lượng task thực hiện được trong 1 đơn vị thời gian.
- Turnaround time: Thời gian để hoàn thành mỗi task(Từ lúc task xuất hiện đến lúc hoàn thành).
- Response time: Thời gian từ khi yêu cầu đến khi nhận được phản hồi đầu tiên.
- Wait Time: Thời gian mỗi task phải chờ trong trạng thái Ready.
- CPU Utilization: % chu kỳ CPU khả dụng đang được dùng.

Một Scheduler tốt sẽ cần:

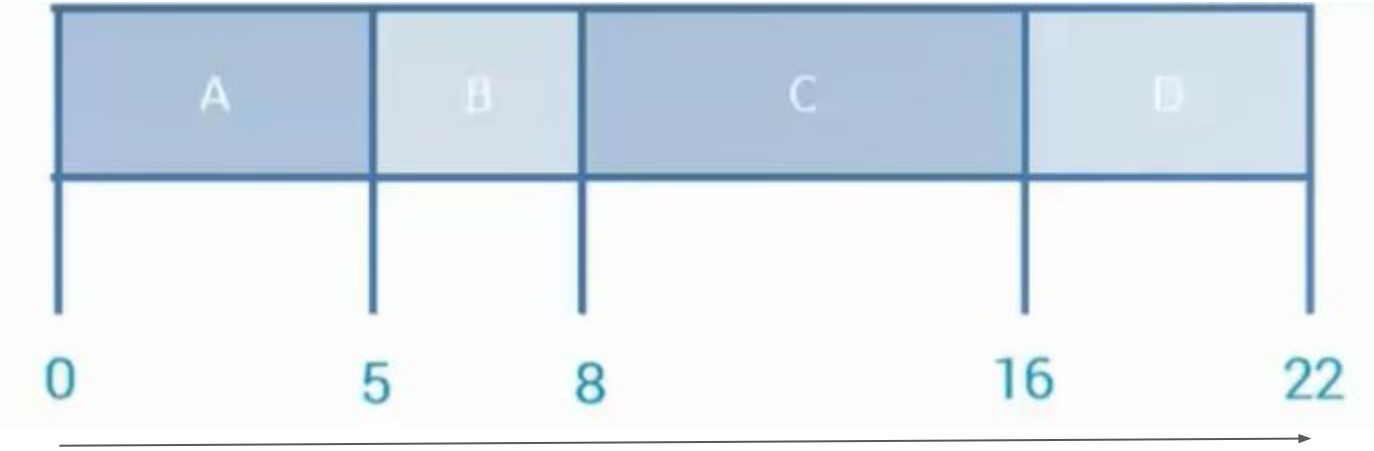
- Tối đa hóa Throughput.
- Giảm Turnaround time.
- Giảm Response time.
- Tối ưu hóa CPU Utilization.

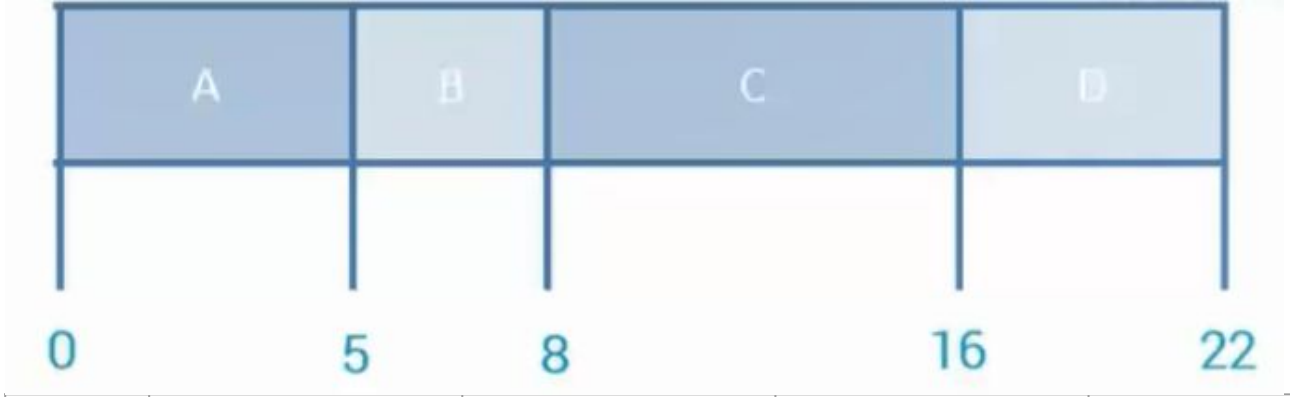
Các thuật toán Scheduler

1/First Come First Serve (FCFS)

-Non Preemptive.
-Task nào đến trước sẽ được thực thi trước, lần lượt theo kiểu FIFO queue.

Thread	A	B	C	D
Execution Time	5	3	8	6
Arrival Time	0	1	2	3





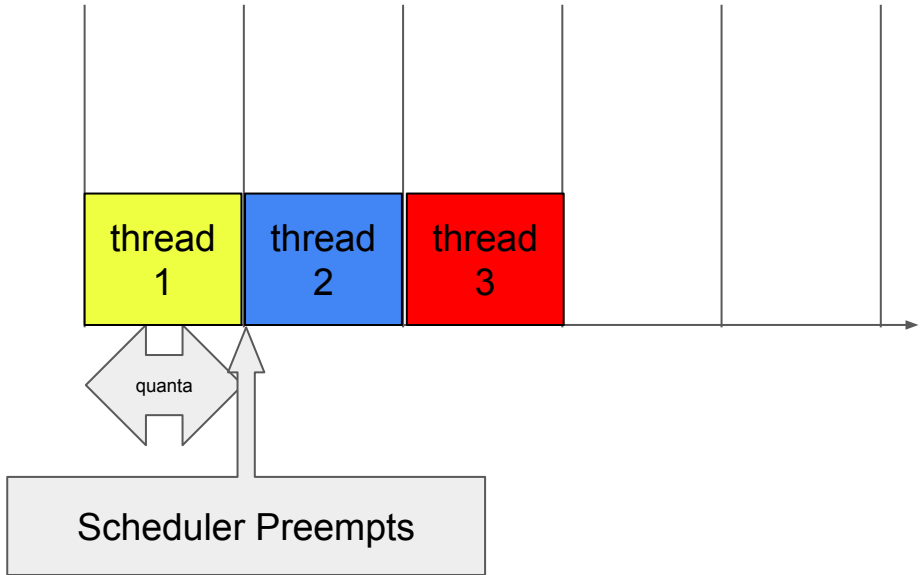
Thread	Arrival time ***(At)***	Execution time ***(t)***	Service time ***(St)***	Wait time ***(Wt = St - At)***	Turnaround time ***(t + Wt)***
A	0	5	0	0	5
B	1	3	5	4	7
C	2	8	8	6	14
D	3	6	16	13	19

- Trung bình Wait time: (0+4+16+13)/4 = 5.75 ms / Thread;
- Trung bình Throughput: 4/22 = 0.18 Threads/ms.

Các thuật toán Scheduler

1/Round Robin (RR)

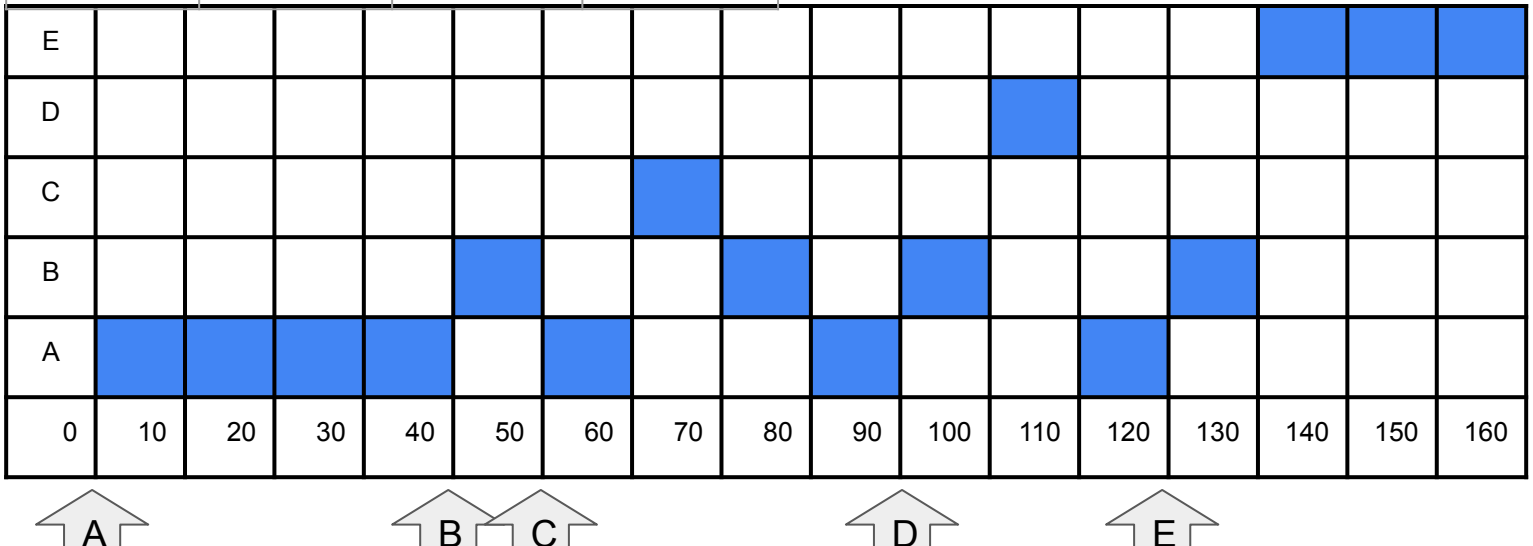
-Preemptive.
-Sử dụng việc chia sẻ thời gian. Mỗi thread sẽ có 1 thời gian thực hiện nhất định- timeslice (quanta).
-Khi 1 thread thực hiện hết 1 khoảng timeslice, Scheduler sẽ chuyển CPU cho thread khác.



Thread	Arrival time	Execution time	Remaining time
A	0	70	0
B	40	40	0
C	50	10	0
D	90	10	0
E	120	30	0

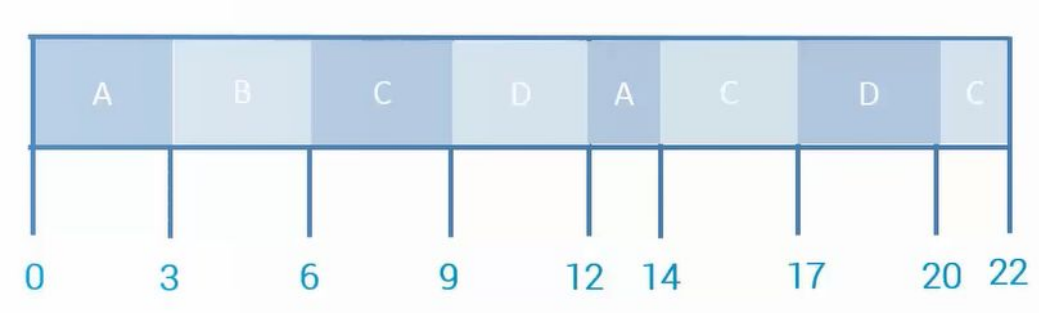
Thread	A	B	C	D	E
Execution Time	70	70	10	10	30
Arrival Time	0	40	50	90	120

Ready Queue



*Quanta = 10ms

Thread	A	B	C	D
Execution Time	5	3	8	6
Arrival Time	0	1	2	3



*Quanta = 3

Thread	Arrival time ***(At)***	Execution time ***(t)***	Service time ***(St)***	Wait time ***(Wt= St - At)***	Turnaround time ***(t + Wt)***
A	0	5	0,12	(0-0)+(12-3) = 9	14
B	1	3	3	3-1 = 2	5
C	2	8	6,14,20	(6-2)+(14-9)+(20-17) =12	20
D	3	6	9,17	(9-3)+(17-12) = 11	17

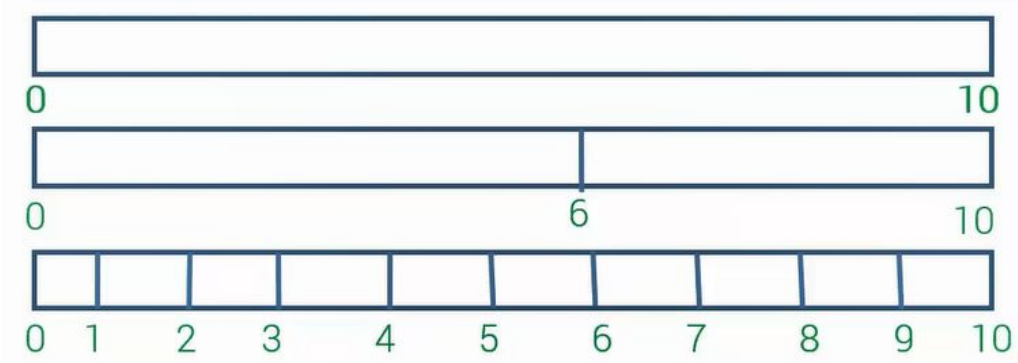
- Trung bình Wait time: (9+2+12+11)/4 = 8.5 ms /Thread;
- Trung bình Throughput: 4/22 = 0.18 Threads/ms.

Lưu ý khi chọn Quanta

- Quanta quá lớn so với thời gian thực thi của Thread, RR=FCFS.
- Quanta quá nhỏ, RR hoạt động như processor. Tuy nhiên phải thực hiện context switching liên tục ⇒Tốn nhiều tài nguyên xử lý.

➨Time quanta không được chọn quá lớn hoặc quá nhỏ so với thời gian thực thi thread trung bình. Nên chọn time quanta vừa phải và lớn hơn so với thời gian context switching.

Ví dụ với 1 Thread có Execution time =10;



Quanta	No.Context Switch
12	0
6	1
1	9