**<https://github.com/Showndarya/Operating-System-Algorithms?fbclid=IwAR0-JFCgAoAHYvKcHhn2TdXpFk0mEFbwTobaTmUcQOGZ7MGeib9rpdlTDyY>**

**Mục tiêu:**

+So Sánh hiệu quả thuật toán lập lịch FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK

+Phân tích bài toán Producer Consumer Problem và Dining Philosopher's problem

**Chức năng chính:**

Nhập vị trí bắt đầu của head, số lượng vị trí mà head phải đọc,Vị trí mà head phải đi qua, Vị trí trước của head trước khi thực hiện(trong 1 số thuật toán), Vị trí head quay lại tìm (trail track location trong 1 số thuật toán) và chương trình sẽ đưa ra tổng thời gian tìm kiếm

**Giải thích thuật toán:**

-FCFS(First Come First Server): head bắt đầu đi từ yêu cầu đầu tiên đến yêu cầu cuối cùng theo thứ tự thời gian

-SSTF(Shortest Seek time first): head tìm 2 yêu cầu cạnh mình, so sánh xem cái nào ngắn nhất và đi đến đó

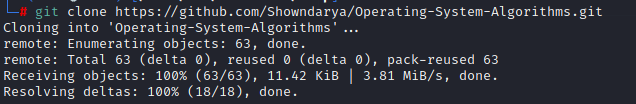
-SCAN: Từ điểm bắt đầu head đi đến cuối và từ điểm bắt đầu nó đi đến 0 qua tất cả request

-C-SCAN: Giống Scan nhưng đi theo vòng tròn đến cuối rồi quay lại từ đầu

-LOOK: Head đi từ giá trị lớn hơn start value rồi đi đến giá trị lớn nhất sau đó từ giá trị nhỏ hơn value đi đến giá trị nhỏ nhất

-C-LOOK: Giống LOOK nhưng đi theo vòng tròn, đến cuối rồi quay về đầu

**Cài đặt và chạy:**  
 Cài đặt trên môi trường linux(không có lưu ý j đặc biệt):



**Xử lí dữ liệu:**

Đầu vào :

+vị trí bắt đầu của head để đọc

+Số lượng vị trí của head để đọc

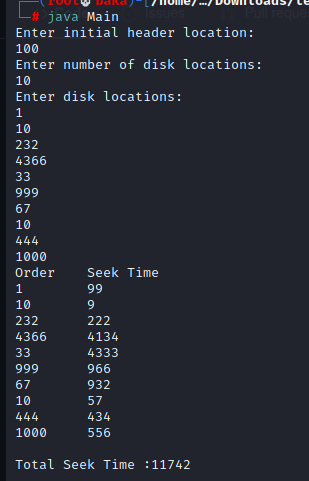
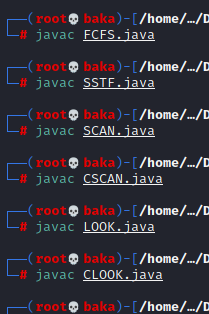
+Vị trí trước khi bắt đầu của head

+Vị trí để head quay về bắt đầu lượt quét mới

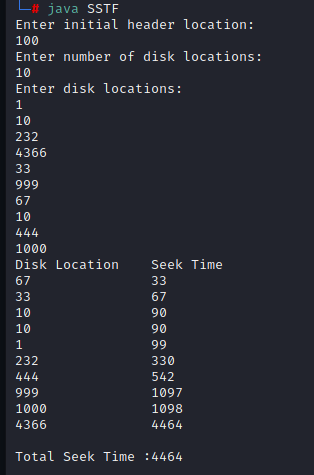
+Vị trí tất cả các yêu cầu

Đầu ra:

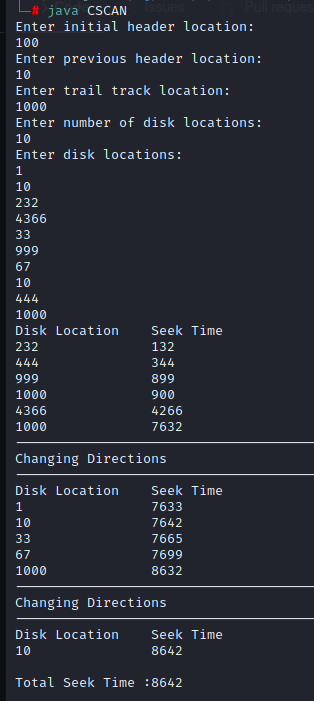
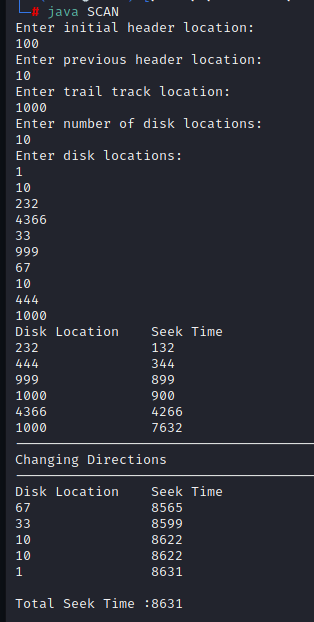
+Tổng thời gian tìm kiếm để đọc hết các yêu cầu



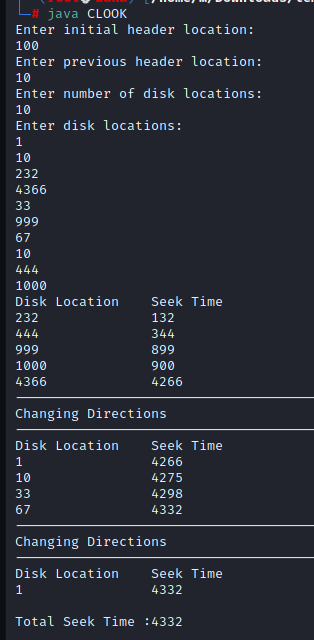
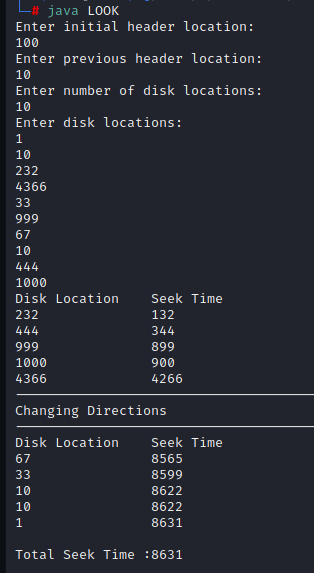
FCFS chạy rất lâu nếu yêu cầu không được sắp xếp hợp lý



SSTF tìm đến yêu cầu gần nhất tốt trong đa số trường hợp



Tùy vào việc chọn vị trí trail track hợp lý mà CSCAN có thể nhanh hơn hay chậm hơn SCAN



Trong trường hợp này CLOOK nhanh hơn hẳn do quay về đầu thay vì quay đầu tại vị trí cuối

* Khuyên dùng SCAN ,SSTF,CLOOK ,LOOK

Phân tích bài toán Producer Consumer Problem

Producer/ Consumer là một ví dụ kinh điển về vấn đề đồng hóa các luồng (multi-threading synchronization)

Vấn đề mô tả hai đối tượng nhà sản xuất (Producer) và người tiêu dùng (Consumer), cả hai cùng chia sẻ một bộ đệm có kích thước cố định được sử dụng như một hàng đợi (queue).

**Producer**: công việc của nhà sản xuất là tạo dữ liệu, đưa nó vào bộ đệm và bắt đầu lại.

**Consumer**: công việc người tiêu dùng là tiêu thụ dữ liệu (nghĩa là loại bỏ nó khỏi bộ đệm), từng phần một và xử lý nó. Consumer và Producer hoạt động song song với nhau.

Vấn đề là đảm bảo rằng nhà sản xuất không thể thêm dữ liệu vào bộ đệm nếu nó đầy và người tiêu dùng không thể xóa dữ liệu khỏi bộ đệm trống, đồng thời đảm bảo an toàn cho luồng (thread-safe).

Ý tưởng về giải pháp:

* Giải pháp cho nhà sản xuất là đi ngủ (wait) nếu bộ đệm đầy. Lần tiếp theo người tiêu dùng xóa một mục khỏi bộ đệm, nó đánh thức (notify) nhà sản xuất, bắt đầu đưa dữ liệu vào bộ đệm.
* Theo cách tương tự, người tiêu dùng có thể đi ngủ (wait) nếu thấy bộ đệm trống. Lần tiếp theo nhà sản xuất đưa dữ liệu vào bộ đệm, nó đánh thức (notify) người tiêu dùng đang ngủ (wait).
* Trong khi làm tất cả điều này, phải đảm bảo an toàn cho luồng (thread safe).

Phân tích bài toán Dining Philosopher's problem

Cho 5 triết gia ngồi chung một bàn tròn với 5 chiếc đũa xếp xem kẽ giữa 2 người ngồi cạnh nhau.

Mỗi triết gia tìm cách để ăn được thức ăn từ đĩa của mình với điều kiện: “chỉ ai có 2 chiếc đũa cạnh mình mới được phép ăn”, do đó họ lần lượt đổi trạng thái giữa ăn (eating) và đợi (thinking). Mỗi người sau khi giữa đôi đũa để ăn sau 1 khoảng thời gian phải bỏ lại 2 chiếc đũa về vị trí cũ để tiếp tục quá trình này. Yêu cầu: tìm một phương pháp đảm bảo để các triết gia đều có thể đk ăn / đợi đổi lượt để không ai bị chết đói (chỉ đợi chứ không được ăn).

Mỗi triết gia sẽ tượng trưng cho một tiến trình, và những chiếc đũa là tài nguyên của hệ thống. Các triết gia sẽ có 4 trạng thái lần lượt theo thứ tự là:

1. Suy nghĩ ( Thinking )
2. Lấy đũa ( Take Choptisks )
3. Đang ăn ( Eating )
4. Thả đũa ( Put Choptisks )

Semaphore là lời giải kinh điển cho bài toán bữa tối của các triết gia (dining philosophers), mặc dù nó không ngăn được hết các Deadlock. Để giải quyết Deadlock, ở vị trí cuối cùng ta sẽ cho triết gia đó được lấy đũa bên phải mình trước sau đó mới lấy đũa bên trái trong khi các triết gia khác sẽ lấy đũa bên trái trước. Điều đó chắc chắn rằng triết gia vị trí cuối luôn sẵn sàng ăn đầu tiên khi mà các triết gia khác đang cố lấy chiếc đũa bên phải của mình.