BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI TẬP LỚN
BỘ MÔN: HỆ ĐIỀU HÀNH

Chủ đề 19: Giải pháp cho bài toán Người thợ cắt tóc và mô tả thuật toán đổi trang theo nguyên tắc FIFO

Nhóm học:	Nhóm 13	
Giảng viên hướng dẫn:	Vũ Anh Đào	
Sinh viên thực hiện :	Trần Đình Hào	: B22DCCN278
	Bùi Tiến Dũng	: B22DCCN122
	Nguyễn Thị Hiền	: B22DCCN289
	Nguyễn Đức Anh	: B22DCCN026

Hà Nội, 11/2024

Mục lục

Phần 1

Giới thiệu: Đề tài 19

1.1 Bối cảnh và động cơ nghiên cứu

Trong thời đại hiện nay, các vấn đề về tối ưu hóa tài nguyên và hiệu quả xử lý trong hệ điều hành ngày càng trở nên quan trọng. Việc mô phỏng và phân tích các giải pháp cho bài toán kinh điển không chỉ giúp hiểu rõ nguyên lý hoạt động của hệ điều hành mà còn tạo cơ sở cho việc phát triển các ứng dụng thực tế.

Hai bài toán "Người thợ cắt tóc" và "Thuật toán đổi trang theo nguyên tắc FIFO" đều là những vấn đề quan trọng trong lĩnh vực đồng bộ hóa và quản lý bộ nhớ trong hệ điều hành. Trong đó:

- **Bài toán Người thợ cắt tóc:** Minh họa bài toán đồng bộ hóa kinh điển giữa các tiến trình trong môi trường đa luồng hoặc đa tiến trình.
- **Thuật toán đổi trang FIFO:** Là một thuật toán cơ bản trong quản lý bộ nhớ, sử dụng nguyên tắc FIFO - "First In, First Out" (Vào trước, ra trước) để thay thế trang.

Thông qua việc nghiên cứu và xây dựng các chương trình mô phỏng, đề tài này không chỉ làm rõ các khái niệm lý thuyết mà còn cung cấp công cụ thực hành hữu ích.

1.2 Mục tiêu đề tài

Đề tài được thực hiện để giải quyết các vấn đề sau:

1. **Xây dựng chương trình minh họa giải pháp cho bài toán Người thợ cắt tóc:** Đảm bảo đồng bộ hóa giữa các tiến trình "Thợ cắt tóc" và "Khách hàng". Hiểu rõ cách sử dụng các công cụ đồng bộ hóa như semaphore, mutex hoặc điều kiện.
2. **Mô tả và cài đặt thuật toán đổi trang theo nguyên tắc FIFO:** Viết chương trình mô tả thuật toán đổi trang theo nguyên tắc FIFO cho tiến trình được cấp 4 khung, không gian nhớ logic của tiến trình gồm 6 trang và các trang của tiến trình.Được truy cập 10 lần theo thứ tự nhập từ bàn phím.
3. Cung cấp các minh họa trực quan giúp người đọc dễ dàng nắm bắt quy trình hoạt động cũng như cách thức triển khai bài toán.

1.3 Phạm vi nghiên cứu

- **Bài toán Người thợ cắt tóc:** Giải quyết bài toán trong môi trường giả lập (mô phỏng đồng bộ hóa với các tiến trình hoặc luồng), sử dụng ngôn ngữ lập trình hỗ trợ cơ chế đồng bộ Java.
- **Thuật toán đổi trang FIFO:** Tập trung vào việc mô phỏng thuật toán trong không gian nhớ logic của một tiến trình. Không mở rộng sang các thuật toán thay thế trang khác như LRU hay OPT.

1.4 Phương pháp thực hiện

Để đạt được mục tiêu đề tài, các bước thực hiện bao gồm:

1. **Nghiên cứu lý thuyết:** Tìm hiểu về bài toán Người thợ cắt tóc và các công cụ đồng bộ hóa (semaphore, mutex). Nghiên cứu nguyên lý hoạt động của thuật toán đổi trang FIFO.
2. **Thiết kế và lập trình:** Viết chương trình minh họa bài toán Người thợ cắt tóc sử dụng các công cụ đồng bộ. Xây dựng chương trình mô phỏng thuật toán đổi trang FIFO, cho phép người dùng nhập dãy truy cập trang từ bàn phím.
3. **Kiểm thử và đánh giá:** Chạy thử chương trình với các kịch bản khác nhau để đảm bảo tính chính xác và hiệu quả. Đánh giá kết quả và rút ra bài học thực tiễn.

1.5 Kết cấu báo cáo

Báo cáo được trình bày theo cấu trúc sau:

- **Giới thiệu:** Tổng quan về đề tài, mục tiêu và phương pháp thực hiện.
- **Cơ sở lý thuyết:** Trình bày lý thuyết về bài toán Người thợ cắt tóc và thuật toán đổi trang FIFO.
- **Thiết kế và cài đặt:** Chi tiết việc xây dựng chương trình mô phỏng.
- **Kết quả và đánh giá:** Mô tả kết quả thu được từ chương trình và các nhận xét đánh giá.
- **Kết luận:** Tóm tắt các kết quả đạt được và đề xuất hướng phát triển trong tương lai.

Phần 2

Bài toán Người thợ cắt tóc

2.1 Giới thiệu bài toán

2.1.1 Giới thiệu chung

Bài toán Người thợ cắt tóc là một bài toán kinh điển trong lĩnh vực đồng bộ hóa tiến trình. Dijkstra đã đề xuất bài toán này vào năm 1965 để chỉ ra sự phức tạp khi có nhiều hơn một tiến trình hệ điều hành.

2.1.2 Mô tả bài toán

Bài toán Người thợ cắt tóc được trình bày như sau:

- **Bối cảnh:**
  - Một tiệm cắt tóc nhỏ có một thợ cắt tóc và một số lượng ghế chờ cố định trong phòng chờ cho khách hàng.
- **Cấu trúc của tiệm cắt tóc:**
  - **Ghế cắt tóc:** Chỉ có một ghế cắt tóc duy nhất. Thợ cắt tóc chỉ có thể cắt tóc cho một khách hàng tại một thời điểm.
  - **Ghế chờ:** Có n ghế chờ trong phòng chờ dành cho khách hàng. Số lượng ghế chờ là cố định và hạn chế.
- **Hành vi của thợ cắt tóc:**
  - Nếu không có khách hàng, thợ cắt tóc sẽ ngồi trên ghế cắt tóc và ngủ.
  - Khi có khách hàng đến và đánh thức, thợ cắt tóc sẽ cắt tóc cho khách hàng đó.
  - Sau khi cắt tóc xong cho một khách hàng, thợ cắt tóc kiểm tra phòng chờ. Nếu có khách hàng đang chờ, thợ cắt tóc sẽ mời khách tiếp theo vào và cắt tóc. Nếu không có khách hàng nào đang chờ, thợ cắt tóc sẽ tiếp tục ngủ.
- **Hành vi của khách hàng:** Khi một khách hàng đến tiệm
  - Nếu thợ cắt tóc đang ngủ và không có khách hàng khác đang được phục vụ, khách hàng sẽ đánh thức thợ cắt tóc và được cắt tóc ngay.
  - Nếu thợ cắt tóc đang bận và còn chỗ trống trong phòng chờ, khách hàng sẽ ngồi đợi đến lượt mình.
  - Nếu phòng chờ đã đầy (không còn ghế trống), khách hàng sẽ rời đi mà không cắt tóc.

Bài toán này thường được giải bằng cách sử dụng các công cụ đồng bộ hóa như **semaphore**, **mutex**, hoặc các điều kiện chờ (*condition variables*).

2.1.3 Vấn đề đồng bộ hóa

- **Tài nguyên chung:**
  - **Ghế cắt tóc:** Chỉ một khách hàng có thể ngồi trên ghế cắt tóc và được phục vụ tại một thời điểm.
  - **Ghế chờ:** Số lượng ghế chờ hạn chế, cần quản lý để tránh vượt quá số lượng ghế có sẵn.

- **Yêu cầu đồng bộ hóa:**
  - Đảm bảo rằng chỉ một khách hàng có thể tương tác với thợ cắt tóc tại một thời điểm.
  - Quản lý số lượng khách hàng trong phòng chờ để không vượt quá số ghế chờ.
  - Tránh điều kiện deadlock (kẹt cứng), nơi mà thợ cắt tóc và khách hàng đều chờ đợi lẫn nhau và không ai tiến hành hành động.
  - Tránh điều kiện starvation (đói), đảm bảo mọi khách hàng vào được phòng chờ sẽ cuối cùng được phục vụ.

2.1.4 Mô tả vấn đề

- **Quản lý đồng bộ trong môi trường đa luồng:**
  - Cần sử dụng các cơ chế đồng bộ hóa như mutex, semaphore, hoặc monitor để quản lý truy cập đến các tài nguyên chung.
  - Đảm bảo rằng các thao tác trên biến chung (ví dụ: số lượng khách hàng đang chờ) phải là nguyên tử để tránh điều kiện tranh chấp (race condition).

- **Xử lý tình huống bất đồng bộ:**
  - Khách hàng đến tiệm vào các thời điểm ngẫu nhiên.
  - Thời gian cắt tóc cho mỗi khách hàng có thể khác nhau.

2.1.5 Mục tiêu

- **Đối với thợ cắt tóc:**
  - Thợ cắt tóc sẽ hoạt động hiệu quả, cắt tóc cho khách hàng nếu có, hoặc ngủ nếu không có khách.

- **Đối với khách hàng:**
  - Khách hàng nếu vào được phòng chờ sẽ cuối cùng được phục vụ.
  - Thời gian cắt tóc cho mỗi khách hàng có thể khác nhau.

Kết luận: Mục tiêu chính là **đảm bảo tính nhất quán và tránh lỗi:**
- Tránh các tình huống mà nhiều khách hàng được phục vụ cùng lúc.
- Tránh việc khách hàng chiếm ghế chờ khi không còn chỗ trống.
- Đảm bảo rằng không có khách hàng nào bị bỏ quên trong phòng chờ.

2.1.6 Các yếu tố cần xem xét trong giải pháp

- **Sử dụng semaphore và mutex:**
  - **Semaphore đếm (Counting Semaphore):** Để quản lý số lượng ghế chờ và khách hàng đang chờ.

- **Mutex (Mutual Exclusion):** Để đảm bảo truy cập độc quyền đến các biến chia sẻ, như biến đếm số khách hàng.

- **Quy trình của thợ cắt tóc:**
  - Chờ đợi khách hàng (có thể ngủ).
  - Khi có khách hàng, thực hiện cắt tóc.
  - Sau khi cắt tóc xong, kiểm tra xem có khách hàng đang chờ không.

- **Quy trình của khách hàng:**
  - Khi đến tiệm, kiểm tra xem có chỗ trong phòng chờ không: Nếu có, ngồi chờ và thông báo cho thợ cắt tóc; còn nếu không thì rời đi.
  - Khi được thợ cắt tóc gọi, tiến vào ghế cắt tóc và được phục vụ.

2.2 Giải quyết vấn đề

2.2.1 Cài đặt thuật toán

1. **Khởi tạo các thành phần đồng bộ hóa:**
   - **mutex = new Semaphore(1):** Đảm bảo truy cập tuần tự vào biến dùng chung **waitingCustomers**.
   - **customers = new Semaphore(0):** Đại diện số lượng khách đang đợi. Nếu **customers = 0**, thợ cắt tóc sẽ ngủ.
   - **haircut = new Semaphore(0):** Quản lý việc đồng bộ giữa thợ cắt tóc và khách hàng.

2. **Biến hỗ trợ:**
   - **waitingCustomers = 0:** Lưu trữ số lượng khách đang đợi trong tiệm.
   - **N = 10:** Số ghế chờ tối đa.
   - **TOTAL_CUSTOMERS = 20:** Tổng số lượng khách đến tiệm.
   - **random:** Tạo thời gian ngẫu nhiên cho hành động của khách và thợ.

3. **Hành vi khách hàng (`Customer`):**
   - Khách hàng vào tiệm sau một khoảng thời gian ngẫu nhiên.
   - Kiểm tra số lượng ghế chờ:
     - Nếu **waitingCustomers** $\geq$ **N**, khách rời đi.
     - Nếu còn ghế trống:
       - Tăng **waitingCustomers**.
       - Báo hiệu (`customers.release()`) để đánh thức thợ.
       - Chờ tín hiệu hoàn thành cắt tóc từ thợ (`haircut.acquire()`).

4. **Hành vi thợ cắt tóc (`Barber`):**
   - Thợ chờ tín hiệu từ khách (`customers.acquire()`).
   - Khi có khách:
     - Giảm **waitingCustomers**.
     - Thực hiện cắt tóc trong thời gian ngẫu nhiên.

- Sau khi cắt xong, gửi tín hiệu (`haircut.release()`) cho khách hàng.

5. **Luồng thực thi (`main`):**
   - Tạo một luồng cho thợ cắt tóc.
   - Tạo nhiều luồng khách hàng (tương ứng với **TOTAL_CUSTOMERS**).
   - Đợi tất cả các khách hàng được phục vụ trước khi đóng tiệm.

2.2.2 Chương trình Java

```
1
2  import java.util.LinkedList;
3  import java.util.Queue;
4  import java.util.Random;
5  import java.util.concurrent.Semaphore;
6
7  public class BarberShop {
8
9      private static final int N = 10; // So ghe cho toi da
10     private static final int TOTAL_CUSTOMERS = 20; // Tong so khach hang
11     private static int waitingCustomers = 0;
12     private static Semaphore mutex = new Semaphore(1);
13     private static Semaphore haircut = new Semaphore(0); // Cho tin hieu tu tho cat
       toc
14     private static Semaphore customers = new Semaphore(0); // So khach dang doi
15     private static Random random = new Random();
16     private static Queue<Customer> waitingQueue = new LinkedList<>();
17
18     static class Customer implements Runnable {
19
20         private int id;
21
22         public Customer(int id) {
23             this.id = id;
24         }
25
26         public void run() {
27             try {
28                 // Khach den sau mot khoang thoi gian ngau nhien
29                 Thread.sleep(random.nextInt(5000));
30
31                 mutex.acquire();
32                 System.out.printf("Khach id %d vao cua hang, ", id);
33
34                 if (waitingCustomers >= N) {
35                     System.out.printf("thay kin cho va roi di\n");
36                     mutex.release();
37                     return;
38                 }
39
40                 waitingCustomers++;
41                 waitingQueue.offer(this);
42                 System.out.printf("ngoi cho (Co tong cong %d khach dang cho)\n",
       waitingCustomers);
```

```
                mutex.release();

                customers.release(); // Bao hieu cho tho cat toc
                haircut.acquire(); // Cho tin hieu hoan thanh cat toc

                // Xong
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    static class Barber implements Runnable {

        public void run() {
            while (true) {
                try {
                    System.out.println("Tho cat toc dang cho khach tiep theo");
                    customers.acquire(); // Doi khach hang
                    mutex.acquire();

                    Customer servingCustomer = waitingQueue.poll();
                    waitingCustomers--; // Giam so khach dang cho
                    System.out.printf("Tho bat dau phuc vu khach id %d (%d khach con
    lai dang cho)\n", servingCustomer.id, waitingCustomers);

                    mutex.release();

                    // Toi gian cat toc
                    Thread.sleep(random.nextInt(3000) + 2000);

                    System.out.printf("Tho hoan thanh cat toc cho khach id %d!\n",
    servingCustomer.id);
                    haircut.release(); // Thong bao khach hang da xong

                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    public static void main(String[] args) {
        Thread barberThread = new Thread(new Barber());
        barberThread.start();

        // Tao nhieu luong khach hang
        Thread[] customerThreads = new Thread[TOTAL_CUSTOMERS];
        for (int i = 0; i < TOTAL_CUSTOMERS; i++) {
            customerThreads[i] = new Thread(new Customer(i));
            customerThreads[i].start();
        }

        // Doi tat ca khach hang hoan thanh
```

```
        try {
            for (Thread customer : customerThreads) {
                customer.join();
            }
            // Ket thuc chuong trinh sau khi hoan thanh phuc vu tat ca cac khach
            System.out.println("\nHet khach, cua hang dong cua!");
            System.exit(0);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

2.2.3 Kết quả chương trình

Ví dụ về 1 kết quả đầu ra:

```
Tho cat toc dang cho khach tiep theo
Khach id 19 vao cua hang, ngoi cho (Co tong cong 1 khach dang cho)
Tho bat dau phuc vu khach id 19 (0 khach con lai dang cho)
Khach id 14 vao cua hang, ngoi cho (Co tong cong 1 khach dang cho)
Khach id 12 vao cua hang, ngoi cho (Co tong cong 2 khach dang cho)
Khach id 3 vao cua hang, ngoi cho (Co tong cong 3 khach dang cho)
Khach id 1 vao cua hang, ngoi cho (Co tong cong 4 khach dang cho)
Khach id 17 vao cua hang, ngoi cho (Co tong cong 5 khach dang cho)
Khach id 18 vao cua hang, ngoi cho (Co tong cong 6 khach dang cho)
Khach id 7 vao cua hang, ngoi cho (Co tong cong 7 khach dang cho)
Tho hoan thanh cat toc cho khach id 19!
Tho cat toc dang cho khach tiep theo
Tho bat dau phuc vu khach id 14 (6 khach con lai dang cho)
Khach id 5 vao cua hang, ngoi cho (Co tong cong 7 khach dang cho)
Khach id 4 vao cua hang, ngoi cho (Co tong cong 8 khach dang cho)
Khach id 6 vao cua hang, ngoi cho (Co tong cong 9 khach dang cho)
Khach id 9 vao cua hang, ngoi cho (Co tong cong 10 khach dang cho)
Khach id 11 vao cua hang, thay kin cho va roi di
Khach id 2 vao cua hang, thay kin cho va roi di
Khach id 0 vao cua hang, thay kin cho va roi di
Khach id 16 vao cua hang, thay kin cho va roi di
Khach id 15 vao cua hang, thay kin cho va roi di
Khach id 13 vao cua hang, thay kin cho va roi di
Khach id 8 vao cua hang, thay kin cho va roi di
Khach id 10 vao cua hang, thay kin cho va roi di
Tho hoan thanh cat toc cho khach id 14!
Tho cat toc dang cho khach tiep theo
Tho bat dau phuc vu khach id 12 (9 khach con lai dang cho)
Tho hoan thanh cat toc cho khach id 12!
Tho cat toc dang cho khach tiep theo
Tho bat dau phuc vu khach id 3 (8 khach con lai dang cho)
Tho hoan thanh cat toc cho khach id 3!
Tho cat toc dang cho khach tiep theo
Tho bat dau phuc vu khach id 1 (7 khach con lai dang cho)
Tho hoan thanh cat toc cho khach id 1!
Tho cat toc dang cho khach tiep theo
Tho bat dau phuc vu khach id 17 (6 khach con lai dang cho)
Tho hoan thanh cat toc cho khach id 17!
```

```
Tho cat toc dang cho khach tiep theo
Tho bat dau phuc vu khach id 18 (5 khach con lai dang cho)
Tho hoan thanh cat toc cho khach id 18!
Tho cat toc dang cho khach tiep theo
Tho bat dau phuc vu khach id 7 (4 khach con lai dang cho)
Tho hoan thanh cat toc cho khach id 7!
Tho cat toc dang cho khach tiep theo
Tho bat dau phuc vu khach id 5 (3 khach con lai dang cho)
Tho hoan thanh cat toc cho khach id 5!
Tho cat toc dang cho khach tiep theo
Tho bat dau phuc vu khach id 4 (2 khach con lai dang cho)
Tho hoan thanh cat toc cho khach id 4!
Tho cat toc dang cho khach tiep theo
Tho bat dau phuc vu khach id 6 (1 khach con lai dang cho)
Tho hoan thanh cat toc cho khach id 6!
Tho cat toc dang cho khach tiep theo
Tho bat dau phuc vu khach id 9 (0 khach con lai dang cho)
Tho hoan thanh cat toc cho khach id 9!
Tho cat toc dang cho khach tiep theo

Het khach, cua hang dong cua!
```

2.2.4 Kết luận chương trình

Như vậy các tiến trình đã được điều độ đúng.

Phần 3

Thuật toán đổi trang theo nguyên tắc FIFO

3.1 Cơ sở lý thuyết

3.1.1 Khái niệm quản lý bộ nhớ ảo

Bộ nhớ ảo là một cơ chế quan trọng trong hệ điều hành, cho phép ánh xạ không gian địa chỉ logic của tiến trình vào không gian địa chỉ vật lý của hệ thống. Khi bộ nhớ vật lý không đủ để chứa toàn bộ dữ liệu của một tiến trình, hệ điều hành sẽ lưu trữ các phần không cần thiết (trang) trong bộ nhớ phụ (như ổ đĩa) và chỉ tải chúng vào bộ nhớ chính khi cần thiết.

3.1.2 Thuật toán thay thế trang

Trong bộ nhớ ảo, khi một tiến trình cần truy cập một trang không nằm trong bộ nhớ chính, xảy ra lỗi trang (*page fault*). Hệ điều hành phải chọn một trang hiện có để thay thế. Thuật toán thay thế trang FIFO (*First In, First Out*) hoạt động theo nguyên tắc:

- Trang nào được nạp vào bộ nhớ đầu tiên sẽ bị thay thế đầu tiên khi xảy ra lỗi trang.
- Một hàng đợi (*queue*) được sử dụng để quản lý thứ tự các trang trong bộ nhớ.

3.1.3 Nguyên lý hoạt động của FIFO

Ví dụ, giả sử tiến trình có không gian nhớ logic gồm 6 trang và bộ nhớ vật lý có 4 khung. Khi dãy truy cập các trang là:

$$\{1, 2, 3, 4, 5, 1, 2, 3, 4, 5\}$$

Thuật toán hoạt động như sau:

1. **Lượt truy cập đầu tiên (1, 2, 3, 4):** Nạp lần lượt các trang 1, 2, 3, 4 vào bộ nhớ, không có trang nào bị thay thế.
2. **Lượt truy cập thứ năm (5):** Bộ nhớ đã đầy, trang 1 (vào trước) bị thay thế bởi trang 5.
3. **Lượt truy cập tiếp theo (1):** Trang 2 bị thay thế bởi trang 1, và quá trình tiếp tục.

3.1.4 Hạn chế của FIFO

Thuật toán FIFO tuy đơn giản nhưng có một số nhược điểm:

- Không đảm bảo hiệu quả trong việc tối ưu hóa số lượng lỗi trang.
- Có thể xảy ra hiện tượng **Belady's Anomaly**, khi tăng số khung bộ nhớ dẫn đến số lỗi trang cũng tăng.

3.1.5 Ứng dụng thực tiễn

Thuật toán FIFO thường được sử dụng trong:

- Các hệ thống đơn giản hoặc có tài nguyên hạn chế.
- Mô phỏng và giảng dạy về quản lý bộ nhớ.

3.2 Giải quyết bài toán

Đề bài: Viết chương trình mô tả thuật toán đổi trang theo nguyên tắc FIFO cho tiến trình được cấp 4 khung, không gian nhớ logic của tiến trình gồm 6 trang và các trang của tiến trình được truy cập 10 lần theo thứ tự nhập từ bàn phím.

3.2.1 Tổng quan bài toán

Cần viết một chương trình mô phỏng thuật toán thay thế trang (page replacement) theo nguyên tắc FIFO (First-In-First-Out), trong bối cảnh một hệ thống quản lý bộ nhớ ảo với các thông tin như sau:

- **Tiến trình được cấp 4 khung:** Bộ nhớ vật lý (RAM) chỉ có thể chứa tối đa 4 trang (*pages*) cùng một lúc.
- **Không gian nhớ logic của tiến trình gồm 6 trang:** Tiến trình có tổng cộng 6 trang logic (được đánh số từ 1 đến 6) mà nó cần truy cập.
- **Truy cập 10 lần:** Tiến trình sẽ truy cập các trang logic này 10 lần, theo thứ tự mà người dùng nhập vào từ bàn phím.

Khi một trang được truy cập mà không có trong bộ nhớ vật lý, xảy ra lỗi trang (*page fault*). Nếu bộ nhớ vật lý đã đầy, cần thay thế một trang cũ theo nguyên tắc FIFO để đưa trang mới vào.

3.2.2 Chương trình mô phỏng

Dưới đây là mã nguồn chương trình mô tả thuật toán đổi trang theo nguyên tắc FIFO cho tiến trình được cấp 4 khung, không gian nhớ logic của tiến trình gồm 6 trang và các trang của tiến trình được truy cập 10 lần theo thứ tự nhập từ bàn phím.

```
1  package hdh;
2
3  import java.util.*;
4
5  public class FIFO19 {
6
7      public static void main(String[] args) {
8          Scanner sc = new Scanner(System.in);
9          int soKhung = 4, soTrang = 6, n = 10;
10         int a[] = new int[n];
11         int d[] = new int[1005];
12         System.out.print("Input: ");
13         for (int i = 0; i < n; i++) {
```

```
            a[i] = sc.nextInt();
        }
        Queue<Integer> qu = new LinkedList<>();
        int loiTrang = 0, cnt = 0;
        for (int i = 0; i < n; i++) {
            boolean check = false;
            if (!qu.contains(a[i])) {
                if (qu.size() == soKhung) {
                    check = true;
                    loiTrang++;
                    int x = qu.poll();
                    for (int j = 1; j <= soKhung; j++) {
                        if (d[j] == x) {
                            d[j] = a[i];
                            break;
                        }
                    }
                } else {
                    cnt++;
                    d[cnt] = a[i];
                }
                qu.add(a[i]);
            }
            System.out.print("Them trang: " + a[i] + " ");
            System.out.print("Hang doi: ");
            for (int j = 1; j <= soKhung; j++) {
                if (d[j] != 0) {
                    System.out.print(d[j] + " ");
                }
            }
            if (check) {
                System.out.print("F");
            }
            System.out.println("");
        }
        System.out.println("So lan loi trang: " + loiTrang);
    }
}
```

3.2.3 Kết quả chương trình

Ví dụ: Với dãy truy cập `1 2 3 5 4 6 3 1 2 6`, chương trình sẽ tạo ra kết quả hiển thị trạng thái bộ nhớ và số lỗi trang tương ứng.

```
Input: 1 2 3 5 4 6 3 1 2 6
Them trang: 1 Hang doi: 1
Them trang: 2 Hang doi: 1 2
Them trang: 3 Hang doi: 1 2 3
Them trang: 5 Hang doi: 1 2 3 5
Them trang: 4 Hang doi: 4 2 3 5 F
Them trang: 6 Hang doi: 4 6 3 5 F
Them trang: 3 Hang doi: 4 6 3 5
Them trang: 1 Hang doi: 4 6 1 5 F
Them trang: 2 Hang doi: 4 6 1 2 F
```

```
Them trang: 6 Hang doi: 4 6 1 2
So lan loi trang: 4
```

3.2.4 Kết luận chương trình

Chương trình sẽ:

- Hiển thị trạng thái bộ nhớ sau mỗi lần truy cập.
- Tính tổng số lỗi trang xảy ra sau khi truy cập toàn bộ các trang.

Phần 4

Tài liệu tham khảo

Bài toán Người thợ cắt tóc

1. Chu Đức Lộc và Phạm Quang Toản, *Bài tập lớn Hệ Điều Hành*, Viện Công nghệ Thông tin và Truyền thông, Đại học Bách Khoa Hà Nội, 2011, trang 15–19.
2. Lê Duy Anh Dũng, Trần Trung Dũng và Lê Đức Mạnh, *Sleeping Barber Problem*, Đại học Bách Khoa Hà Nội.
3. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, *Operating System Concepts*, 9th edition, John Wiley & Sons, Inc., 2012.
4. Allen B. Downey, *The Little Book of Semaphores*, 2nd edition, Green Tea Press, 2008. `http://greenteapress.com/semaphores/`.
5. Wikipedia, *Sleeping barber problem*, Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Sleeping_barber_problem`.

Thuật toán đổi trang theo nguyên tắc FIFO

1. Từ Minh Phương, *Giáo trình Hệ điều hành*, Học viện Công nghệ Bưu Chính Viễn Thông, 12/2015.