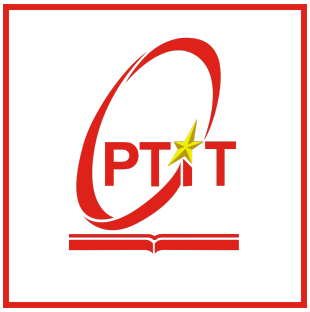
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO BÀI TẬP LỚN

Đề 19: Xây dựng chương trình minh họa bài toán Người thợ cắt tóc và chương trình mô tả thuật toán đổi trang theo nguyên tắc FIFO

Giảng viên:	Vũ Anh Đào
Nhóm bài tập:	19
Nhóm lớp:	12
Họ và tên:	Nguyễn Quốc Anh - B22DCAT019
	Vũ Trọng Khôi - B22DCCN468
	Mai Xuân Nhân - B22DCCN576
	Lê Tiến Đạt - B22DCCN188

Hà Nội, tháng 11 năm 2024

Mục lục

A Xây dựng chương trình minh họa giải pháp cho bài toán Người thợ cắt tóc **3**
1 Bế tắc và tình trạng đói tài nguyên 3
2 Bài toán người thợ cắt tóc 3
2.1 Mô tả bài toán 3
2.2 Vấn đề của bài toán 3
3 Giải quyết bài toán người thợ cắt tóc bằng phương pháp cổ điển 4
3.1 Ý tưởng bài toán 4
3.2 Giả mã của chương trình mô phỏng bài toán người thợ cắt tóc 4
3.3 Chương trình mô phỏng bài toán người thợ cắt tóc bằng ngôn ngữ Java 5
4 Giải quyết bài toán người thợ cắt tóc bằng phương pháp FIFO . 7
4.1 Ý tưởng bài toán 7
4.2 Giả mã chương trình mô phỏng bài toán người thợ cắt tóc bằng phương pháp FIFO 7
4.3 Chương trình mô phỏng bài toán người thợ cắt tóc phương pháp FIFO bằng ngôn ngữ Java 8

B Đổi trang và thuật toán FIFO **10**
1 Đổi trang là gì? Tại sao phải đổi trang 10
2 Thuật toán vào trước, ra trước (FIFO) 11
3 Ví dụ: 11
4 Giả mã thuật toán FIFO 12
5 Cài đặt thuật toán FIFO bằng Java 12

A Xây dựng chương trình minh họa giải pháp cho bài toán Người thợ cắt tóc

1 Bế tắc và tình trạng đói tài nguyên

Một vấn đề gây nhiều khó khăn đối với các tiến trình đồng thời là tình trạng bế tắc (deadlock). Bế tắc là tình trạng một nhóm tiến trình có cạnh tranh về tài nguyên hay có hợp tác phải dừng (phong tỏa) vô hạn. Lý do dừng vô hạn của nhóm tiến trình là do tiến trình phải chờ đợi một sự kiện chỉ có thể sinh ra bởi tiến trình khác cũng đang trong trạng thái chờ đợi của nhóm.

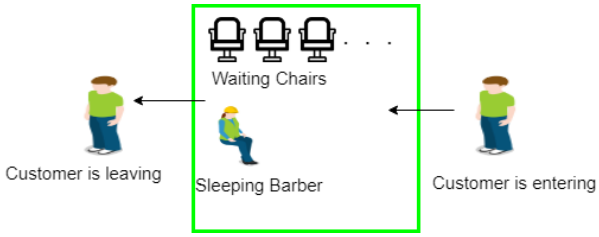
Đói tài nguyên là tình trạng chờ đợi quá lâu mà không đến lượt sử dụng tài nguyên nào đó.

2 Bài toán người thợ cắt tóc

2.1 Mô tả bài toán

Trạng thái của thợ cắt tóc có 3 trạng thái: khi không có người cắt tóc sẽ đi ngủ; khi có người cắt tóc sẽ cắt tóc; khi cắt tóc xong sẽ xem phòng chờ có người không, nếu không thì đi ngủ, có người thì gọi một người ngẫu nhiên vào cắt tóc.

Trạng thái của khách hàng: khi đến cắt tóc mà người cắt tóc ngủ sẽ đánh thức thợ cắt tóc để cắt tóc cho họ, khi người thợ cắt tóc đang cắt tóc thì khách hàng sẽ ở phòng chờ đợi thợ cắt tóc với n ghế có sẵn. Nếu phòng chờ đầy ghế sẽ đi về.



2.2 Vấn đề của bài toán

Khi một khách cắt tóc xong, không có cơ chế nào thông báo cho khách khác về việc thợ cắt tóc đang trống việc để gọi thợ cắt tóc, nên khách chỉ đơn thuần luôn nghĩ thợ cắt tóc đang cắt tóc. Mặt khác, cũng không có cơ chế nào để thợ cắt tóc biết được còn người cắt tóc, giả sử thợ cắt tóc gọi 1 người nhưng người đó đã rời đi từ trước, thợ cắt tóc sẽ hiểu nhầm là phòng chờ không có người và đi ngủ. Như vậy, thợ cắt tóc đợi khách gọi mình dậy, còn khách thì đợi thợ cắt tóc (do chỉ biết thợ cắt tóc đang cắt tóc).

Thêm nữa, khi một người khách khác tới muốn cắt, do ghế đã đầy nên khách này sẽ đi về, nên không thể có trường hợp khách mới tới gọi thợ cắt tóc.

Một tình huống khác xảy ra, đó là trên thực tế, các hành động đi tới phòng chờ, dừng cắt đều mất một khoảng thời gian nhất định. Trong thời gian đệm đó, khách rời đi, thợ cắt tóc sẽ đợi khách còn khách thì vẫn đang đợi thợ cắt tóc.

Chung quy lại, các trường hợp trên sẽ dẫn tới 2 vấn đề:

- Vấn đề 1: Thợ cắt tóc đợi khách và khách cũng đợi thợ cắt tóc. Hai bên đợi nhau dẫn tới tình trạng không bên nào hoạt động vô thời hạn dẫn tới bế tắc.
- Vấn đề 2: Việc chờ đợi do bế tắc có thể dẫn tới đói tài nguyên. Mặt khác, bởi không có cơ chế gọi cụ thể thợ cắt tóc có thể chỉ gọi 1 khách trong tất cả các lần, từ đó dẫn tới việc nhiều khách không được cắt tóc và chỉ một khách được cắt. Điều này cũng dẫn tới đói tài nguyên.

3 Giải quyết bài toán người thợ cắt tóc bằng phương pháp cổ điển

3.1 Ý tưởng bài toán

Vấn đề của bài toán người thợ cắt tóc đó là thiếu cơ chế giao tiếp giữa thợ cắt tóc và khách hàng. Chính vì vậy, ý tưởng giải quyết bài toán này là tạo ra các cờ hiệu (semaphore) để người thợ cắt tóc biết rằng đang có khách cần cắt và ngược lại, khách hàng biết thợ cắt tóc đang trống việc và có thể vào cắt.

Dựa trên ý tưởng đó, ta tạo cờ `barber = Semaphore(0)` để khách hàng chờ thợ cắt tóc, cờ `customer = Semaphore(0)` để thợ cắt tóc chờ khách hàng và tạo biến `waiting_customers` để biết số lượng khách đang chờ ở phòng chờ, nếu số lượng khách đã đầy mà có khách mới vào, khách sẽ tự rời đi thông qua hàm `balk()` (hàm chỉ duy nhất lệnh `return`, để kết thúc). Ngoài ra, ta sẽ thêm 1 cờ `mutex = Semaphore(1)` để tránh việc các luồng sẽ ảnh hưởng đến nhau, điều này cho phép 1 thời điểm chỉ duy nhất 1 luồng được hoạt động chung cho cả chương trình.

3.2 Giả mã của chương trình mô phỏng bài toán người thợ cắt tóc

Tạo các cờ, biến khởi tạo (số ghế ở phòng chờ mặc định là 10):

```
mutex = Semaphore(1);
customer = Semaphore(0);
barber = Semaphore(0);
waiting_customers = 0, N = 10;
```

Khách hàng thực hiện:

```
mutex.wait();
if customers == n :
mutex.signal();
balk();                              ▷ Khách rời đi nếu phòng chờ đầy
waiting_customers += 1;
mutex.signal();
```

13: customer.signal();
14: barber.wait();
15: **Thợ cắt tóc thực hiện:**
16: customer.wait();
17: mutex.wait();
18: waiting_customers -= 1;
19: mutex.signal();
20: barber.signal();

3.3 Chương trình mô phỏng bài toán người thợ cắt tóc bằng ngôn ngữ Java

```
import java.util.concurrent.Semaphore;

public class baberSolution {

    public static int waiting_customers = 0, N = 10;
    public static Semaphore mutex = new Semaphore(1);
    public static Semaphore customers = new Semaphore(0);
    public static Semaphore barber = new Semaphore(0);

    public static class Customer extends Thread {
        public void run() {
            while (true) {

                try {
                    mutex.acquire();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("Khách vào tiệm");

                if (waiting_customers >= N) {
                    mutex.release();
                    return;
                }

                waiting_customers += 1;

                mutex.release();

                customers.release();
                try {
                    barber.acquire();
                } catch (InterruptedException e) {
                    e.printStackTrace();
```

```
            }
        }
    }
}

public static class Barber extends Thread {
    public void run() {
        int t;
        boolean ok;
        while (true) {
            try {
                mutex.acquire();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            if (waiting_customers == 0) {
                ok = true;
                System.out.println("Thợ cắt tóc đi ngủ");
            } else {
                ok = false;
            }
            mutex.release();
            try {
                customers.acquire();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            try {
                mutex.acquire();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            if (ok) {
                System.out.println("Thợ cắt tóc dậy");
            }
            mutex.release();
            try {
                mutex.acquire();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            waiting_customers -= 1;
            System.out.println("Cắt tóc");
            mutex.release();
            System.out.println("Cắt tóc xong");
            barber.release();
```

```
            }
        }
    }

    public static void main(String[] args) {
        Customer cus = new Customer();
        Barber bar = new Barber();
        cus.start();
        bar.start();
    }
}
```

4 Giải quyết bài toán người thợ cắt tóc bằng phương pháp FIFO

Ở phần trước, ta đã nói về việc làm sao để tạo nên 1 cơ chế cho phép khách hàng và thợ cắt tóc biết nhau. Tuy nhiên, với phương pháp trên sẽ xảy ra trường hợp đó là, thợ cắt tóc chỉ cắt tóc cho 1 khách. Như vậy, phương pháp trên chỉ mới ngăn chặn tình trạng bế tắc chứ chưa ngăn chặn hoàn toàn tình trạng đói tài nguyên, bởi khi thợ chỉ cắt tóc cho 1 khách, các khách còn lại sẽ đợi nhưng rất lâu không cắt được tóc. Để giải quyết tình trạng này, chúng ta sẽ tạo nên cơ chế giúp thợ cắt tóc chọn khách hàng để cắt tóc công bằng. Ở đây, cơ chế để chọn là khách nào đến trước sẽ được cắt tóc trước (FIFO).

4.1 Ý tưởng bài toán

Thay vì sử dụng 1 cờ `Customer` cho tất cả các khách như trước, ta sẽ gán với từng khách có 1 cờ của riêng mình. Khi tới cửa hàng và phòng trống không đầy, khách sẽ đợi, luồng khách hàng được lưu vào trong hàng đợi. Thợ cắt tóc sẽ gọi khách trong hàng đợi thông qua cờ riêng của khách.

4.2 Giả mã chương trình mô phỏng bài toán người thợ cắt tóc bằng phương pháp FIFO

Giả mã chương trình mô phỏng bài toán người thợ cắt tóc bằng phương pháp FIFO

```
1: Tạo các cờ, biến khởi tạo
2: Semaphore customer = 0;
3: mutex = Semaphore(1);
4: customer = 0;
5: Queue queue;
6: n = 4;
7: Với khách hàng
8: Semaphore self = 0;
9: mutex.wait();
10: if customer == n then
```

```
11:    mutex.signal();
12: end if
13: customer += 1;
14: queue.push(self);
15: mutex.signal();
16: customer.signal();
17: self.wait();
18: customerDone.signal();
19: barberDone.wait();
20: mutex.wait();
21: customers -= 1;
22: mutex.signal();
23: Với thợ cắt tóc
24: Semaphore curr;
25: customer.wait();
26: mutex.wait();
27: curr = queue.pop();
28: mutex.signal();
29: curr.signal();
30: customerDone.wait();
31: barberDone.signal();
```

4.3 Chương trình mô phỏng bài toán người thợ cắt tóc phương pháp FIFO bằng ngôn ngữ Java

```
public static int maxCustomers = 5;
 public static int numCustomers = 0;
 public static Semaphore mutex = new Semaphore(1);
 public static Semaphore customer = new Semaphore(0);
 public static Semaphore customerDone = new Semaphore(0);
 public static Semaphore barberDone = new Semaphore(0);
 public static Queue<Customer> queue = new LinkedList<>();

 public static class Customer extends Thread {

  int id;
  private Semaphore semaphore = new Semaphore(0);

  public Customer(int id) {
   this.id = id;
  }

  public void acquire() throws InterruptedException {
   semaphore.acquire();
  }
```

```
 public void release() {
  semaphore.release();
 }

 private void getHairCut() throws InterruptedException {
   System.out.printf("Khách thứ %d đang cắt tóc\n", id);
 }

 public void run() {
  try {
   System.out.printf("Khách hàng thứ %d đã tới\n", id);
   mutex.acquire();
   if (numCustomers == maxCustomers) {
    mutex.release();
    System.out.printf("Khách hàng thứ %d đã đi vì phòng chờ đầy\n", id);
    return;
   }
   numCustomers++;
   queue.add(this);
   mutex.release();

   customer.release();
   this.acquire();

   getHairCut();

   mutex.acquire();
   numCustomers--;
   mutex.release();
  } catch (InterruptedException e) {
   e.printStackTrace();
  }
 }
}

public static class Barber extends Thread {
 public static void cutHair(Customer customer) throws InterruptedException {
  System.out.printf("Thợ cắt tóc đang cắt cho khách thứ %s\n", customer.id);
  Thread.sleep(10);
 }

 public void run() {
  try {
   while (true) {
    System.out.println("Thợ cắt tóc đang ngủ");
    Customer cust = null;
```

```
    try {
     customer.acquire();

     mutex.acquire();
     cust = queue.poll();
     mutex.release();

     cutHair(cust);
    } finally {
     if (cust != null) {
      cust.release();
     }
    }

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

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

 for (int i = 0;; i++) {
  try {
   Thread.sleep(50);
   Customer cus = new Customer(i);
   cus.start();
  } catch (InterruptedException e) {
   e.printStackTrace();
  }
 }
}
}
```

B Đổi trang và thuật toán FIFO

1 Đổi trang là gì? Tại sao phải đổi trang

Đổi trang Đổi trang là một quá trình trong hệ điều hành khi không có khung nào trống, hệ điều hành chọn một khung đã cấp phát nhưng hiện giờ không dùng tới và giải phóng khung trang này. Nội dung của khung được trao đổi ra

đĩa, trang nhớ chứa trong khung sẽ được đánh dấu không còn nằm trong bộ nhớ (bằng cách thay đổi bit P tương ứng) trong bảng phân trang có chứa trang này. Khung sẽ được giải phóng và cấp phát cho trang mới cần nạp vào.

Lý do phải đổi trang:

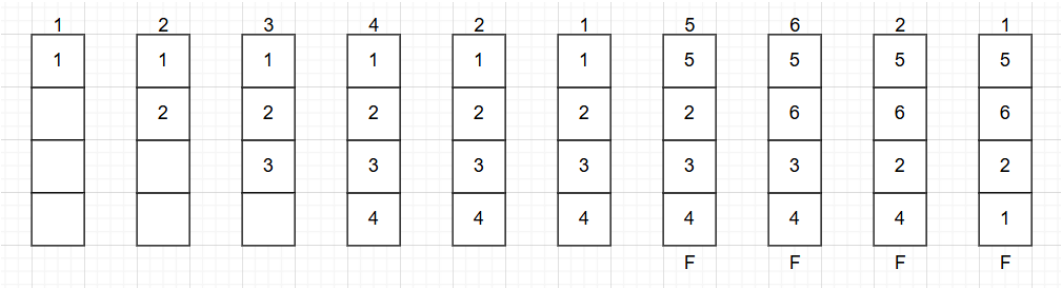
- Khắc phục tình trạng kích thước tiến trình lớn hơn kích thước bộ nhớ thực.
- Giúp hệ điều hành có thể chạy nhiều tiến trình cùng lúc.
- Tăng hiệu suất với một số thuật toán tối ưu như LRU, FIFO, LFU.
- Giúp hệ điều hành kiểm soát các trang truy cập của mỗi tiến trình, ngăn chặn truy cập trái phép vào bộ nhớ của tiến trình khác.

2 Thuật toán vào trước, ra trước (FIFO)

Mô tả: Dây là chiến lược đơn giản nhất. Trang được đọc vào bộ nhớ trước sẽ bị đổi ra trước khi có yêu cầu đổi trang. Chiến lược này có thể triển khai một cách đơn giản bằng cách sử dụng hàng đợi FIFO. Khi trang được nạp vào bộ nhớ, số thứ tự trang được thêm vào cuối hàng đợi. Khi cần đổi, trang có số thứ tự ở đầu hàng đợi sẽ bị đổi.

3 Ví dụ:

Xét 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.



Giải thích hoạt động của thuật toán:

- Từ trang 1 đến trang 4, do khung còn trống nên lần lượt nạp các trang vào khung như trong hình.
- Trang 2 và trang 1 ở lần truy cập thứ 5 và 6, do trong khung đã có nên khung vẫn giữ nguyên.

- Khi truy cập trang số 5, trong khung có các trang 1, 2, 3, 4. Trong đó, trang số 1 được nạp vào khung sớm nhất nên trang 1 bị đổi.
- Tương tự, khi truy cập trang số 6, trang 2 bị đổi. Quá trình này tiếp tục, trang 3 được đổi bởi trang 5 và trang 4 được đổi bởi trang 1.
- Kết quả: Trong 10 lần truy cập, có 4 lần đổi trang.

4 Giả mã thuật toán FIFO

Thuật toán:

```
// Khởi tạo
soKhung = 4
khung = Rỗng
viTriTraoDoi = 0    // Vị trí có trang được nạp vào bộ nhớ sớm nhất
soLanDoiTrang = 0
// Lặp
For 1 → 10:
    Nhập → trang     // Nhập trang muốn truy cập
    If (trang thuộc khung):
        In trạng thái khung;
    Else if (khung.size() < 4):
        Khung.add(trang);
        In trạng thái khung;
    Else:
        Khung[viTriTraoDoi] = trang;
        viTriTraoDoi ← (viTriTraoDoi + 1) % soKhung;
        soLanDoiTrang ← soLanDoiTrang + 1;
        In trạng thái khung + ký hiệu thay đổi;
// Kết thúc
Return(khung, soLanDoiTrang)
```

5 Cài đặt thuật toán FIFO bằng Java

Dưới đây là đoạn mã cài đặt thuật toán FIFO với ví dụ trên:

```
package HeDieuHanh;

import java.util.ArrayList;
import java.util.Scanner;

public class FIFO {
 public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // Số khung được cấp của tiến trình
        final int soKhung = 4;
```

```
        // Khung của tiến trình
        ArrayList<Integer> khung = new ArrayList<>();
        // Chỉ số trang bị trao đổi (Trang được nạp vào bộ nhớ sớm nhất)
        int viTriTraoDoi = 0;
        int soLanDoiTrang = 0;
        System.out.println("Thực hiện thuật toán FIFO:");
        for (int i = 0; i < 10; i++) {
         System.out.print("Nhập trang muốn truy cập: ");
         int trang = sc.nextInt();
            System.out.print("Trạng thái bộ nhớ :");
            // Nếu trang đã có trong khung: không thay đổi
            if (khung.contains(trang)) {
                System.out.println(khung);
            }
            // Nếu trang chưa có trong khung
            else {
                // Nếu khung chưa đầy, thêm trang vào khung
                if (khung.size() < soKhung) {
                    khung.add(trang);
                    System.out.println(khung);
                } else {
                    // Thay thế trang mới vào vị trí trang được nạp vào bộ nhớ sớm nhất
                    khung.set(viTriTraoDoi, trang);
                    // Cập nhật vị trí trao đổi kế tiếp
                    viTriTraoDoi = (viTriTraoDoi + 1) % soKhung;
                    soLanDoiTrang++;
                    System.out.println(khung + " F");
                }
            }
        }
        System.out.println("Số lần đổi trang: " + soLanDoiTrang);
        System.out.println("Kết thúc thuật toán FIFO!");
    }
}
```

Áp dụng với ví dụ trên cho ta kết quả:

```
Thực hiện thuật toán FIFO:
Nhập trang muốn truy cập: 1
Trạng thái bộ nhớ :[1]
Nhập trang muốn truy cập: 2
Trạng thái bộ nhớ :[1, 2]
Nhập trang muốn truy cập: 3
Trạng thái bộ nhớ :[1, 2, 3]
Nhập trang muốn truy cập: 4
Trạng thái bộ nhớ :[1, 2, 3, 4]
Nhập trang muốn truy cập: 2
Trạng thái bộ nhớ :[1, 2, 3, 4]
Nhập trang muốn truy cập: 1
Trạng thái bộ nhớ :[1, 2, 3, 4]
Nhập trang muốn truy cập: 5
Trạng thái bộ nhớ :[5, 2, 3, 4] F
Nhập trang muốn truy cập: 6
Trạng thái bộ nhớ :[5, 6, 3, 4] F
Nhập trang muốn truy cập: 2
Trạng thái bộ nhớ :[5, 6, 2, 4] F
Nhập trang muốn truy cập: 1
Trạng thái bộ nhớ :[5, 6, 2, 1] F
Số lần đổi trang: 4
Kết thúc thuật toán FIFO!
```

Tài liệu tham khảo

1. Allen B. Downey (2016), *The Little Book of Semaphores*
2. FIFO Barbershop in Process synchronization *[Trực tuyến]*. Địa chỉ: `https://www.geeksforgeeks.org/fifo-barbershop-in-process-synchronization/` *[Truy cập: 16/11/2024]*
3. Phạm Hải Đăng, *Giáo trình Nguyên lý Hệ điều hành – Đại học Bách khoa Hà Nội*
4. Từ Minh Phương (2015), *Giáo trình Hệ điều hành - Học viện Công nghệ Bưu chính viễn thông*