BỘ CÔNG THỨC TONY

HỆ THỐNG MÁY TÍNH

(Overview of Computer Systems for Non-Majors)

Pisces Kibo

MỤC LỤC

CHƯƠNG 1: TIẾN TRÌNH PROCESS

1. **Khái niệm về tiến trình:**
   - Tiến trình là chương trình được thực thi trong bộ nhớ và quản lý hệ điều hành
     - Semaphore cờ hiệu: là 1 kiểu biến nguyên không âm và chỉ cho phép thay đổi giá trị thông qua hai thao tác down và up
       → Ý nghĩa:
       - Biểu diễn số lượng tài nguyên
       - Xin tài nguyên sẽ down()
       - Kết thúc tài nguyên sẽ up()

     - Semaphore nhị phân (đèn báo): chỉ nhận giá trị 0 hoặc 1, được dùng để ngăn ngừa trên hai đoạn mã chương trình chạy song song
       - Down(&sem):
         + Nếu sem > 0 thì down giảm sem đi 1 đơn vị

         + Nếu sem = 0 thì down bị trì hoãn và chuyển sang block

       - Up(&sem) → tăng sem lên 1 đơn vị
   - Giải pháp luân phiên < Peterson < TSL < Semaphore:
     - Luân phiên → giới hạn số tiến trình
     - Peterson → có ưu tiên, 2 tiến trình
     - TSL → có ưu tiên, không giới hạn số tiến trình

2. **Các bài toán về Semaphore:**
   a) Bài toán gửi và rút tiền:

GIẢI PHÁP LUÂN PHIÊN

```
processA {
        NhapMaPin();
        NhapSoTien t;
        while (turn != 0);
                x = a
                x += t             Critical - Section
                a = t
        turn 1;

        InBienLai();
        TraThueTien();             Non Critical - Section
}
```

GIẢI PHÁP PETERSON

```
turn = 0
interested = [False, False]

// Gửi tiền
desposit(turn) {
        interested[0] = TRUE;
        turn = 1;
        while (interested[1] and turn)
                pass
        print(GuiTien);
        interested[0] = False;
}

// Rút tiền
withdraw(turn) {
        interested[1] = TRUE;
        turn = 0;
        while (interested[0] and turn):
                pass;
        print(RutTien);
        interested[1] = False;
}
```

b) Bài toán nhà sản xuất và người tiêu dùng:

```
semaphore empty N;
semaphore full = 0;
```

```
consumer() {
      int item;
      while (TRUE) {
            down(&full);
            down(&mutex);
            get_item(item);
            up(&mutex);
            up(&empty);
      consumer_item(item);
      }
}
```

```
producer() {
      int item;
      while (TRUE) {

      producer_item(item);
            down(&empty);
            down(&mutex);
            enter_item(item);
            up(&mutex);
            up(&full);
      }
}
```

c) Bài toán bữa ăn nhà hiền triết:

```
define THINKING 0;
define HUNGRY 1;
define EATING 2;
int state[N]; state[i];
semaphore mutex = 1;
semaphore S[i];         // 0 hoặc 1 (Hai đũa NHT i đã có)

LEFT(i) --> ((i-1+N)%N)
RIGHT(i) --> ((i+1) %N)

// Nhà hiền triết
philosophre(int i) {
      while (TRUE) {
            THINKING();
            take_chopsticks(i);
            eat();
            drop_chopsticks(i);
      }
}

// Lấy 2 đũa
take_chopsticks(i) {
      down(&mutex);
      state[i] = HUNGRY;
      test[i];    // Kiểm tra 2 đũa trên bàn không
      up(&mutex);
      down(S[i]); // Lấy 2 đũa
}

// Trả lại 2 đũa
drop_chopsticks(i) {
      down(&mutex);
      state[i] = THINKING;
      test(LEFT(i));
      test(RIGHT(i));
      up(&mutex);
}

// Kiểm tra trạng thái
test(i) {
      if ((state[i] == HUNGRY) and state[left, right] != EATING) {
            up(S[i]);
            state[i] = EATING;
      }
}
```

d) Bài toán cửa hàng cắt tóc:

```
define N;
int count = 0;    // đếm số khách chờ
semaphore barders = 0; // số thợ sẵn sàng
semaphore clients = 0; // số khách chờ
semaphore mutex = 1;
```

```
// Thợ cắt tóc
barder() {
      while (TRUE) {
            down(&client);    // ngủ nên chưa có khách
            down(&mutex);
            count--;
            up(&barder);      // Gọi thợ cắt tóc
            up(&mutex);
            cuthair();
      }
}

// Khách
client() {
      down(&mutex);
      if (count ≤ N) {
            count++;
            up(&client);      // gọi cắt tóc
            up(&mutex);
            down(&barder);    // ngủ nên chưa có
            get_haircut();
      }
}
```

e) Bài toán tạo phân tử nước:

```
int water_count; // đếm số phần tử H20
semaphore H_sem, O_sem;
semaphore mutex;
```

```
make_H() {
      while (TRUE) {
            down(&mutex);
            up(&H_sem);
            up(&mutex);
            sleep(1);
      }
}
```

```
make_O() {
      while (TRUE) {
            down(&mutex);
            up(&O_sem);
            up(&mutex);
            sleep(1);
      }
}
```

```
make_H2O() {
      while (TRUE) {
            down(&H_sem);
            down(&H_sem);
            down(&O_sem);

            down(&mutex);
            water_count++;
            printf(water_count);
            up(&mutex);
      }
}
```

f) Bài toán đọc và ghi:

```
semaphore db = 1;
int readcount = 0;
```

```
semaphore mutex = 1;
```

```
// Đọc dữ liệu
void reader(void) {
      while (TRUE) {
            down(&mutex);
            readcount++;
            if (readcount = 1) {
                  down(&db);
                  up(&mutex);
                  readData();
            }
      down(&mutex);
      readcount--;
      if (readcount = 0) {
            up(&db);
      up(&mutex);
      useData();
}
```

```
// Ghi dữ liệu
void writer() {
      while (TRUE) {
            thinkupData();
            down(&db);
            writeData();
            up(&db);
      }
}
```

3. **Đồng bộ luồng sử dụng Monitor:**
   - Monitor sinh ra từ NLHĐH và lập trình tương tranh (không bao giờ có 2 đoạn mã cạnh tranh cùng chạy)
   - VD cho bài toán gửi và rút tiền

```
class GuiRutTien {
      int account;
      public synchronized void withdraw {
            int x;
            x = account;
            x -= account;
            account = x;
      }

      public synchronized void deposit {
            int y;
            y += account;
            account = y;
      }
}
```

   - ***condition_var:***
     - **Receiver priority**: luồng gửi đánh thức ngay lập tức ngưng thực hiền và giao CPU cho luồng End
     - **Sender priority**: luồng gửi vẫn tiếp tục gửi CPU để thực hiện thoát khỏi cạnh tranh

4. **Giải pháp TSL: (phần cứng giống Peterson)**
   - TSL register, flag (0 nếu rỗi, 1 nếu bận)
     - register = flag
     - flag = 1

=> Nhận xét:

- Tách mã cạnh tranh khỏi tiến trình
- Ưu tiên giống Peterson
- Không giới hạn số lượng tiến trình
- Lãng phí CPU

CHƯƠNG 2: QUẢN LÝ BỘ NHỚ

1. **Các phương pháp đánh địa chỉ bộ nhớ:**
   - Phương pháp đánh địa chỉ tuyến tính: đánh địa chỉ cho trường byte lần lượt từ 0 đến n – 1 (biết n bytes = $\lceil \log_2 n \rceil$ bit)
   - Phương pháp đánh địa chỉ phân đoạn:
     - Segment => địa chỉ tuyệt đối → (đoạn)
     - Offset => địa chỉ tương đối → (thành phần)
   - Phương pháp quản lý bằng bản đồ bit: bộ nhớ chia thành các trang 4KB làm đơn vị cấp phát
   - Phương pháp quản lý bằng khối kích thước chẵn bội mũ 2: một khối có kích cỡ chẵn bội mũ 2 gọi là 1 buddy

   Địa chỉ tuyến tính tuyệt đối = segment x 16_b + offset = $segment_H$ x 10_H + offset

   Chuyển đổi địa chỉ ảo sang thực:
   Địa chỉ nhớ = địa chỉ trong + địa chỉ thành phần trong trang

2. **Các chiến lược cấp phát:**
   - First-fit → tìm từ đầu để lấy vùng nhớ có kích cỡ đủ để cấp phát cho tiến trình
   - Next-fit → tìm từ vị trí tiếp theo cho tới khi gặp vùng nhớ có kích cỡ đủ cấp phát
   - Best-fit → tìm vùng nhớ có kích thước nhỏ nhất đủ cấp cho tiến trình
   - Worst-fit → tìm vùng nhớ có kích cỡ to nhất để cắt ra cấp phát cho tiến trình

   => Nhận xét:
   - first-fit và next-fit nhanh hơn
   - best-fit → phần nhớ rỗi còn lại quá nhỏ để cấp phát
   - worst-fit → luôn cắt vùng nhớ lớn nhất ra để cấp phát

3. **Các thuật toán:**
   a) NRU – Not Recently Used:
   - Bit R – Referenced → bật khi trang nhớ dùng tới
   - Bit M – Modified → bật khi trang nhớ có nội dung bị đổi

	R	M
1	0	0
2	0	1
3	1	0
4	1	1

   => Tác dụng: việc hoán đổi dựa theo thứ tự nhãn, mỗi nhãn chọn ngẫu nhiên 1 trang để loại

=> Nhận xét:

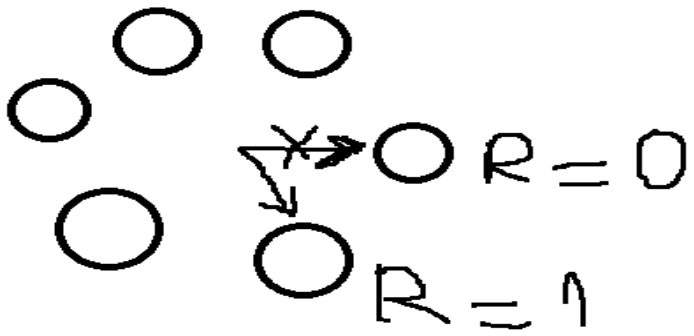
- Chỉ dựa 2 bit thuộc tính
- Trong mỗi nhãn chọn ngẫu nhiên

b) FIFIO – First In First Out:

- Các trang nhớ được sắp xếp theo thời điểm tải vào, trang cũ nhất được chọn để hoán đổi
- Nhận xét: những trang tải vào lớn nhất có thể dùng thường xuyên

c) Phương pháp Clock:

- Các trang nhớ sắp xếp thành vòng tròn theo thứ tự địa chỉ trang. Ban đầu kim chỉ số trỏ tới trang tải đầu tiên
  - Bit R = 0 → loại trang nhớ
  - Bit R = 1 → xóa bit R, dịch tới trang tiếp theo



- Nhận xét:
  - Không phải duy trì
  - Có thể loại trang dùng thường xuyên nhưng không liên tục

d) Phương pháp Second – Chance:

- Các trang gắn thêm 1 bit thuộc tính R (đặt = 1 khi R dùng tới)
- Nhận xét:
  - Đã sử dụng bit R để tránh loại trang dùng tới
  - Các trang dùng thường xuyên vẫn loại
  - Cần duy trì và quản lý các trang nhớ theo danh sách móc nối

e) Thuật toán LRU dùng phần cứng:

- Least Recently Used là tiêu chí để chọn trang hoán đổi
  => Mục đích → những trang ít dùng sẽ là những trang có xác suất thấp dùng trong bước tiếp theo
  - C1: Duy trì 1 thanh 64b đếm thời gian hiện tại t:

        - Mỗi trang gắn kèm 1 nhãn thời gian gần nhất được sử dụng tới
          → Khi trang nhớ I được sử dụng tại thời điểm t thì nhãn thời gian của trang nhớ sẽ được cập nhật là t
          → Khi Page Fault, trang có nhãn thời gian min (lâu không dùng) sẽ bị hoán đổi

    - C2: Duy trì ma trận cỡ nxn:
        - Trang nhớ i được dùng tới, ma trận được cập nhật:
          + Bật các bit thuộc hàng i
          + Tắt các bit thuộc cột i

f) Thuật toán Aging dùng phần mềm:
   → mô phỏng LRU bằng phần mềm
   - Mỗi trang nhớ gắn kèm 1 thanh ghi l bits lưu trạng thái bit R của l nhịp đồng hồ gần nhất
   - Khởi tạo tất cả các thanh ghi được gán = 0
   - Sau mỗi nhịp đồng hồ, các thanh ghi được dịch phải 1 bit và bit trái nhất được điền vào là bit R của trang nhớ tương ứng
   - Tại thời điểm Page Fault, trang có giá trị thanh ghi min là trang bị loại

   → Nhận xét:

   - Cho phép nhớ lại lịch sử l bit trước đó
   - Phải duy trì 1 thanh ghi cho mỗi trang nhớ
   - Chi phí thực hiện cao

g) Thuật toán WS-Clock:
   - Duy trì khoảng thời gian T cho các tiến trình sử dụng gần nhất trang khoảng thời gian ≤ T chưa loại (mỗi trang nhớ gắn kèm 2 bit thuộc tính R và M)
   - Nhận xét:
     - Có chi phí thực hiện thấp (khả thi = phần mềm)
     - Chỉ cần duy trì nhãn thời gian cho trang nhớ

CHƯƠNG 3: HỆ THỐNG TỆP VÀ BẾ TẮC

1. **Lỗi tình trạng khối đĩa:**
   - Lỗi vừa bận vừa rỗi => A[0,i] + A[1,i] = 1 (xóa khỏi danh sách móc nối các khối rỗi)
   - Lỗi bận nhiều lần => A[0,i] > 1 (thuê thêm khối rỗi, sao lặp nội dung ra khối rỗi mới cho mỗi tệp trỏ tới một khối đĩa riêng)
   - Lỗi rỗi nhiều lần => A[1,i] > 1 (xóa bớt trùng lặp khỏi danh sách các khối rỗi)
   - Lỗi không bận không rỗi => A[0,i] + A[1,i] = 0 (bổ sung vào danh sách các khối rỗi)

2. **Cài đặt liên kết cứng:**
   - Phương án 1: (hai tệp chung một i-node)
     - Tạo:
       - B1: Tạo và bổ sung bản ghi tệp
       - B2: Móc nối con trỏ đến i – node
       - B3: Tăng số Linkcount đi 1 đơn vị
     - Xóa:
       - B1: Giảm số Linkcount đi 1 đơn vị
       - B2: Nếu Linkcount = 0 thì giải phóng khối đĩa chứa nội dung tệp và giải phóng i-node
       - B3: Xóa bản ghi tệp
   - Phương án 2: (hai tệp sử dụng 2 i-node khác nhưng trỏ đến cùng 1 khối đĩa)
     - Xóa tệp:
       - B1: Giải phóng i-node
       - B2: Quan sát toàn bộ i-node xem còn i-node nào trỏ tới các khối đĩa chưa ND không (nếu không thì giải phóng)
       - B3: Xóa bản ghi khối tệp thư mục

3. **Bế tắc: (DEADLOCK)**
   - Điều kiện bế tắc:
     - ĐK loại trừ lẫn nhau
     - ĐK giữ và chờ
     - ĐK chờ thành chu trình
     - ĐK không có quyền ưu tiên
   - Deadlock là trạng thái tiến trình giữ tài nguyên và chờ tài nguyên từ tiến trình khác nhưng không có tiến trình nào đủ tài nguyên để kết thúc → cạn kiệt tài nguyên