**MÔN HỌC: HỆ ĐIỀU HÀNH**

**CÂU HỎI VÀ BÀI TẬP CHƯƠNG 5**

1. Khi nào thì xảy ra tranh chấp?

Race condition là tình trạng nhiều process truy xuất và thao tác đồng thời lên dữ liệu chia sẻ. Kết quả cuối cùng của việc truy xuất đồng thời này phụ thuộc thứ tự thực thi của các lệnh thao tác dữ liệu.

1. Vấn đề vùng tranh chấp (critical section) là gì?

Đầu tiên, chúng ta hãy xét một hệ thống có n tiến trình (tạm đặt tên của n tiến trình này là {P0, P1, ..., Pn−1}). Từng tiến trình đều có một đoạn mã, gọi là **critical section (CS),** tên tiếng Việt là **vùng tranh chấp.** Trong CS, các đoạn mã thao tác lên dữ liệu chia sẻ giữa các tiến trình.

Một đặc tính quan trọng mà chúng ta cần quan tâm, đó chính là khi process P0 đang chạy đoạn mã bên trong CS thì không một process nào khác được chạy đoạn mã bên trong CS (để đảm bảo cho dữ liệu được nhất quán). Hay nói cách khác là 1 CS trong một thời điểm nhất định, chỉ có 1 process được phép chạy.

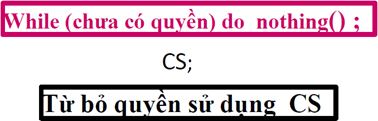
Và **vấn đề vùng tranh chấp (Critical Section Problem)** là vấn đề về việc tìm một cách thiết kế một giao thức (một cách thức) nào đó để các process có thể phối hợp với nhau hoàn thành nhiệm vụ của nó.

1. Có những yêu cầu nào dành cho lời giải của bài toán vùng tranh chấp?

Có 2 nhóm giải pháp chính :

Nhóm giải pháp Busy Waiting :

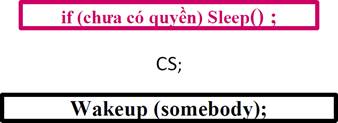
* Tính chất :
  + Tiếp tục tiêu thụ CPU trong khi chờ đợi vào miền găng (thông qua việc kiểm tra điều kiện vào CS liên tục).
  + Không đòi hỏi sự trợ giúp của hệ điều hành.
* Cơ chế chung :



* Bao gồm một vài loại :
  + Sử dụng các biến cờ hiệu.
  + Sử dụng việc kiểm tra luân phiên.
  + Giải pháp của Peterson.
  + Cấm ngắt (giải pháp phần cứng – hardware).
  + Chỉ thị TSL (giải pháp phần cứng – hardware).

Nhóm giải pháp Sleep & Wakeup.

* Tính chất :
  + Từ bỏ CPU khi chưa được vào CS.
  + Cần sự hỗ trợ từ hệ điều hành (để đánh thức process và đưa process vào trạng thái blocked).
* Cơ chế chung :



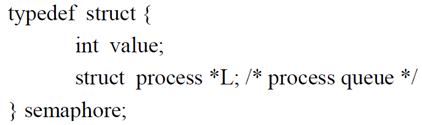
Cơ chế chung của nhóm giải pháp Sleep & Wakeup.

* Bao gồm một vài loại :
  + Semaphore.
  + Monitor.
  + Message.

1. Có mấy loại giải pháp đồng bộ? Kể tên và trình bày đặc điểm của các loại giải pháp đó?
2. Phân tích và đánh giá ưu, nhược điểm của các giải pháp đồng bộ busy waiting (cả phần cứng và phần mềm)?
3. Semaphore là gì? Đặc điểm của semaphore? Cách thức hiện thực semaphore? Có mấy loại semaphore? Khi sử dụng semaphore cần lưu ý những vấn đề gì?

Semaphore là một công cụ đồng bộ cung cấp bởi OS mà không đòi hỏi Busy Waiting.

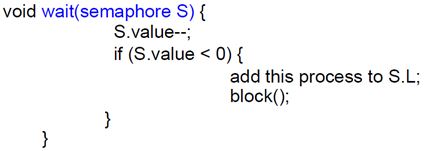
Semaphore là một số nguyên. Có cấu trúc như sau :



Cấu trúc của Semaphore.

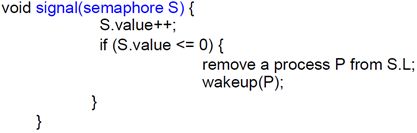
Giả sử ta đang có một Semaphore S. Có 3 thao tác có thể thực thi trên S :

* Khởi tạo semaphore. Giá trị khởi tạo ban đầu của Semaphore chính là số lượng process được thực hiện CS trong cùng 1 thời điểm.
* Wait(S) hay còn gọi là P(S) : Giảm giá trị semaphore đi một đơn vị (S = S – 1). Nếu giá trị S âm, process thực hiện lệnh wait() này sẽ bị blocked cho đến khi được đánh thức.



Nội dung hàm wait(S).

* Signal(S) hay còn gọi là V(S) : Tăng giá trị semaphore (S = S + 1). Kế đó nếu giá trị S <= 0 (S<=0 tức là vẫn còn process đang bị blocked), lấy một process Q nào đó đang bị blocked rồi gọi wakeup(Q) để đánh thức process Q đó.

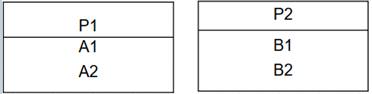


Nội dung hàm signal(S).

Tóm lại :

* P(S) hay wait(S) sử dụng để giành tài nguyên và giảm biến đếm S = S – 1.
* V(S) hay signal(S) sẽ giải phóng tài nguyên và tăng biến đếm S = S + 1.
* Nếu P được thực hiện trên biến đếm <= 0, tiến trình phải đợi V hay chờ đợi sự giải phóng tài nguyên.

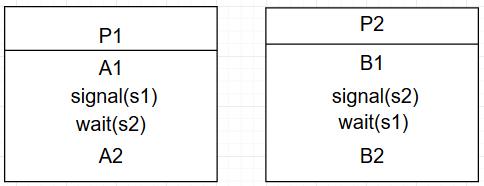
Ví dụ : Xét 2 tiến trình xử lý đoạn chương trình sau :



Tiến trình P1 và tiến trình P2.

Ta cần đồng bộ hoá hoạt động của 2 tiến trình sao cho cả A1 và B1 sao cho cả A1 và B1 đều hoàn tất trước khi A2 và B2 bắt đầu.

Để đồng bộ hoạt động theo yêu cầu, ta phải định nghĩa P1 và P2 như sau :



Định nghĩa lại tiến trình P1 và P2.

Giải thích cơ chế hoạt động :

Có 2TH có thể xảy ra :

* P1 chạy trước.
* P2 chạy trước.

Xét trường hợp P1 chạy trước.

Ở đây, ta hình dung signal chính là “chìa khoá” để mở khoá cho cánh cổng. Và wait chính là “cổng” chặn lại không cho process được thực thi tiếp.

* Ở P1, ta thấy signal được đặt sau A1, đồng nghĩa với việc sau khi thực hiện A1 xong, P1 sẽ “mở cổng” chặn của P2 và cho phép P2 thực hiện B2. Sau khi thực hiện signal, P1 tiếp tục bị chặn lại bởi hàm wait.
* Ở đây, ta thấy ở P2 cũng có signal nằm đằng sau B1. Vậy chỉ sau khi thực hiện B1, P1 mới được cho phép thực hiện tiếp A2.

Với 2 phân tích như trên, ta thấy thoả mãn việc đồng bộ hoạt động theo yêu cầu.

Xét tương tự với trường hợp P2 chạy trước.

1. Monitor và Critical Region là gì?
2. Đặc điểm và yêu cầu đồng bộ của các bài toán đồng bộ kinh điển?
3. (Bài tập mẫu) Xét giải pháp phần mềm do Dekker đề nghi để tổ chức truy xuất độc quyền cho 2 tiến trình. Hai tiến trình P0 và P1 chia sẻ các biến sau:

|  |
| --- |
| boolean flag[2]; /\* initially false \*/  int turn; |

Cấu trúc một tiến trình Pi (với i = 0 hay 1 và j là tiến trình còn lại) như sau:

|  |
| --- |
| while (true) {  flag[i] = true;  while (flag[j]) {  if (turn == j) {  flag[i] = false;  while (turn == j)  ; /\* do nothing \*/  flag[i] = true;  }  }  /\* critical section \*/  turn = j;  flag[i] = false;  /\* remainder section \*/  } |

Giải pháp này có thỏa 3 yêu cầu trong viêc giải quyết tranh chấp không?

Trả lời:  
Giải pháp này thỏa 3 yêu cầu trong giải quyết tranh chấp vì:

- Loại trừ tương hỗ: Tiến trình Pi chỉ có thể vào vùng tranh chấp khi flag[j] = false. Giả sử P0 đang ở trong vùng tranh chấp, tức là flag[0] = true và flag[1] = false. Khi đó P1 không thể vào vùng tranh chấp (do bị chặn bởi lệnh while (flag[j])). Tương tự cho tình huống P1 vào vùng tranh chấp trước.

- Progress: Giá trị của biến turn chỉ có thể thay đổi ở cuối vùng tranh chấp. Giả sử chỉ có 1 tiến trình Pi muốn vào vùng tranh chấp. Lúc này, flag[j] = false và tiến trình Pi sẽ được vào vùng tranh chấp ngay lập tức. Xét trường hợp cả 2 tiến trình đều muốn vào vùng tranh chấp và giá trị của turn đang là 0. Cả flag[0] và flag[1] đều bằng true. Khi đó, P0 sẽ được vào vùng tranh chấp, bởi tiến trình P1 sẽ thay đổi flag[1] = false (lệnh kiểm tra điều kiện if (turn == j) chỉ đúng với P1). Tương tự cho trường hợp turn = 1.

- Chờ đợi giới hạn: Pi chờ đợi lâu nhất là sau 1 lần Pj vào vùng tranh chấp (flag[j] = false sau khi Pj ra khỏi vùng tranh chấp). Tương tự cho trường hợp Pj chờ Pi.

1. Xét giải pháp đồng bộ hóa sau:

|  |
| --- |
| while (TRUE) {  int j = 1-i;  flag[i]= TRUE;  turn = i;  while (turn == j && flag[j]==TRUE);  critical-section ();  flag[i] = FALSE;  Noncritical-section ();  } |

Giải pháp này có thỏa yêu cầu độc quyền truy xuất không?

1. Giả sử một máy tính không có chỉ thị TSL, nhưng có chỉ thị Swap có khả năng hoán đổi nội dung của hai từ nhớ chỉ bằng một thao tác không thể phân chia:

|  |
| --- |
| procedure Swap(var a,b: boolean){  var temp : boolean;  begin  temp := a;  a:= b;  b:= temp;  end;  } |

Sử dụng chỉ thị này có thể tổ chức truy xuất độc quyền không? Nếu có, xây dựng cấu trúc chương trình tương ứng.

1. Xét hai tiến trình sau:

process A {while (TRUE) na = na +1;}

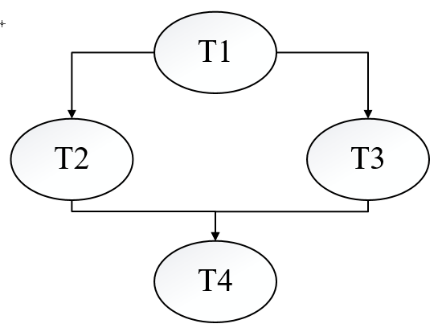
process B {while (TRUE) nb = nb +1;}

a. Đồng bộ hóa xử lý của 2 tiến trình trên, sử dụng 2 semaphore tổng quát, sao cho tại bất kỳ thời điểm nào cũng có nb <= na <= nb +10.

b. Nếu giảm điều kiện chỉ còn là na <= nb +10, cần sửa chữa giải pháp trên như thế nào?

c. Giải pháp trên còn đúng nếu có nhiều tiến trình loại A và B cùng thực hiện?

1. (Bài tập mẫu) Xét một hệ thống có 4 tiểu trình T1, T2, T3, T4. Quan hệ giữa các tiểu trình này được biểu diễn như sơ đồ bên dưới, với mũi tên từ tiểu trình (Tx) sang tiểu trình (Ty) có nghĩa là tiểu trình Tx phải kết thúc quá trình hoạt động của nó trước khi tiểu trình Ty bắt đầu thực thi. Giả sử tất cả các tiểu trình đã được khởi tạo và sẵn sàng để thực thi. Hãy sử dụng semaphore để đồng bộ hoạt động của các tiểu trình sao cho đúng với sơ đồ đã cho.



Trả lời:

Khai báo và khởi tạo các semaphore:

init(sem1,0); //khởi tạo semaphore sem1 có giá trị bằng 0

init(sem2,0); //khởi tạo semaphore sem2 có giá trị bằng 0

void T4(void)   
{

wait(sem2)

wait(sem2)

//T4 thực thi

}

void T3(void)   
{

wait(sem1)

//T3 thực thi

signal(sem2)

}

void T2(void)   
{

wait(sem1)

//T2 thực thi

signal(sem2)

}

void T1(void)   
{

//T1 thực thi

signal(sem1)

signal(sem1)

}

1. Một biến X được chia sẻ bởi 2 tiến trình cùng thưc hiện đoạn code sau:

|  |
| --- |
| do  X = X + 1;  if (X == 20) X = 0;  while (TRUE); |

Bắt đầu với giá trị X = 0, chứng tỏ rằng giá tri X có thể vượt quá 20. Cần sửa chữa đoạn chương trình trên như thế nào để đảm bảo X không vượt quá 20?

1. Xét 2 tiến trình xử lý đoạn chương trình sau:

process P1 {A1 ; A2 }

process P2 {B1 ; B2 }

Đồng bộ hóa hoạt động của 2 tiến trình này sao cho cả A1 và B1 đều hoàn tất trươc khi A2 và B2 bắt đầu.

1. Tổng quát hóa bài tập 14 cho các tiến trình có đoạn chương trình sau:

process P1 { for ( i = 1; i <= 100; i ++) Ai }

process P2 { for ( j = 1; j <= 100; j ++) Bj }

Đồng bộ hóa hoạt động của 2 tiến trình này sao cho với k bất kỳ (2<=k<=100), Ak chỉ có thể bắt đầu khi B(k-1) đã kết thúc và Bk chỉ có thể bắt đầu khi A(k-1) đã kết thúc.

1. Sử dụng semaphore để viết lại chương trình sau theo mô hình xử lý đồng hành:

w := x1 \* x2

v := x3 \* x4

y := v \* x5

z := v \* x6

x := w \* y

z := w \* z

ans := y + z