Chương 5

Race condition:

* Bài toán cấp phát PID: Nếu có cả 2 tiến trình cùng fork -> kernel sẽ phải quyết định PID nào sẵn sàng để cấp cho tiến trình nào.
* Race condition xảy ra khi các tiến trình cùng truy cập vào dữ liệu được chia sẻ. Kết quả cuối cùng phụ thuộc vào thứ tự thực hiện tiến trình.
* Race condition gây mất, sai, thiếu nhất quán dữ liệu. (inconsistency)
* Cần đảm bảo 1 thời điểm chỉ có 1 tiến trình thao tác lên dữ liệu chia sẻ.

Vùng tranh chấp:

Xét 1 hệ thống có n tiến trình P1->Pn

Mỗi tiến trình có vùng tranh chấp (critical section) là đoạn code

* Thực hiện thay đổi dữ liệu được chia sẻ.
* Khi tiến trình đang thực hiện thì tiến trình khác không thực hiện vùng tranh chấp của nó.

Vấn đề vùng tranh chấp là thiết kế cách xử lí vấn đề trên.

A computer code with text

Description automatically generated

Các bước của tiến trình:

* Yêu cầu được vào vùng tranh chấp (entry section)
* Thực hiện vùng tranh chấp (critical section)
* Thoát khỏi vùng tranh chấp (exit section)
* Thực thi những phần còn lại (remainder)

Yêu cầu cho lời giải:

* Loại trừ tương hỗ (Mutual exclusion): Khi P thực thi trong CS thì không Q nào đang thực thi trong CS của Q. (?)
* Tiến triển (Progress): 1 tiến trình dừng bên ngoài vùng tranh chấp thì để tiến trình khác vào. Nếu P đợi Q và Q cũng đợi P -> deadlock.
* Giới hạn chờ đợi (Bounded waiting): Mỗi Process chỉ phải chờ 1 thời gian nhất định để vào vùng tranh chấp -> k đói tài nguyên (starvation)

Giải pháp Peterson:

* Giải pháp cho 2 tiến trình.
* Giả sử 2 lệnh asm load và store đơn nguyên (không thể bị ngắt)
* 2 tiến trình cùng chia sẻ 2 biến:

Int turn;

Boolean flag[2];

* Biến turn để quyết định lượt tiến trình nào vào vùng tranh chấp.
* Mảng Flag để xác định tiến trình sẵn sàng vào vùng tranh chấp chưa, true = đã sẵn sàng.

Mutual exclution đảm bảo

Pi chỉ được phép vào khi flag[j] = false hoặc turn = i

Turn không thể vừa bằng i vừa bằng j ( nếu i = 0, j = 1 thì turn k thể vừa bằng 0 vừa bằng 1).

Progress đảm bảo

P1 không được vào CS -> flag[0] và turn = 0

P0 không vào CS

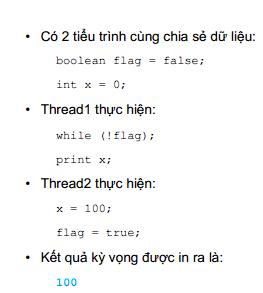
Turn = 1 & flag[0] = false

P0 không vào CS -> không cản P1 vào CS và ngược lại.

Bounded Waiting đảm bảo

P1 chờ tối đa 1 lần P0 vào vùng tranh chấp, CS nhỏ -> thời gian đợi ngắn.

Để cải thiện hiệu suất, các thao tác độc lập được sắp xếp lại -> các tiến trình đa tiểu trình bị sai.



Nếu bị xếp lại flag trước x thì kq sẽ ra khác kì vọng (0).

A black rectangle with black text

Description automatically generated

Giải pháp Peterson nó bị ntn.

Việc gán turn và flag sắp xếp lại thứ tự -> sai.

Memory Barrier

Memory Model trong hệ điều hành hoạt động như bộ nhớ của hệ thống. Memory barrier là 1 instruction bắt buộc mọi thay đổi trong bộ nhớ phải được truyền tải để tất cả bộ xử lí khác.

Nó đảm bảo mọi thao tác load hoặc store hoàn thành trước khi load hoặc store khác thực hiện

Do đó, kể cả khi lệnh bị xếp lại thì các thao tác ghi dữ liệu đều đã được hoàn thành trong bộ nhớ và được truyền đến các bộ xử lí khác trước khi thao tác nạp ghi khác thực thi.

Mutex Lock không busy waiting

Để tránh busy waiting trong mutex locks, ta đặt tiến trình vào trạng thái ngủ khi khoá và đánh tức khi khoá được mở. HĐH cung cấp 2 thao tác block và wakeup cho việc đó.

Cách dùng:

* Khai báo 1 mutex ở hàm main
* Xác định vùng tranh chấp, acquire ở trước vùng tranh chấp và release sau vùng tranh chấp.

Semaphore

Semaphore linh hoạt hơn để các tiến trình đồng bộ hoạt động.

Bản chất là 1 biến số nguyên, cập nhật qua 2 thao tác wait() và signal() hoặc P() và V()

Wait bắt tiến trình chờ khi semaphore s không dương, ngược lại s- -

Signal trả tài nguyên, s++

Phân loại

Không giới hạn -> counting semaphore

0 hoặc 1 -> binary semaphore -> khoá mutex

Các vấn đề của semaphore

Bản thân wait và signal cũng có vùng tranh chấp -> giải quyết bằng các giải pháp khác.

Lưu ý khởi tại và thứ tự xếp lệnh để không xảy ra deadlock.

Bài toán Bounded Buffer

Phát biểu:

Producer: thêm phần tử vào mảng có n phần tử

Consumer: xoá phần tử ra khỏi mảng.

* Vùng tranh chấp: mảng buffer và biến count

Số lượng semaphore:

* Semaphore empty: n
* Semaphore full: 0
* Semaphore mutex: 1

Bài toán Readers – Writers

Phát biểu:

Reader thực hiện đọc, không thay đổi.

Writer thực hiện vừa đọc vừa ghi.

Cho phép Reader đồng thời và 1 Writer tai 1 thời điểm