**Vấn đề thảo luận**

Vấn đề 1: Nhóm 5

Nhờ có thực hiện 1 cách không thể phân chia, semaphore đã giải quyết được vấn đề tín hiệu “đánh thức” bị thất lạc. Điều gì sẽ xảy ra xảy ra khi chúng ta quên gọi hàm Up(s) và khi chúng ta vô tình đặt các primitive Down và Up sai vị trí.

VD: while(True){

Down(s)

Critical – section ();

Up(s)

Noncritical – section ();

}

Vấn đề 2: Nhóm 3

Vấn đề ẩm thực - triết gia

Vấn đề: 5 nhà triết học cùng ngồi ăn tối với món mỳ Ý nổi tiếng. Mỗi nhà triết học cần dùng 2 cái nĩa để có thể ăn mỳ Ý . Nhưng trên bàn chỉ có tổng cộng 5 cái nĩa để xen kẽ với 5 cái đĩa. Mỗi nhà triết học sẽ suy ngẫm các triết lý của mình đến khi cảm thấy đói thì dự định lần lượt cầm 1 cái nĩa bên trái và 1 cái nĩa bên phải để ăn. Nếu cả 5 nhà triết học đều cầm cái nĩa bên trái cùng lúc, thì sẽ không có ai có được cái nĩa bên phải để có thể bắt đầu thưởng thức mỳ Ý . Đây chính là tình trạng tắc nghẽn.

Vấn đề 3: Nhóm 4

Khi mà chúng ta thiết kế các luồng trong hệ điều hành, chúng ta gặp phải những vấn đề gì và có hướng giải quyết như thế nào?

Vấn đề 4: Nhóm 1

Như ta đã biết IPC là một cơ chế cho phép một tiến trình giao tiếp với một tiến trình khác, thông thường là trong cùng một hệ thống. Và trong bài ta đã tìm hiểu về phương pháp shared memory, ngoài nó ra thì ta còn các phương pháp nào khác để thực hiện giao tiếp liên tiến trình không? Nếu có thì hãy trình bày các phương pháp dó?

Vấn đề 5: Nhóm 6

TÌm hiểu về luồng? Thế nào là luồng cấp người dùng và thế nào là luồng cấp nhân?

**Giải pháp mà các nhóm đưa ra**

***Vấn đề 1:***

- Khi đặt sai vị trí các primitive Down và Up thì tiến trình sẽ bị khóa vĩnh viễn và không thoát ra được.

- Khi quên gọi hàm Up(s) thì kết quả là khi ra gọi miền găng nó sẽ không cho tiến trình khác vào miền găng

***Vấn đề 2: Nhóm 3***

- Từ đề bài ta có thể thấy rằng, nếu cả 5 nhà triết học đề ăn cùng một lúc thì sẽ không thể có đủ nĩa cho bất kì người nào có thể ăn. Khi đó cả 5 nhà triết học sẽ gặp phải tình trạng tác nghẽn. Để ngăn chặn tình huống này xả ra, 5 nhà triết học phải thảo luận với nhau để chọn ra những ai là người sẽ ăn trước và những ai là người sẽ ăn sau. Ta có thể nhận thấy rằng chúng ta có tổng cộng 5 cái nĩa vì thế tối đa cùng một lúc, chỉ có 2 nhà triết học có thể thưởng thức bữa ăn của mình. Chính vì thế, giải pháp có thể đưa ra ở đây là chọn ra 2 người sẽ ăn đầu tiên, khi 2 người đầu tiên ăn xong thì đến 2 người tiếp theo, và sau đó người cuối cùng sẽ thưởng thức bữa ăn của mình. Khi đó chúng ta có thể ngăn chạn tình trạng tắc nghẽn xảy ra.

***Vấn đề 3:***

Lời giải:

* **Tăng độ phức tạp** - Các quy trình đa luồng khá phức tạp. Mã hóa cho những điều này chỉ có thể được xử lý bởi các lập trình viên chuyên nghiệp.
* **Các biến chứng do đồng thời** - Rất khó để xử lý đồng thời trong các quy trình đa luồng. Điều này có thể dẫn đến các biến chứng và các vấn đề trong tương lai.
* **Khó xác định lỗi** - Việc xác định và sửa lỗi khó khăn hơn nhiều trong quy trình đa luồng so với quy trình đơn luồng.
* **Kiểm tra các biến chứng** - Kiểm tra là một quá trình phức tạp i các chương trình đa luồng so với các chương trình đơn luồng. Điều này là do các khiếm khuyết có thể liên quan đến thời gian và không dễ xác định.
* **Kết quả không thể đoán trước** - Các chương trình đa luồng đôi khi có thể dẫn đến kết quả không thể đoán trước được vì về cơ bản chúng là nhiều phần của một chương trình đang chạy cùng một lúc.
* **Các biến chứng cho** việc chuyển mã hiện có - Cần rất nhiều thử nghiệm để chuyển mã hiện có trong đa luồng. Các biến static cần được loại bỏ và bất kỳ lệnh gọi mã hoặc hàm nào không an toàn cho luồng cần phải được thay thế.

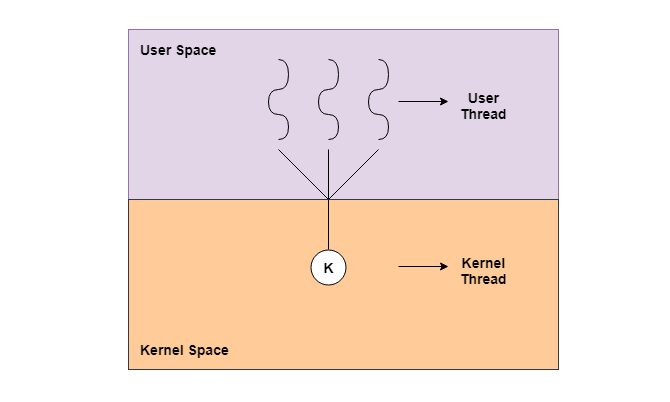
***Vấn đề 4: - Nhóm 1***

Sau đây là một số phương pháp phổ biến để thực hiện IPC:

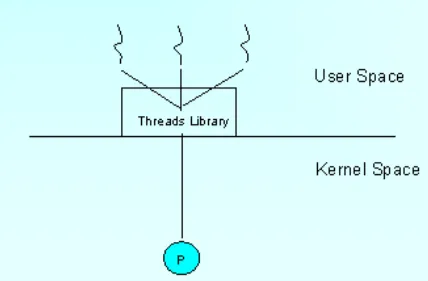
* Pipes – Giao tiếp giữa hai tiến trình liên quan. IPC sử dụng Pipe là phương pháp bán song công (half duplex)- nghĩa là chỉ giao tiếp theo một phía: tiến trình thứ nhất sẽ giao tiếp với tiến trình thứ hai. Để tiến trình thứ hai có thể giao tiếp ngược lại với tiến trình thứ nhất (song công), chúng ta phải sử dụng thêm một Pipe khác
* Shared files (FIFO) giao tiếp giữa hai tiến trình riêng biệt. Giao tiếp sử dụng FIFO là song công, tiến trình thứ nhất có thể giao tiếp với tiếng trình thứ hai vào ngược lại
* Message queues – giao tiếp giữa nhiều tiến trình song công. Các tiến trình sẽ giao tiếp với nhau bằng cách gửi một bản tin và nhận nó thông qua hàng đợi (queue). Bản tin đã đc nhận bới một tiếng trình sẽ bị xóa khỏi hàng đơi.
* Signal – tiến trình nguồn sẽ gửi một signal (phân biệt bằng số thứ tự định sẵn) để tiến trình đích tương ứng sẽ xử lý.

***Vấn đề 5: Nhóm 6***

**Thế nào là luồng:** Luồng CPU là tập hợp các trình điều khiển và truyền dữ liệu xử lý của CPU. Do vậy, một CPU có nhiều luồng sẽ phát huy tối ưu năng lực xử lý và cho phép CPU đó hoạt động đa tác vụ hơn trong cùng thời điểm. Điều này giúp tối ưu tài nguyên và rút ngắn thời gian chờ đợi của người dùng.



**Luồng cấp độ người dùng:** Các luồng cấp người dùng được thực hiện bởi người dùng và hạt nhân không biết về sự tồn tại của các luồng này. Nó xử lý chúng như thể chúng là các quy trình đơn luồng. Các luồng cấp người dùng nhỏ và nhanh hơn nhiều so với các luồng cấp nhân. Chúng được đại diện bởi một bộ đếm chương trình (PC), ngăn xếp, các thanh ghi và một khối điều khiển quy trình nhỏ. Ngoài ra, không có sự tham gia của hạt nhân trong đồng bộ hóa cho các luồng cấp người dùng.



**Ưu điểm của chủ đề cấp độ người dùng**

Một số ưu điểm của chủ đề cấp người dùng như sau:

* Các luồng cấp người dùng được tạo dễ dàng và nhanh hơn so với các luồng cấp nhân. Chúng cũng có thể được quản lý dễ dàng hơn.
* Chủ đề cấp độ người dùng có thể được chạy trên bất kỳ hệ điều hành nào.
* Không có đặc quyền chế độ hạt nhân cần thiết để chuyển đổi luồng trong các luồng cấp người dùng.

**Nhược điểm của Chủ đề cấp độ người dùng**

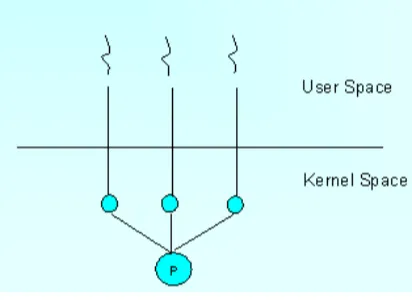
Một số nhược điểm của luồng cấp người dùng như sau:

* Các ứng dụng đa luồng trong luồng cấp người dùng không thể sử dụng đa xử lý để làm lợi thế của chúng.
* Toàn bộ quá trình bị chặn nếu một luồng cấp người dùng thực hiện thao tác chặn.

**Ví dụ** : POSIX Pthreads và Mach C-thread

* Luồng cấp người dùng sử dụng không gian người dùng để lập lịch luồng. Các luồng này là  **minh bạch**  đối với hệ điều hành.
* Các luồng cấp độ người dùng được tạo bởi các thư viện thời gian chạy không thể thực thi các hướng dẫn đặc quyền.
* Các chủ đề cấp độ người dùng có chi phí thấp nhưng nó có thể đạt được hiệu suất cao trong tính toán.
* Chủ đề cấp người dùng được quản lý hoàn toàn bởi hệ thống thời gian chạy.
* Chủ đề cấp độ người dùng đã ăn nhỏ thêm nhanh hơn. Một luồng được biểu diễn đơn giản bởi một PC, các thanh ghi, ngăn xếp và khối điều khiển luồng nhỏ.
* Mã để tạo và hủy luồng, truyền thông báo và truyền dữ liệu, lập lịch luồng được đưa vào thư viện luồng. Kernel không biết về luồng cấp người dùng.
* Các luồng cấp độ người dùng không gọi Kernel cho quyết định lập lịch trình.
* Luồng mức người dùng còn được gọi là  **nhiều luồng ánh xạ**  vì hệ điều hành ánh xạ tất cả các luồng trong một quy trình đa luồng vào một ngữ cảnh thực thi duy nhất. Hệ điều hành coi mỗi quy trình đa luồng là một đơn vị thực thi duy nhất.
* **Ví dụ** : POSIX Pthreads và Mach C-thread

**Luồng cấp nhân:** Các luồng cấp nhân được hệ điều hành xử lý trực tiếp và việc quản lý luồng được thực hiện bởi nhân. Thông tin ngữ cảnh cho tiến trình cũng như các luồng tiến trình đều được quản lý bởi hạt nhân. Do đó, các luồng cấp nhân chậm hơn các luồng cấp người dùng.



**Ưu điểm của luồng cấp nhân**

Một số ưu điểm của luồng cấp nhân như sau:

* Nhiều luồng của cùng một quá trình có thể được lập lịch trên các bộ xử lý khác nhau trong các luồng cấp hạt nhân.
* Các quy trình hạt nhân cũng có thể được đa luồng.
* Nếu một luồng cấp nhân bị chặn, thì nhân có thể lên lịch một luồng khác của cùng một tiến trình.

**Nhược điểm của luồng cấp nhân**

Một số nhược điểm của luồng cấp nhân như sau:

* Cần chuyển chế độ sang chế độ hạt nhân để chuyển quyền điều khiển từ luồng này sang luồng khác trong một quy trình.
* Các luồng cấp nhân chậm hơn để tạo cũng như quản lý so với các luồng cấp người dùng
* Ví dụ: Windows 95/99 / NT. Sun Solaris và Digital UNIX.
* Trong luồng cấp Kernel, quản lý luồng được thực hiện bởi Kernel. Hệ điều hành hỗ trợ luồng cấp Kernel.
* Vì Kernel quản lý luồng, nên Kernel có thể lên lịch cho một luồng khác nếu một luồng nhất định chặn thay vì chặn toàn bộ quá trình.
* Luồng mức hạt nhân hỗ trợ ánh xạ luồng một đến một. Ánh xạ này yêu cầu mỗi luồng người dùng với luồng nhân. Hệ điều hành thực hiện ánh xạ này.
* Các luồng được xây dựng và điều khiển bởi các lệnh gọi hệ thống. Hệ thống biết trạng thái của mỗi luồng.
* Mã quản lý luồng không được bao gồm trong mã ứng dụng. Nó là API duy nhất cho luồng Kernel. Hệ điều hành Windows sử dụng cơ sở này.
* Bất kỳ ứng dụng nào cũng có thể được lập trình để chạy đa luồng. Tất cả các chuỗi trong một ứng dụng được hỗ trợ trong một quy trình duy nhất.
* Kernel thực hiện lập lịch trên cơ sở luồng. Hỗ trợ và quản lý Kernel, chỉ tạo luồng trong không gian Kernel.
* Các luồng cấp nhân chậm hơn các luồng cấp người dùng.
* Ví dụ: Windows 95/99 / NT. Sun Solaris và Digital UNIX.

Một luồng người dùng là một chuỗi thực thi mã không gian người dùng. Nhưng nó có thể gọi vào không gian kernel bất cứ lúc nào. Nó vẫn được coi là một luồng "Người dùng", mặc dù nó đang thực thi mã hạt nhân ở mức bảo mật cao.

Ba mô hình sau cũng là ba cách để hình thành mối quan hệ giữa luồng người dùng và luồng nhân: mô hình Many-to-one, mô hình one-to-one, và mô hình one-to-many.

**Mô hình many-to-one**

**Mô hình many-to-one** là mô hình nhiều user threads nối vào một kernel thread. Việc quản lý các luồng này dựa vào thư viện luồng trong không gian người dùng. Tuy nhiên, cả hệ thống sẽ bị chặn nếu một luồng nào đó thực hiện một blocking system call. Và cũng vì chỉ một luồng được tiếp cận kernel trong một lần, các threads khác không thể chạy song song trong hệ thống đa lõi. Green threads – tên gọi của một thư viện luồng của hệ thống Solaris và đã được sử dụng trong những versions cũ của Java – sử dụng mô hình many-to-one. Không có quá nhiều hệ thống sử dụng mô hình này vì nó không sử dụng được lợi thế của multiple processing cores.

**Mô hình one-to-one**

**Mô hình one-to-one** là mô hình 1-1, một user thread kết nối với một kernel thread. Mô hình 1-1 này đảm bảo được tính liên tục vì nếu một thread bị block thì một thread khác sẽ vẫn kết nối được với kernel. Nó cũng đảm bảo được nhiều luồng có thể hoạt động cùng một lúc trong bộ đa xử lý. Khuyết điểm duy nhất của mô hình này là nó đòi hỏi khi một user thread hoạt động thì một kernel thread phải được kích hoạt theo. Và vì quá nhiều kernel threads sẽ gây nên sự quá tải trong ứng dụng, những app sử dụng model này đều giới hạn số lượng thread được tạo ra trong hệ thống. Linux và Windows là những hệ điều hành sử dụng model one-to-one.

**Mô hình many-to-many**

**Mô hình many-to-many** chia các user-level threads cho một lượng nhỏ hơn hoặc bằng các kernel threads. Lượng kernel threads này tùy thuộc vào yêu cầu của ứng dụng sử dụng hoặc bộ máy sử dụng (một ứng dung thường dùng nhiều kernel threads trên multiprocessors hơn là trên single processor). Mô hình này khác mô hình many-to-one ở tính liên lục. Trong mô hình many-to-one, người dùng có thể tạo bao nhiêu user threads tùy thích, nhưng nó không đảm bảo được tính liên tục vì một kernel chỉ kết nối được với một user thread, như đã nói ở trên. Mô hình one-to-one cho phép tính liên tục cao hơn, tuy nhiên số lượng threads được tạo ra rất quan trọng nếu bạn không muốn ứng dụng của mình bị quá tải.

Mô hình many-to-many sẽ giải quyết được vấn đề của hai mô hình trước: số lượng user threads tạo ra là tùy thích, và kernel thread tương ứng có thể chạy song song trong hệ đa xử lý. Và, khi một thread đang thực hiện blocking system call, kernel threads tương ứng có thể chuyển sang một user thread khác và giải quyết user thread đó. Một biến thể của many-to-many model là *two-level model*. Model này như là sự kết hợp giữa many-to-many model và one-to-one model, vì nó vừa chia các user-level threads cho một lượng nhỏ hơn hoặc bằng các kernel threads tương ứng, và vừa cho phép một user thread kết nối riêng với một kernel thread. Trước version Solaris 9, hệ điều hành Solaris đã sử dụng *two-level model.* Tuy nhiên, từ version 9 trở đi, Solaris sử dụng one-to-one model.