HỆ THỐNG THỜI GIAN THỰC
VÀ ỨNG DỤNG TRONG KỸ THUẬT MÔ PHỎNG

Nguyễn Văn Trường

Trung tâm Công nghệ Mô phỏng –HVKTQS

Abstract:

we all tend to throw the term *real-time* around quite a bit. There seems to be an innate understanding of what it means, but not a formal definition that is widely agreed upon. Whatever the case, it does not stop us from using it to impress our listeners. It has, after all, a bit of a *macho* ring to it. But the use of it such in diverse contexts often can be confusing. Read on for an attempt at an applicable definition.

Keywords: simulation, RTS, realtime system, 3D graphic

1. Đặt vấn đề

Trong những năm gần đây, các hệ thống điều khiển theo thời gian thực là một trong những lĩnh vực thu hút nhiều sự chú ý trong giới khoa học nghiên cứu về khoa học máy tính. Trong đó, vấn đề điều hành thời gian thực và vấn đề lập lịch là đặc biệt quan trọng. Một số ứng dụng quan trọng của hệ thống thời gian thực (RTS) đã và đang được ứng dụng rộng rãi hiện nay là các dây chuyền sản xuất tự động, rôbốt, điều khiển không lưu, điều khiển các thí nghiệm tự động, truyền thông, điều khiển trong quân sự... Bên cạnh đó các thiết bị mô phỏng được đưa vào với mục đích đào tạo, tạo sự thân thiện giữa mô hình với chính đối tượng trong thực tế, giúp người học có được sự hiểu biết về thiết bị cũng như kỹ năng thực hành trên thiết bị đó.

Hiện nay một số đơn vị trong quân đội được trang bị hệ thống mô phỏng huấn luyện lái. Điều này khẳng định giải pháp ứng dụng công nghệ mô phỏng để nâng cao chất lượng huấn luyện là hướng đi đúng đắn. Tuy nhiên để hệ thống mô phỏng ngày càng sát thực tế và sống động cần đòi hỏi nhiều công sức nghiên cứu lý thuyết, trong đó hiểu biết đầy đủ về hệ thống thời gian thực để xây dựng các thiết bị phục vụ huấn luyện và đào tạo. Qua đó tập cho người học cách đưa ra những phán quyết trong khoảng thời gian hợp lý với các tình huống thực tế.

Với kết qủa nghiên cứu và tích lũy, TTCN Mô phỏng xin trình bầy bài báo để cùng nhau hiểu rõ về hệ thống thời gian thực và áp dụng cho đồ họa 3D thời gian thực trong các sản phẩm mô phỏng.

2. Hệ thống thời gian thực

1.1. Khái niệm hệ thống thời gian thực :

Một hệ thống thời gian thực (RTS – Realtime Systems) có thể được hiểu như là một mô hình xử lý mà tính đúng đắn của hệ thống không chỉ phụ thuộc vào kết quả tính toán lôgic mà còn phụ thuộc vào thời gian mà kết quả này phát sinh ra.

Hệ thống thời gian thực được thiết kế nhằm cho phép trả lời lại các yếu tố kích thích phát sinh từ các thiết bị phần cứng trong một ràng buộc thời gian xác định. Ở đây ta có thể hiểu thế nào là một RTS bằng cách hiểu thế nào là một tiến trình, một công việc thời gian thực. Nhìn chung, trong những RTS chỉ có một số công việc được gọi là công việc thời gian thực, các công việc này có một mức độ khẩn cấp riêng phải hoàn tất, ví dụ một tiến trình đang cố gắng điều khiển hoặt giám sát một sự kiện đang xảy ra trong thế giới thực. Bởi vì mỗi sự kiện xuất hiện trong thế giới thực nên tiến trình giám sát sự kiện này phải xử lý theo kịp với những thây đổi của sự kiện này. Sự thay đổi của sự kiện trong thế giới thực xảy ra rất nhanh, mỗi tiến trình giám sát sự kiện này phải thực hiện việc xử lý trong một khoản thời gian ràng buộc gọi là deadline, khoản thời gian ràng buộc này được xác định bởi thời gian bắt đầu và thời gian hoàn tất công việc. Trong thực tế, các yếu tố kích thích xảy ra trong thời gian rất ngắn vào khoảng vài mili giây, thời gian mà hệ thống trả lời lại yếu tố kích thích đó tốt nhất vào khoảng dưới một giây, thường vào khoảng vài chục mili giây, khoảng thời gian này bao gồm thời gian tiếp nhận kích thích, xử lý thông tin và trả lời lại kích thích. Một yếu tố khác cần quan tâm trong RTS là những công việc thời gian thực này có tuần hoàn hay không ? Công việc tuần hoàn thì ràng buộc thời gian ấn định theo từng chu kỳ xác định. Công việc không tuần hoàn xảy ra với ràng buộc thời gian vào lúc bắt đầu và lúc kết thúc công việc, ràng buộc này chỉ được xác định vào lúc bắt đầu công việc. Các biến cố kích hoạt công việc không tuần hoàn thường dựa trên kỹ thuật xử lý ngắt của hệ thống phần cứng.

Về mặt cấu tạo, RTS thường được cấu thành từ các thành tố chính sau :

- Đồng hồ thời gian thực : Cung cấp thông tin thời gian thực.

- Bộ điều khiển ngắt : Quản lý các biến cố không theo chu kỳ.
- Bộ định biểu : Quản lý các qua trình thực hiện.
- Bộ quản lý tài nguyên : Cung cấp các tài nguyên máy tính.
- Bộ điều khiển thực hiện : Khởi động các tiến trình.

Các thành tố trên có thể được phân định là thành phần cứng hay mềm tùy thuộc vào hệ thống và ý nghĩa sử dụng. Thông thường, các RTS được kết hợp vào phần cứng có khả năng tốt hơn so với hệ thống phần mềm có chức năng tương ứng và tránh được chi phí quá đắt cho việc tối ưu hoá phần mềm. Ngày nay, chi phí phần cứng ngày càng rẻ, chọn lựu ưu tiên phần cứng là một xu hướng chung.

2.2. Các loại hệ thống thời gian thực:

Các RTS thường được phân thành hai loại sau Soft reatime system và Hard realtime system : Đối với Soft realtime system, thời gian trả lời của hệ thống cho yếu tố kích thích là quan trọng, tuy nhiên trong trường hợp ràng buộc này bị vi phạm, tức là thời gian trả lời của hệ thống vượt quá giới hạn trễ cho phép, hệ thống vẫn cho phép tiếp tục hoạt động bình thường, không quan tâm đến các tác hại do sự vi phạm này gây ra (Thường thì tác hại này là không đáng kể).

Ngược lại với Soft realtime system là Hard realtime system, trường hợp này người ta quan tâm khắc khe đến các hậu quả do sự vi phạm giới hạn thời gian để cho phép bởi vì những hậu quả này có thể là rất tồi tệ, thiệt hại về vật chất, có thể gây ra những ảnh hưởng xấu đến đời sống con người. Một ví dụ cho loại này là hệ thống điều khiển không lưu, một phân phối đường bay, thời gian cất cánh, hạ cánh không hợp lý, không đúng lúc có thể gây ra tai nạn máy bay mà thảm họa của nó khó mà lường trước được.

Trong thực tế thì có nhiều RTS bao gồm cả hai loại soft và hard. Trong cả hai loại này, máy tính thường can thiệp trực tiếp hoặc gián tiếp đến các thiết bị vật lý để kiểm soát cũng như điều khiển sự hoạt động của thiết bị này. Đứng trên góc độ này, người ta thường chia các RTS ra làm hai loại sau :

i) Embedded system : Bộ vi xử lý điều khiển là một phần trong toàn bộ thiết bị, nó được sản xuất trọn gói từ yếu tố cứng đến yếu tố mềm từ nhà máy, người sử dụng không biết về chi tiết của nó và chỉ sử dụng thông qua các nút điều

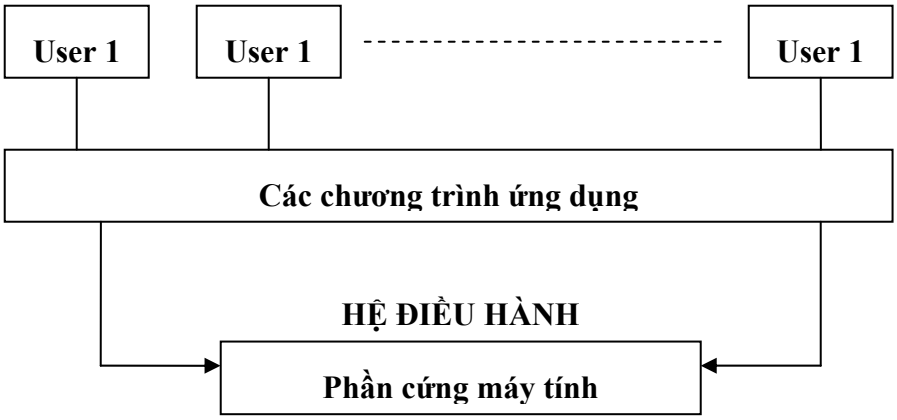
khiển, các bảng số. Với hệ thống này, ta sẽ không thấy được những thiết bị như trong máy tính bình thường như bàn phím, màn hình... mà thây vào đó là các nút điều khiển, các bảng số, đèn tín hiệu hay các màn hình chuyên dụng đặc trưng cho từng hệ thống. Máy giặt là một ví dụ. Người sử dụng chỉ việc bấm nút chọn chương trình giặt, xem kết quả qua hệ thống đèn hiệu... Bộ vi xử lý trong Embedded system này đã được lập trình trước và gắn chặt vào ngay từ khi sản xuất và không thể lập trình lại. Những chương trình này chạy độc lập, không có sự giao tiếp với hệ điều hành (HĐH) cũng như không cho phép người sử dụng can thiệp vào.

ii) Loại thứ hai là bao gồm những hệ thống có sự can thiệp của máy tính thông thường. Thông qua máy tính ta hoàn toàn có thể kiểm soát cũng như điều khiển mọi hoạt động của thiết bị phần cứng của hệ thống này. Những chương trình điều khiển này có rất nhiều loại, phục vụ cho nhiều mục đích khác nhau và có thể được viết lại cho phù hợp với yêu cầu thực tế. Hiển nhiên thì loại hệ thống này hoạt động được phải cần một HĐH điều khiển máy tính. HĐH này phải có khả năng nhận biết được thiết bị phần cứng, có khả năng hoàn tất công việc trong giới hạn thời gian nghiêm ngặt. HĐH này phải là HĐH hổ trợ xử lý thời gian thực – Realtime operating system (RTOS).

3. Hệ điều hành cho hệ thống thời gian thực :

3.1. Sơ lược về hệ điều hành :

Cho đến nay, nhìn chung thì chưa có một định nghĩa nào là hoàn hảo về hệ điều hành (HĐH). HĐH được xem như một chương trình hoạt động giữa người sử dụng và phần cứng máy tính với mục tiêu cung cấp một môi trường để thực thi các chương trình ứng dụng và thuận lợi, hiệu quả hơn trong việc sử dụng máy tính.



Hình 1: Mô hình trừu tượng của hệ thống máy tính

Cho đến ngày nay, HĐH đã phát triển với nhiều loại khác nhau như :HĐH quản lý theo lô đơn giản, quản lý theo lô đa chương (Multiprogram), chia xẻ thời gian (Multitasking), xử lý song song, mạng và phân tán...

3.2. Quan niệm tiến trình, tiểu trình :

Trong các HĐH hiện đại ngày nay, quan niệm tiến trình và tiểu trình là trung tâm của cả hệ thống, tất cả các xử lý đều tập trung vào tiến trình, vào tiểu trình. Ở đây để thuận tiện, ta chú trọng vào môi trường Windows 32 bit.

Một tiến trình được xem như là một thể hiện đang thực thi của một chương trình. Trên môi trường Windows 32 bit, một tiến trình sở hửu 4 GB không gian địa chỉ bộ nhớ không phụ thuộc vào bộ nhớ vật lý. Tất cả các DLL cần thiết đều được map vào không gian địa chỉ này. Khi một tiến trình được tạo lập, có một tiểu trình chính được tạo lập và tiến trình kết thúc khi tất cả các tiểu trình con đều kết thúc. Một tiến trình có thể có nhiều tiểu trình con và có thể tạo lập các tiến trình khác.

Tiểu trình là một thành phần xử lý cơ bản của tiến trình, tiểu trình sở hữu một con trỏ lệnh riêng, tạp các thanh ghi riêng, stack riêng và tất cả nằm trong không gian địa chỉ của tiến trình sở hữu. Như vậy, các tiểu trình trong một tiến trình có thể chia sẻ các tài nguyên với nhau. Tất cả các công việc điều phối tiến trình đều nhắm vào hoạt động của tiểu trình.

Các tiểu trình, tiến trình phải liên lạc với nhau để có một cơ chế điều phối hợp lý, để có một cách thức chia sẽ dữ liệu với nhau. Các cơ chế liên lạc và chia sẽ dữ liệu được

các HĐH và NNLT hiện đại quan tâm như sử dụng tín hiệu, pipe, vùng nhớ chia sẽ, trao đổi thông điệp, sử dụng socket...v.v.

3.3. Hệ điều hành thời gian thực :

Hệ điều hành thời gian thực (RTOS - Realtime Operating system) là HĐH có sự chú trọng giải quyết vấn đề đòi hỏi khắc khe về thời gian cho các thao tác xử lý hoặt dòng dữ liệu. Đây là HĐH hiện đại, tinh vi, thời gian xử lý nhanh, phải cho kết quả chính xác trong thời gian bị thúc ép nhanh nhất. HĐH này

thường sử dụng một đồng hồ hệ thống có cho kỳ ngắt nhỏ vào khoảng vài micro giây để thực hiện điều phối các tiến trình.

Các HĐH hiện đại ngày nay phần lớn đều hổ trợ (ở mức tương đối) xử lý thời gian thực, cung cấp một môi trường có thể tổ chức các RTS. Theo sự đánh giá của các chuyên viên RTS thì cho đến nay, các HĐH thuộc họ UNIX là có thể đáp ứng tốt nhất các yêu cầu khắc khe của các RTS phức tạp. Tuy nhiên, trong khuôn khổ luận văn này cùng với các yêu cầu cũng như hiện trạng thực tế, hệ thống đang đươc quan tâm sẽ triển khai trên hệ thống máy PC sủ dụng HĐH Windows 9x 32 bit.

<u>Hê điều hành windows và vấn đề thời gian thực:</u>

Windows được thiết kế bởi hãng Microsoft, ra đời vào 11/1985, cho đến nay đã trải qua nhiều phiên bản và cải tiến. Windows được sử dụng rộng rãi nhất trên thế giới máy vi tính cá nhân (PC) và đã đưa Microsoft thành công ty hàng đầu thế giới trong lĩnh vực tin học. Ở đây ta quan tâm đến các Windows với các phiên bản 32 bit.

Là HĐH đa nhiệm (Multitasking) xử lý 32 bít, chạy trên môi trường máy PC, có hỗ trợ cho việc xử lý thời gian thực, có yêu cầu cấu hình không cao, nếu cấu hình cao thì tốc độ xử lý càng nhanh, tương thích các HĐH khác, có nhiều đặc điểm được mọi người ưu chuộng như giao diện đồ hoạ thân thiện, tính an toàn, khả năng Plus and Play... v.v. Về vấn đề thời gian thực, HĐH đa nhiệm này đặt nền móng trên sự chia xẻ thời gian. Khái niệm tiến trình và khái niệm tiểu trình là trung tâm nền tảng cho vấn đề xử lý điều phối, đồng bộ và các xử lý liên quan thời gian thực khác. HĐH này vốn được viết phần lớn bằng ngôn ngữ C/C++... là một trong những ngôn ngữ lập trình (NNLT) có khả năng hổ trợ xử lý thời gian thực (Xem phần Ngôn ngữ lập trình RTS). Cung cấp thư viện dùng chung API cho phép giao tiếp với hệ thống cũng như

tổ chức đồng bộ hoá tiến trình, tiểu trình. Windows không hổ trợ can thiệp trực tiếp vào hệ thống hay các thiết bị ngoại vi (nhưng vẫn cho phép), tuy nhiên lại cung cấp một môi trường giao tiếp dễ dàng.

4. Ngôn ngữ lập trình cho hệ thống thời gian thực:

4.1. Tổng quan về ngôn ngữ lập trình cho hệ thống thời gian thực:

Phần lớn các ứng dụng thời gian thực không thể viết bằng các ngôn ngữ lập trình (NNLT) truyền thống dưới những HĐH truyền thống bởi vì các NNLT này không hỗ trợ các xử lý có sự ràng buộc khắc khe về thời gian thực thi. Cũng có một số NNLT loại này có phần mở rộng hổ trợ cho phép viết chương trình xử lý thời gian thực bằng cách can thiệp trực tiếp vào phần cứng mà không thông qua NNLT đang chạy.

Một số RTS được viết từ ngôn ngữ kinh điển như C nếu được cung cấp thêm thư viện các hàm hổ trợ xử lý thời gian thực, yếu tố thời gian thực lúc này là sự chia xẻ giữa NNLT và RTOS đang chạy.

Ngày nay có nhiều NNLT hổ trợ viết chương trình xử lý thời gian thực, ví dụ như Ada chuyên trong các lĩnh vực quân sự. Java vốn được thiết kế để dùng trong các hệ thống nhúng trong các thiết bị dân dụng, truyền thông. Java có cơ chế hổ trợ đa nhiệm riêng không phụ thuộc vào HĐH. C/C++ được cung cấp các thư viện hàm hỗ trợ cơ chế xử lý thời gian thực theo nhiều HĐH hổ trợ xử lý thời gian thực khác nhau... v.v.

4.2. Sơ lược về ngôn ngữ lập trình C(/C++):

NNLT C(/C++) ngày nay được sử dụng rộng rãi trên nhiều phương diện cũng như nhiều loại máy tính, là NNLT dùng để viết nhiều NNLT, trình biên dịch cũng như viết các ứng dụng thương mại...

NNLT C(/C++) được thiết kế vào năm 1973 bởi tiến sĩ Denis Ritche thuộc diện nghiên cứu Bell trực thuộc hãng AT&T, NNLT này được thuyết kế để viết HĐH UNIX - một (họ) HĐH được rất nhiều người sử dụng cho đến hiện nay, trên cả máy mainframe và hiện nay là PC.

Ngày nay trên thị trường có rất nhiều trình biên dịch cho cả C và C++, phần lớn đều dựa trên chuẩn ANSI như Turbo C/C++, Borland C/C++, Builder C/C++ của hãng Borland, Microsoft C/C++, Visual C/C++ của Microsoft... C là NNLT cấp trung, có

cấu trúc (nhưng không chính thống), tuy nhiên C là một NNLT mạnh cả về khía cạnh cú pháp cũng như phát sinh mã thực thi. C kết hợp được cả yếu tố mềm dẽo và khả năng điều khiển mạnh mẽ của Assembly cũng như tính dễ hiểu, rỏ ràng.. của các ngôn ngữ cấp cao khác như BASIC, Pascal...

Về vấn đề thời gian thực, NNLT C(/C++) vốn dùng để viết HĐH UNIX – một HĐH có khả năng xử lý thời gian thực tốt nhất hiện nay như đã đề cập trên. C(/C++) còn được dùng để viết nhiều HĐH hiện đại khác ngày nay. C(/C++) có sẳn thư viện hàm xử lý thời gian chuẩn, mức độ chính xác thời gian xử lý có thể lên đến hàng micro giây. Bên cạnh đó là khả năng giao tiếp trực tiếp với thiết bị phần cứng, khi cần có thể gọi trực tiếp các đoạn mã viết bằng Assembly hay chèn trực tiếp mã Assembly vào chương trình viết bằng C.

Trên mỗi nền HĐH khác nhau như Windows, UNIX... C(/C++) còn được cung cấp hệ thống các hàm hổ trợ xử lý thời gian thực, hổ trợ đồng bộ hóa các quá trình, ràng buộc toàn vẹn, độc quyền truy xuất... giúp cho C(/C++) có khả năng điều khiển đến từng tiến trình, từng tiểu trình đang thực thi.

Trong xu hướng ngày nay, công nghệ hướng đối tượng cũng rất được quan tâm trong lĩnh vực RTS, có nhiều NNLT hướng đối tượng hổ trợ xây dựng RTS ra đời như C++, Java... Tuy nhiên những NNLT này lại vi phạm về phương pháp luận của RTS như có quá nhiều thao tác phụ làm mất thời gian xử lý. Trong thực tế, đã có một số đề án về RTS xây dựng dựa trên không gian đối tượng (cả về thiết kế và cài đặt) phải ngưng giữa chừng hoặt phải chuyển hướng vì không đáp ứng được các ràng buộc thời gian mà nguyên nhân sâu xa là bắt nguồn từ NNLT có qua nhiều thao tác phụ nói trên và việc bao bọc dữ liệu theo phương pháp hướng đối tượng làm mất thời gian truy xuất.

5. Quan niệm thời gian trong hệ thống thời gian thực:

5.1. Đồng hồ hệ thống:

Thời gian hệ thống được báo bằng một đồng hồ gọi là đồng hồ hệ thống. Trong môi trường có nhiều vi xử lý có thể tồn tại nhiều đồng hồ, thì những đồng hồ này phải được đồng bộ với nhau.

Có thể biểu diễn mức độ chính xác của đồng hồ hệ thống qua hàm số sau:

$$C(t) = t, \forall t$$

Đồng hồ được gọi là chính xác vào thời điểm t_i nếu :

$$C(t_i) = t_i, \forall t \qquad \text{hay} \qquad \left.\frac{dC(t)}{dt}\right|_{t_i} = 1$$

5.2. Các loại đồng hồ hệ thống:

Đơn giản nhất là hệ thống chỉ có một đồng hồ (sever clock), yêu cầu độ chính xác và tin cậy rất cao. Loại này giá thành rất đắt.

Một loại khác gồm một đồng hồ chính (master clock) đồng bộ với nhiều đồng hồ phụ (slave clock) theo kiểu "polling", tất cả các đồng có cùng độ chính xác, nếu đồng hồ chính bị hỏng thì một trong những đồng hồ phụ sẽ thay thế.

Đối với các hệ thống phân tán, đồng hồ hệ thống bao gồm tất cả các đồng hồ phân tán và được đồng bộ với nhau theo cùng một thuật toán.

5.3. Quan niệm về sự rời rạc thời gian:

Trong quan niệm của RTS, thời gian được xem như là một yếu tố rời rạc. Đây là một khía cạnh rất phức tạp và lý thú.

i) Trong các HĐH kinh điển, có một đồng hồ quản lý thời gian đồng bộ giữa các tiến trình. Đồng hồ này phát sinh ra ngắt báo hiệu cho hệ thống theo chu kỳ. Chu kỳ này có thể được điều chỉnh nhưng không quá nhanh hay quá chậm làm ảnh hưởng đến thời gian thực thi các tiến trình, và thường là vào khoảng vài chục mili giây. Chính chu kỳ này đã chia thời gian ra thành các mảnh đủ nhỏ.

ii) Còn trong các RTOS, hệ thống sử dụng một đồng hồ có khả năng lập trình điều phối ngắt theo một chu kỳ đủ nhỏ hợp lý, chu kỳ ở hệ thống này vào khoảng vài micrô giây.

Trong thực tế thì các RTS thường dựa trên cách tiếp cận kết hợp giữa hai quan niệm trên, thường thì quan điểm i) là nền tảng có sự hổ trợ của quan điểm ii).

5.4. Ràng buộc về thời gian:

Với mỗi yếu tố kích thích, hệ thống tiếp nhận vào một thời điểm t_0, hệ thống tiến hành cấp phát tài nguyên, thực hiện các xử lý tính toán và hoàn tất việc trả lời vào thời điểm tk khác sau đó.

Một ràng buộc tối thiểu có thể được định nghĩa qua bộ ba sau:

(ID, T*begin*(condition1), T*end*(condition2))

Trong đó:

ID :	Chỉ số của tiến trình
Tbegin(condition1) :	Thời gian bắt đầu tiến trình
Tend(condition2) :	Thời gian tiến trình hoàn tất xử lý

Phụ thuộc vào hệ thống và thời gian xác định được tài nguyên cần cấp phát, cũng như quá trình giải phóng tài nguyên sau khi tiến trình sử dụng… Một ràng buộc khắc khe hơn có thể xác định như sau:

(ID, T*begin*(condition1), C*ID,* F*ID,* T*end*(condition2))

Trong đó:

ID :	Chỉ số của tiến trình
Tbegin(condition1) :	Thời gian bắt đầu tiến trình
Tend(condition2) :	Thời gian tiến trình hoàn tất xử lý
CID :	Thời gian ước tính của tiến trình (số mẫu thời gian)
FID :	Tần số mẫu thời gian

Mỗi chỉ thị cơ sở (Assembly) có một thời gian thực thi cố định phụ thuộc vào phần cứng, ví dụ :

Chỉ thị	**Thời gian thực thi (clock)**
MOV reg8, reg8	2
JMP	15
IRET	24
IN	10
OUT	10

Như vậy, mỗi tiểu trình thực hiện một công việc được viết bằng một nhóm các chỉ thị (hàm) sẽ có thời gian thực hiện là cố định, thêm vào đó còn có thời gian dùng để khởi tạo tiểu trình, kết thúc tiểu trình dẫn đến thời gian thực hiện công việc đó sẽ lớn hơn

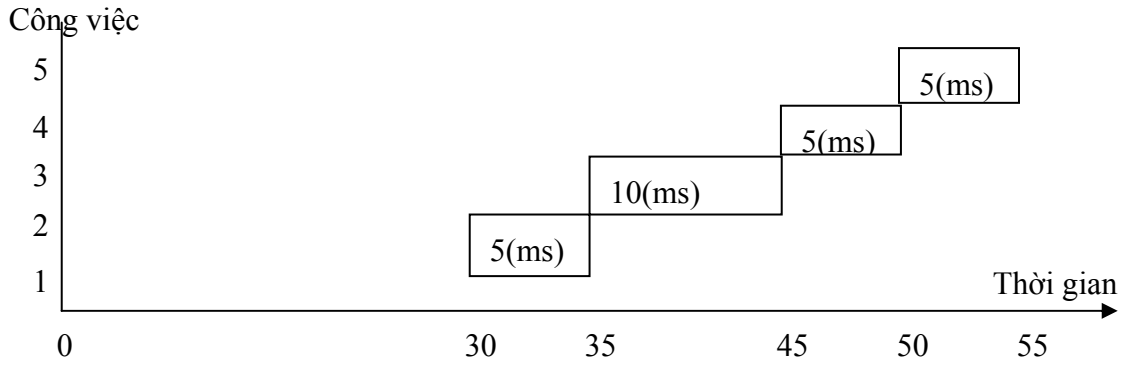
thời gian thực thụ thực hiện tiểu trình. Câu hỏi đặt ra là làm thế nào những công việc có thể thực thi một cách hoàn chỉnh trong thời gian bị hạn chế. Câu trả lời đó là cơ chế Điều Phối Quá Trình được xem xét ở phần tiếp theo.

6. Vấn đề điều phối công việc :

Để thấy được tính năng của việc điều phối, ta xem xét ví dụ sau : Giả sử có một yêu cầu tác vụ gồm các công việc sau :

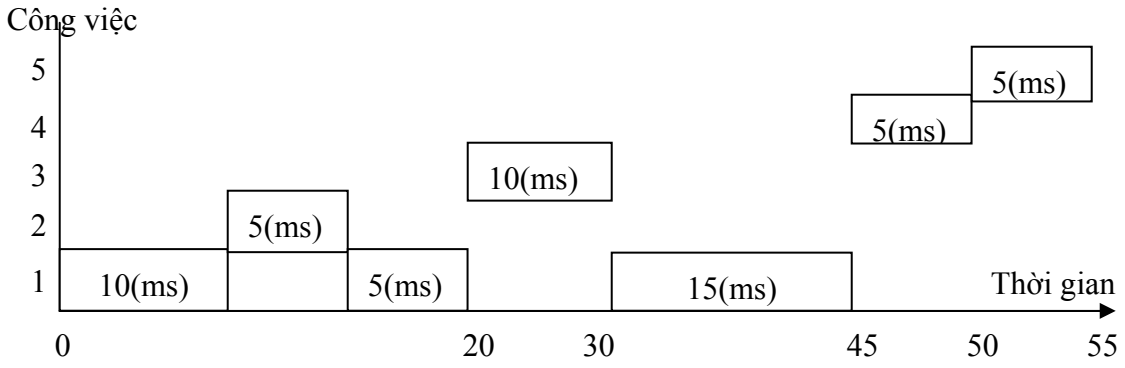
1) Đọc đĩa cứng lấy dữ liệu, thời gian thực hiện hết 30 ms.
2) Xác lập vùng nhớ trên bô nhớ chính, hết 5 ms và phải bắt đầu tại thời điểm 10 ms sau khi tác vụ bắt đầu.
3) Nhận tín hiệu từ thiết bị ngoại vi, hết 10 ms, bắt đầøu lúc 20 ms.
4) Tính toán số liệu dựa trên kết quả tín hiệu nhận và dữ liệu trên đĩa đã đọc, hết 5 ms.
5) Kết xuất màn hình, hết 5 ms.

Trong các hệ thống kinh điển (xử lý tuần tự theo lô) thì khó mà có thể đáp ứng được yêu cầu công việc trên. Các công việc thực hiện một cách tuần tự theo sơ đồ sau :



Hình 2 : Mô hình điều phối tiến trình cổ điển (FIFO)

Và như vậy các yêu cầu về ràng buộc thời gian đã bị phá vở. Như nếu có thể tổ chức cho các công việc thực hiện theo mô hình dưới đây thì hoàn toàn có thể đáp ứng được như cầu về thời gian :



Hình 3 : Mô hình điều phối tiến trình cải tiến (Round Robin, quantum = 5ms)

Như vậy điều phối tiến trình là một công việc cần thiết cho RTOS nói chung và các RTS nói riêng. Mục tiêu của việc điều phối đem lại là : Sự công bằng trong chia sẽ CPU, tính hiệu quả (tận dụng CPU 100%), thời gian đáp ứng (response time) hợp lý, cực tiểu thời gian lưu lại trong hệ thống, thông lượng tối đa (throughtput). Tuy vậy, bản thân các mục tiêu này đã có sự mâu thuẩn nên không thể thoả mãn tất cả các mục tiêu, chiến lược cụ thể phụ thuộc vào từng hệ thống.

Trong môi trường đa nhiệm, để tránh việc một tiến trình độc chiếm CPU quá lâu làm ngăn cản công việc của các tiến trình khác. Sử dụng một đồng hồ hệ thống để tổ chức phân phối thời gian thực thi của mỗi tiến trình. Cơ chế điều phối có thể là độc quyền hoặt không độc quyền. Điều phối có thể là điều phối tác vụ hoặt điều phối tiến trình. Một tiến trình được phân phối CPU dựa trên các độ ưu tiên khác nhau, độ ưu tiên này có thể là tỉnh hoặt động. Có nhiều chiến lược điều phối khác nhau như chiến lược FIFO, Round Robin, Sử dụng độ ưu tiên, công việc ngắn nhất...

Điều phối tiến trình cho khả năng hoàn thành tốt nhóm các công việc trong khoảng thời gian ràng buộc. Lại dẫn đến vấn đề tranh chấp tài nguyên dùng chung, cần phải có một cơ chế phối hợp, đồng bộ hoá việc độc quyền truy xuất. Vấn đề Đồng Bộ Hoá sẽ được xem xét dưới đây.

7. Vấn đề đồng bộ hoá:

7.1. Cơ chế đồng bộ hoá:

Trong các HĐH hiện đại nói chung và trong các RTS nói riêng thì việc đồng bộ hoá việc thực thi các tiến trình, tiểu trình là rất quan trọng, phức tạp và nhiều điều thú vị. Mục tiêu chính của việc đồng bộ là tránh sự tranh chấp tài nguyên, môi trường của

các tiến trình đang thực thi, yêu cầu phối hợp giữa các công việc. Tồn tại rất nhiều cơ chế cũng như thuật toán đồng bộ, ở đây ta xem xét một số cơ chế đồng bộ mang tính cơ bản như : Giải pháp "busy waiting" luôn phiên kiểm tra, các cơ chế được hỗ trợ từ phần cứng, giả pháp "SLEEP and WAKEUP" .

7.2 Phương pháp đồng bộ trên môi trường Windows:

HĐH Windows là HĐH đa nhiệm, có hổ trợ cơ chế đồng bộ hoá các tiến trình, tiểu trình đặt biệt là trong môi trường Windows NT và các Windows 9x phiên bản 32 bít. Ở đây ta chú ý đến các phương pháp đồng bộ thông dụng, tương thích với Windows, phiên bản 32 bít.

1) Phương pháp sử dụng miền găng (Critical section): Miền găng là một đoạn chương trình chỉ cho phép một tiểu trình thực hiện tại một thời điểm, tiểu trình thực hiện đoạn mã đó gọi là tiểu trình trong miền găng.
2) Phương pháp sử dụng Mutex: Là một đối tượng đồng bộ hoá nhận trạng thái TRUE khi không có tiểu trình nào sở hửu nó và nhận trạng thái FALSE khi có một tiểu trình sở hữu nó.
3) Phương pháp sử dụng Semaphore: Là đối tượng đồng bộ hoá lưu giữ một biến đếm có giá trị từ 0 đến Max, Semaphore nhận giá trị TRUE khi biến đếm lớn hơn 0 và giá trị FALSE khi biến đếm có giá trị bằng 0.
4) Phương pháp sử dụng biến cố (Event) :Là đối tượng đồng bộ hoá được dùng để đồng bộ các thao tác truy xuất đồng thời đến các đối tượng dùng chung, thực chất thì biến cố là một cờ có hai trạng thái TRUE/FALSE. Có hai loại biến cố :
    - Biến cố không tự động : Đối tượng sẽ giữ trạng thái TRUE cho đến khi có tiểu trình tường minh xác lập lại trạng thái FALSE cho đối tượng.
    - Biến cố tự động : Đối tượng sẽ giữ trạng thái TRUE cho đến khi có một tiểu trình đang chờ đợi được giải phóng, hệ thống sẽ đặt lại trạng thái FALSE cho đối tượng.

8. Một số yêu cầu của hệ thống thời gian thực :

Các RTS có một số đặc biệt đặc trưng cho loại hệ thống này, tuy nhiên không phải tất cả các RTS đều quan tâm đến các đặc điểm này. Thường thì NNLT và HĐH cho RTS đã rất nhiều cho một số đặc điểm hoặt tạo môi trường thuận lợi cho việc thực hiện các đặc điểm.

8.1. Hệ thống lớn và phức tạp:

Đây là vấn đề chung cho cả lĩnh vực phần mềm, yếu tố phức tạp và tầm cở của hệ thống thường tỉ lệ thuận với nhau. Đặc biệt khi mà RTS phải phân chia thời gian hợp lý, sử dụng nhiều thuật toán phức tạp, phải thực hiện lập lịch, đồng bộ.. nên độ phức tạp là rất lớn, cả từ các giai đoạn đặt vấn đề, phân tích, thiết kế, tiến hành, kiểm tra, bảo trì...Ở đây sẽ có một nghịch lý, đáp ứng thời gian thực yêu cầu giải quyết vấn đề nhanh gọn và chính xác, mã thực thi chương trình nhỏ gọn.

8.2. Xử lý trên số thực:

RTS luôn làm việc trên các thông số trạng thái thực của thiết bị vật lý. Việc tính toán trên số thực tốn rất nhiều thời gian xử lý. Ngày nay, tốc độ xử lý của máy tính đã rất nhanh, việc xử lý số thực được hổ trợ ngay từ phần cứng, HĐH và NNLT nhưng sử dụng một phương pháp tính toán phù hợp, ít tốn thời gian nhất vẫn là một yêu cầu thực tế.

8.3. Thực sự an toàn và đáng tin cậy:

Những hậu quả do sự thiếu an toàn của những hệ thống thông tin nói chung và RTS nói riêng có thể lên đến hàng tỉ đôla, thậm chí gây thiệt hại về tính mạng của nhiều người. Việc thiết lập một RTS có độ tin cậy cao và an toàn là một yêu cầu hàng đầu, phải có cách lường trước được những lỗi có thể xãy ra và các biện pháp khắc phục.

8.4. Giao tiếp trực tiếp với thiết bị phần cứng:

Các thiết bị vật lý giao tiếp trực tiếp thường là các bộ cảm biến, các loại đồng hồ trạng thái, nhiệt kế, các thiết bị điện điện tử, bán dẫn… Các thiết bị này có khả năng phát phát sinh hoặt tiếp nhận các tín hiệu, phát sinh các ngắt được nhận biết bởi máy tính. Thông qua các tín hiệu, các ngắt đó mà máy tính có thể kiểm soát các trạng thái hoặt điều khiển sự hoạt động của thiết bị.

8.5. Thực hiện trên môi trường và ngôn ngữ lập trình hiệu quả:

Khác với các hệ thống khác, RTS có yêu cầu thực thi nhanh và hiệu quả. Vì vậy việc sử dụng một môi trường không hợp lý hay một NNLT bình thường thì khó mà có thể đạt được yêu cầu, ví dụ: Hệ thống xử lý thời gian có độ chia cho đến micro giây thì không thể thực hiện trên NNLTị chỉ hổ trợ đến mili giây.

8.6. Người sử dụng điều khiển :

Trong các hệ thống kinh điển, thường thì người sử dụng ra yêu cầu và chờ nhận kết quả. Trong những RTS còn yêu cầu người sử dụng phải nắm vững về hệ thống nền (HĐH). Để kết quả công việc thành công tốt, người sử dụng có thể can thiệp trực tiếp đến từng tiến trình, từng tiểu trình, cho phép, cấp quyền ưu tiên đến từng công việc nhằm tạo sự thông suốt trong hệ thống, tránh sự tắt ngẽn.

10. Phương pháp phân tích thiết kết Hệ thống thời gian thực :

10.1. Sơ lược về phương pháp thiết kế phần mềm:

Quá trình xây dựng một phần mềm trãi qua nhiều giai đoạn liên tiếp nhau, trong quá trình thực hiện có thể quay lại những giai đoạn trước đó. Việc phân chia thành các giai đoạn này làm cho quá trình xây dựng rỏ ràng hơn, các giai đoạn có thể thực hiện độc lập bởi đội ngủ làm việc. Có thể phân thành các giai đoạn sau:

1) Xác định vấn đề bài toán.
2) Phân tích hệ thống.
3) Thiết kế dữ liệu và chương trình.
4) Cài đặt.
5) Kiểm tra và cải tiến.
6) Nghiệm thu.
7) Khai thác và bảo trì.

Trong các giai đoạn trên thì giai đoạn thiết kế là rất quan trọng. Chất lượng của phần mềm phụ thuộc rất nhiều vào bản thiết kế. Một bản thiết kế tốt còn giúp cho việc thực hiện các giai đoạn khác dể dàng hơn, giúp cho những người thực hiện hoàn thành chính xác hơn công việc của mình. Các chiến lược phân tích thiết kế thường được sử dụng như :

1) Chiến lược từ trên xuống (Bottom-Up design) : Chiến lược này được tiếp cận theo hướng xem xét bài toán từ các khía cạnh chi tiết và sau đó mới tổng quát lên.

2) Chiến lược từ dưới lên (Top-Down design):Ngược với Bottom-Up là Top-Down, tiếp cận theo hướng từng bước từ tổng quát dần đến chi tiết bài toán, ban đầu bài toán được nhìn dưới dạng tổng quan và dần đi sau vào từng chi tiết.

3) Kết hợp cả hai chiến lược:Trong thực tế có nhiều chương trình được thiết kế kết hợp cả hai hướng tiếp cận Bottom-Up và Top-Down, cách tiếp cận này là một phương pháp tốt, tận dụng được các ưu điểm của hai cách tiếp cận trên thậm chí còn loại bớt một số khuyết điểm của chúng.

10.2. Thiết kế ứng dụng thời gian thực :

Thông thường, mỗi RTS thường được thiết kế dựa trên một số hệ thống chuẩn. Có ba dạng chuẩn thường gặp là :

- Các hệ thống giám sát (Monitoring system).
- Các hệ thống kiểm soát (Control system).
- Các hệ thống thu thập dữ liệu (Data accquistion system). (Hệ thống mà luận văn đang quan tâm kết hợp giữa hai hệ thống kiểm soát và thu thập dữ liệu – Data accquistion and Control)

Xây dựng một RTS cũng bắt đầu bằng giai đoạn xác định yêu cầu, sau đó là các giai đoạn phân tích, thiết kế, cài đặt, kiểm tra…. Các giai đoạn này có thể là tổng quan hay là đi vào chi tiết, có thể phân thành các giai đoạn con. Các giai đoạn có thể gối chồng lên nhau. Xác định Phân tích Thiết kế Cài đặt Kiểm tra Nghiệm thu Khai thác & bảo trì Hình 4 : Mô hình thác nước các giai đoạn xây dựng phần mềm

Trong quá trình thiết kế cần có sự mềm dẻo, không đi quá sau vào chi tiết đặt biệt là các chi tiết về kỹ thuật, phần cứng. Sự phân định giữa phần cứng và phần mềm càng trì hoãn trong giai đoạn thiết kế càng tốt.

Song song với các giai đoạn là việc tổ chức tài liệu kỹ thuật của giai đoạn đó.

Việc làm này cần thiết cho các giai đoạn khác trong toàn bộ quá trình. Trong quá trình xây dựng một ứng dụng thời gian thực thì các giai đoạn phân tích và thiết kế có tính chất quan trọng đặt biệt, giai đoạn này bao gồm cả việc xác định các ràng buộc về thời gian. Giai đoạn này cần phải:

- Xác định các nhân bên ngoài ảnh hưởng đến hệ thống.
- Xác định cách trả lời của hệ thống cho các tác nhân đó.
- Xác định các ràng buộc thời gian ứng dụng phải đáp ứng. Theo DART – Design Approach for Realtime System (được phát triển bởi General Electric), trong RTS thì không phải công việc nào cũng đòi hỏi thời gian thực, mỗi công việc tương ứng với những chức năng cụ thể, được phân thành các nhóm sau :
- Phụ thuộc nhập xuất (I/O dependency): Thường thì tốc độ thực hiện trên các thiết bị chậm hơn rất nhiều so với tốc độ xử lý của CPU, những công việc liên quan đến nhập xuất trên các thiết bị này thường được phân vào một nhóm.
- Ràng buộc thời gian (Time-critical): Nhóm công việc có ràng buộc thời gian thực thi, yêu cầu quyền ưu tiên cao được phân vào một nhóm.
- Thực thi theo định kỳ (Periodic execution): Nhóm công việc yêu cầu thực hiện theo một chu kỳ chỉ định.
- Yêu cầu tính toán (Computational requirements): Những công việc có nhu cầu tính toán cao được phân thành một nhóm.

Để thiết kế được những hệ thống phức tạp nói chung và RTS nói riêng thì phải có những phương pháp luận rỏ ràng, những công cụ cụ thể phản ánh được bản chất của vấn đề nhưng không quá rườm rà phức tạp. Phần dưới đây là phần trình bày những phương pháp luận, những công cụ thường dùng trong thiết kế các RTS, như mô hình đối tượng, mạng Petri...

10.3. Mô hình đối tượng :

Là phương tiếp cận dựa trên cách tiếp tiếp cận mô hình các đối tượng của thế giới thực được quan tâm trong chương trình. Không đồng nghĩa với phương pháp lập trình hướng đối tượng, phương pháp này được thực hiện dễ dàng, đáp ứng được các yêu cầu của Công nghệ phần mềm như tính đúng đắn, tính tiến hoá, tính hiệu quả, tính tiện dụng, tính tương thích, tính tái sử dụng...

Từ mô hình đối tượng sẽ có một loạt các mô hình liên quan dựa trên quan điểm đối tượng như :

- Mô hình trạng thái : Diễn tả chu trình sống của một đối tượng từ lúc sinh ra đến lúc mất đi.
- Mô hình xử lý : Hệ thống những nghiệp vụ trong thế giới thực tác động lên đối tượng.
- Mô hình không gian : Hệ thống các vị trí mà trong đó các đối tượng được sinh ra và mất đi, các nghiệp vụ được tiến hành.
- Mô hình thời gian : Hệ thống các mà trong đó các nghiệm vụ trên các đối tượng được thực hiện, các đối tượng được phát sinh, được mất đi theo những ràng buộc cụ thể.
- Mô hình người sử dụng... v.v.

Trong lĩnh vực ứng dụng thời gian thực, mô hình đối tượng nếu được sử dụng để thiết kế nên tập trung chủ yếu vào mô hình trạng thái và mô hình xử lý.

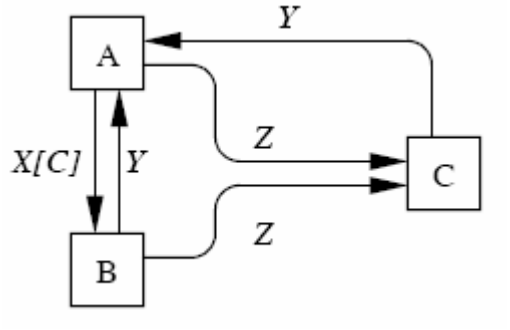
Tuy vậy trong lĩnh vực RTS này, yêu cầu cao nhất và nhiều nhất là yêu cầu xử lý, ràng buộc về thời gian, vấn đề đồng bộ, điều phối. Dữ liệu cho quá trình xử lý là cần thiết nhưng không phải là trung tâm. Sự phân lớp đối tượng sẽ gặp nhiều khó khăn thậm chí nếu cố gắng đôi khi lại đem đến kết quả sai lệch với mục đích của hệ thống.

Do vậy thường trong lĩnh vực này người ta chỉ sử dụng mô hình đối tượng như một mô hình tổng quát của hệ thống. Khi đi vào thiết kế chi tiết xử lý sẽ sử dụng những phương pháp đặc biệt, đặc trưng cho lĩnh vực này. Trong luận văn này, sẽ trình bày hai phương pháp thiết kế cho đặc trương này và dùng nó cho phần thiết kế ứng dụng cụ thể. Hai phương pháp đó là Sơ đồ trạng thái – một phương pháp thường gặp và Phương pháp Mạng Petri – Đồ thị Petri (Petri net và Petri graph).

10.4. Sơ đồ trạng thái (State chart, state diagram):

Một công cụ tương đối mạnh mẽ trong lĩnh vực thiết kế RTS là dùng sơ đồ trạng thái. Sơ đồ trạng thái mô tả các trạng thái cũng như quá trình biến đổi giữa các trạng thái đó trong một hệ thống cùng với các sự kiện được kích hoạt, các điều kiện ràng buộc.

Các trạng thái được mô tả bằng các hình chữ nhật, quá trình biến đổi từ trạng thái này sang trạng thái khác mô tả bằng các mũi tên, các sự kiện, ràng buộc là các nhãn kèm theo các mũi tên...



Hình 5: Ví dụ mộ sơ đồ trạng thái

Sơ đồ trạng thái cho phép nhìn hệ thống dưới những mức độ chi tiết khác nhau. Sơ đồ trạng thái có thể được phân rã xuống mức trạng thái thấp hơn hoặt là liên kết với mức trạng thái cao hơn, sự kết hợp này cho phép nhìn thấy giao tiếp giữa các lớp trạng thái khác nhau trong hệ thống.

10.5. Mạng Petri và đồ thị Petri (Petri net and Petri graph):

Mạng Petri và đồ thị Petri là một công cụ rất mạnh mẽ và thường được dùng trong việc thiết kế những hệ thống có sự ràng buộc về thời gian. Những RTS về bản chất, phức tạp ở chổ phải thường xuyên giám sát chặt chẽ toàn bộ các tác động qua lại giữa các tiến trình, các công việc thời gian thực cũng như phi thời gian thực, giám soát toàn bộ các xung đột cũng như diễn biến của các quá trình nội tại theo thời gian thực. Ở mức độ quan niệm, mạng Petri (Petri net) là một công cụ rất mạnh trong việc thiết kế những RTS, nó cho phép trình bày toàn bộ các tác động qua lại giữa các tiến trình cũng như diễn tiến của các tiến trình trong hệ thống theo thời gian.

Mạng Petri là một bộ bốn (quadruple) ***C = (P, T, I, O)*** bao gồm: *N trạng thái (places) pi Є P, L chuyển đổi (transition) ti Є T, hai ma trận I và O kích thước L Є M xác định các đầu vào (input) và đầu ra (output) của các chuyển đổi. Thành* phần của ma trận là các số nguyên biểu diễn cho trọng lượng cho mối liên kết giữa *trạng thái và chuyển đổi, nếu không có liên kết thì trọng lượng này bằng 0.*

Cũng cần nói thêm ở đây rằng, đối với mỗi tác vụ xử lý nếu ta nhìn nhận dưới góc độ chi tiết thì sẽ khó mà nhận ra được mối tương quan tổng hoà của nó trong toàn bộ hệ thống. Tuy vậy nếu ta nhìn toàn bộ hệ thống dưới một sơ đồ tổng quát nhưng đủ chi

tiết để tìm ra mối tương quan giữa các thành phần cũng như giữa các xử lý thì càng khó khăn khăn bởi vì tính phức tạp của một tổng thể phức tạp. Trên quan điểm đó, mạng Petri dựa trên mô hình toán học – Đại số tuyến tính – Ma trận cụ thể cho phép có thể cộng trừ nhân chia, cho ra những kết quả cụ thể mà dựa trên đó sẽ có một sự đánh giá chính xác hệ thống cũng như từng thành phần của hệ thống.

$$C = (P, T, I, O)$$

$$P = |p_1, p_2, p_3, p_4, p_5|$$

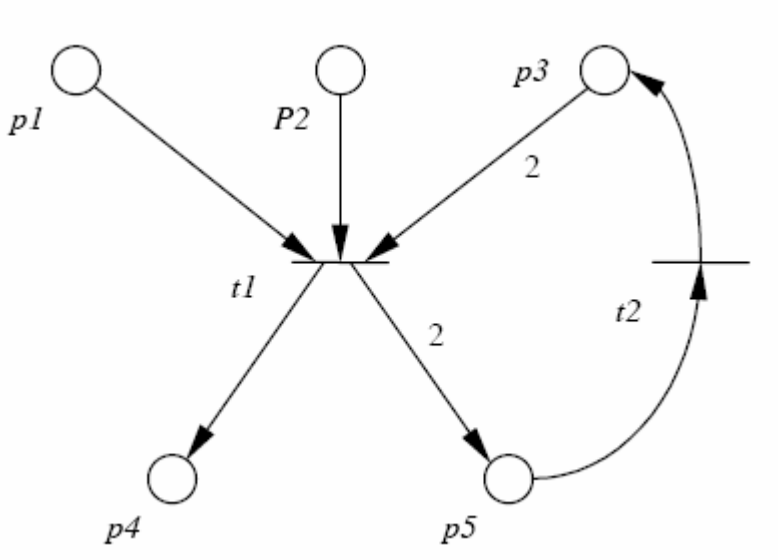
$$T = |t_1, t_2|$$

$$I = \begin{vmatrix} p_1 & p_2 & p_3 & p_4 & p_5 & \\ 1 & 1 & 2 & 0 & 0 & t_1 \\ 0 & 0 & 0 & 0 & 1 & t_2 \end{vmatrix}$$

$$O = \begin{vmatrix} p_1 & p_2 & p_3 & p_4 & p_5 & \\ 0 & 0 & 0 & 1 & 2 & t_1 \\ 0 & 0 & 1 & 0 & 0 & t_2 \end{vmatrix}$$

Ví dụ một mạng Petri

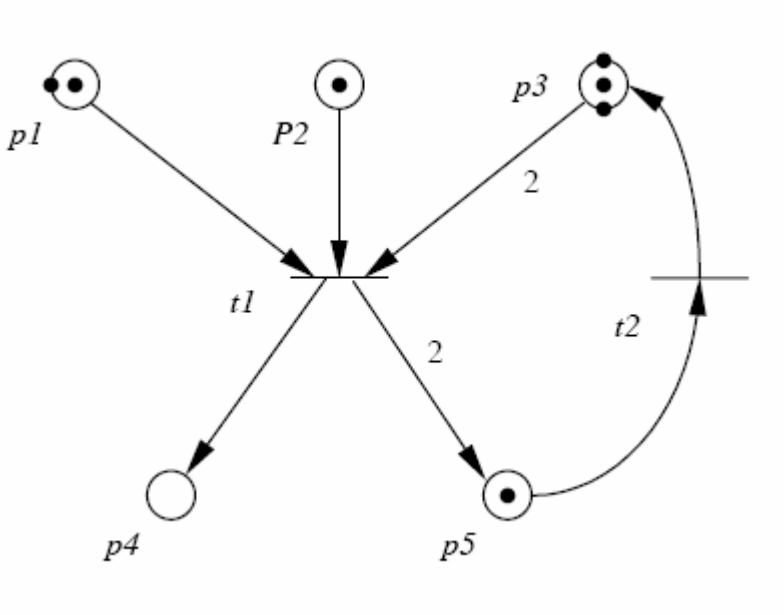
Mạng Petri có thể được biểu diễn bằng đồ thị Petri (Petri graph), với hai loại node: trạng thái và chuyển đổi. Cung nối trực tiếp chỉ liên kết giữa hai node khác loại (trạng thái - chuyển đổi hoặt chuyển đổi - trạng thái, trường hợp đặc biệt có thể là cùng loại).



Hình 8: Đồ thị Petri liên kết với mạng Petri ví dụ trên

Với định nghĩa trên thì mạng Petri chỉ trình bày được những yếu tố tĩnh trong hệ thống. Như vậy sẽ không mô hình hoá được những tác nhân mang tính động. Mạng

Petri đánh dấu (Marked Petri Net) được sử dụng để mô hình sự biến đổi theo thời gian của hệ thống. Trong đó các trạng thái sẽ được đánh dấu bằng những điểm trong đồ thị gọi là thẻ đánh dấu (marking tokens).



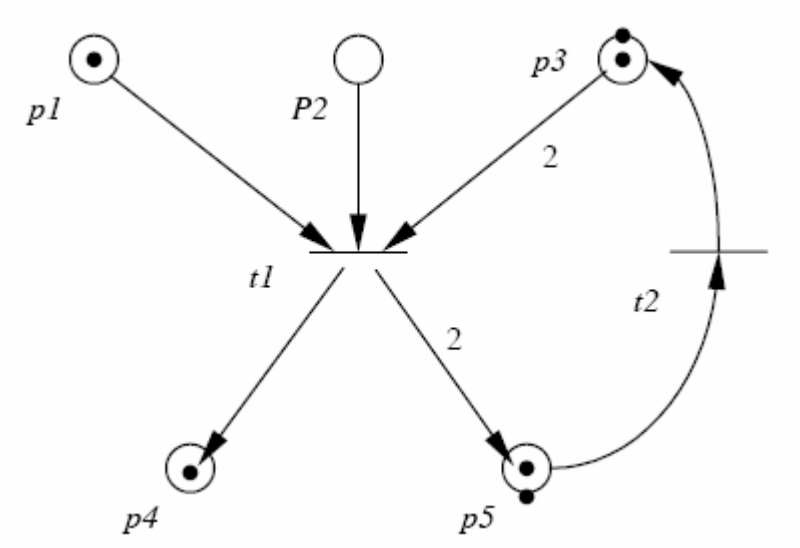
Hình 9: Một đồ thị Petri đánh dấu

Thẻ đánh dấu là một vector N chiều xác định số thẻ mỗi trạng thái . Hệ thống trở thành hệ thống động khi các thẻ lần lược duyệt qua các node trên mạng. Quá trình di chuyển các thẻ xuyên qua các chuyển đổi tới hạn. Một biến đổi được gọi là *tới hạn* chỉ khi tất cả các trạng thái đứng trước nó được đánh dấu, các chuyển đổi này còn được gọi là chuyển đổi cho phép. Chỉ có duy nhất một chuyển đổi tới hạn tại một thời điểm và tuỳ chọn ngẫu nhiên giữa các chuyển đổi cho phép (ưu tiên cho cạnh có trọng lượng lớn nhất). Một chuyển đổi tới hạn kéo theo những ảnh hưởng trên những trạng *thái* đứng trước và sau chuyển đổi đó :

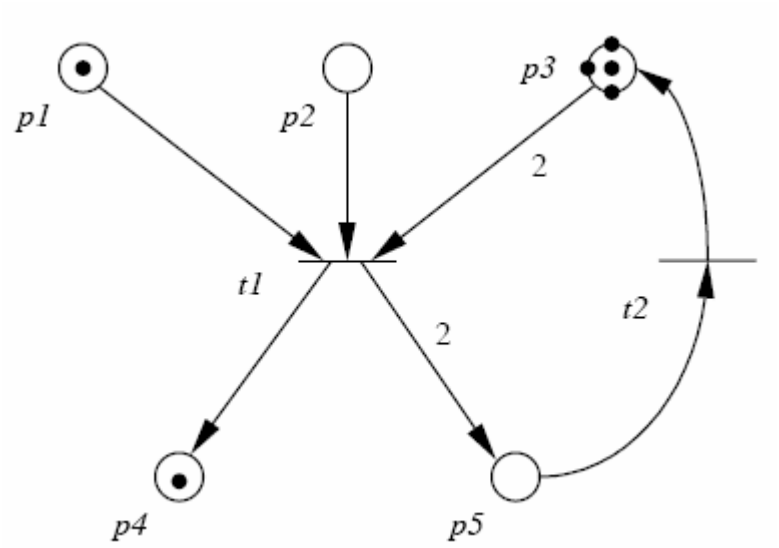
+ w thẻ được gở bỏ khỏi mỗi trạng thái đứng trước.
+ w thẻ được thêm vào mỗi trạng thái đứng sau.

Một giới hạn xảy ra có các tính chất sau :

+ Chủ động : Một chuyển đổi cho phép được tới hạn nhưng không bắt buộc.
+ Trọn vẹn : Tất cả các quá trình liên quan cũng tới hạn.
+ Tức thời : Không có tồn tại thời gian trể giữa các quá trình liên quan.



Hình 10: Một đồ thị Petri hình 8 sau khi chuyển đổi t_1 tới hạn



Hình 11: Một đồ thị Petri hình 8 sau khi kết thúc tất cả các chuyển đổi

Không cần đi vào chi tiết từng thiết kế, ta nhận thấy rằng trong mạng Petri, điều kiện, trạng thái trong thực tế tương ứng với node trạng thái trong mô hình và sự kiện, kết quả tương ứng với chuyển đổi.

Hình 12: Mô tả mạng Petri của một quá trình chia sẽ CPU cho các tiến trình Khi CPU rảnh rỗi (idle) - p2 được đánh dấu. Ngay khi có một tiến trình vào chờ trên hàng đợi CPU - p1 được đánh dấu, tiến trình đó được thực hiện - t1. Kết thúc t1

– p3 được đánh dấu. t2 thực hiện. Kết thúc t2 – p4 được đánh dấu, CPU được giải phóng – p2 được đánh dấu. Quá trình được lặp lại khi có một tiến trình vào hàng đợi CPU.

11. Đồ họa 3D thời gian thực

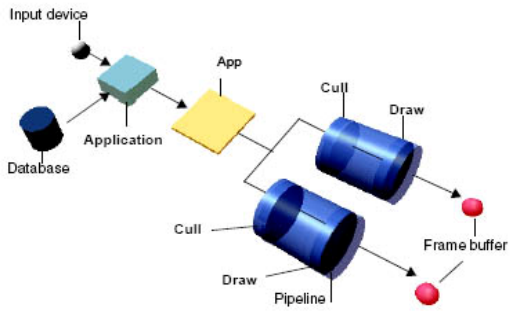
Đến đây thuật ngữ "thời gian thực" đã rõ ràng và không có nghĩa là thực sự nhanh. Đối với hệ thống hiển thị thời gian thực điều kiện tốt nhất của số khung hình hiên thị trên màn hình là trong khoảng 60 đến 85 frames/second. Để hiểu rõ hơn ta hãy tìm hiểu sự khác nhau giữ đồ họa 3D thời gian thực và hoạt cảnh 3D.

Hoạt cảnh được sử dụng trong các sản phẩm Mutilmedia cụ thể là tạo film, hoặc các sản phẩm đồ họa phục vụ công việc in ấn. Còn phần mềm đồ họa thời gian thực được ứng dụng trong các ứng dụng mô phỏng, ví dụ như tập bay, tập lái, trò chơi, có khả năng tương tác. Cả hai loại sản phẩm này đều sử dụng các hình ảnh mô hình thực tế với mức độ chi tiết cao cùng với các thuật toán làm trơn các trạng thái thay đổi của cảnh đồ họa. Thông qua các thuật toán tô bóng số lượng khung hình ngữ cảnh trong một đơn vị thời gian. Nhưng có vài sự khác biệt như sau.

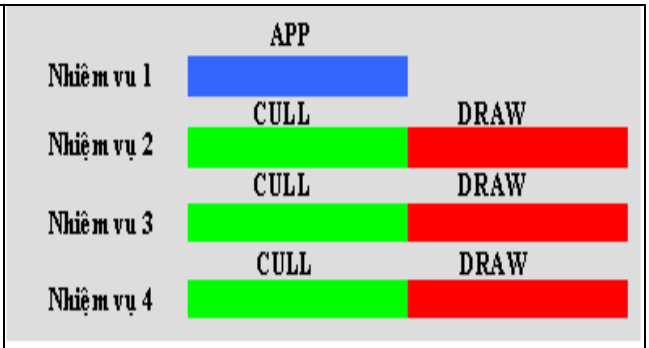
- Phần mềm mô phỏng cần số lượng khung hình là thời gian thực, nghĩa các khung hình phải liên tục khi dữ liệu được cập nhật (bao gồm cả vị trí và hướng quan sát). Còn hoạt cảnh số lượng khung hình được đặt trước, khi tạo cảnh phải mất hàng giờ để tạo.
- Phần mềm mô phỏng cần độ tương tác cao để điều khiển sự chuyển động của các đối tượng trong cảnh. Hoạt cảnh không cho phép tương tác, người dùng cảm nhận thự động với cảnh.
- Sự khác biệt quan trọng khác là tương tác ở mô phỏng là có mục đích. Ngoài ra mô hình trong mô phỏng có phần kém chi tiết hơn trong hoạt cảnh mục đích là để tăng số lượng khung hình
- Phần mềm mô yêu cầu số lượng khung hình đạt 15-60 fps, phụ thuộc vào độ phức tạp và chi tiết của cảnh. Còn film luôn yêu cầu số lượng khung hình là 24 hoặc tùy thuộc vào yêu cầu và chuẩn của nó nhừng số lượng khung hình luôn cố định theo thời gian.

11.1. Hiển thị đồ họa thời gian thực:

Phương pháp truyền thống khi hiển thị cảnh đồ họa 3D thời gian thực sử dụng ba pha riêng biệt APP, CULL, DRAW: APP làm nhiệm vụ cập nhật dữ liệu động, bao gồm vị trí camera, vị trí của các đối tượng chuyển động, CULL phụ thuộc vào APP làm nhiệm vụ lọc cảnh và sắp xếp đối tượng theo độ ưu tiên trong khung nhìn để tăng tốc độ hiển thị cảnh đồng thời tùy thuộc vào việc cập nhật vị trí của camera, tạo dữ liệu hiển thị theo kiểu danh sách để pha DRAW vẽ cảnh lên màn hình. Quá trình vẽ là quá trình duyệt qua danh sách và thông báo cho OpenGL xử lý ta có thể xem hình 12 dưới đây.



Hình 12: Ba pha trong hiển thị đồ họa thời gian thực



Hình 13 - Chia những pha thành những nhiệm vụ song song cho hệ thống có nhiều màn hình hiển thị đồ họa

Trong một hệ thống có nhiều màn hình hiển thị đồ họa, CULL và DRAW trở nên rất cần thiết vì các pha này sẽ sản sinh ra danh sách hiển thị và thực hiện vẽ trên các màn hình ở các khung nhìn khác nhau. Tuy nhiên trong hệ thống đó vẫn chỉ cần một pha APP chung để cập nhật dữ liệu. Dưới đây là trình tự yêu cầu cần thực hiện theo quan điểm đa nhiệm cho nhiều màn hình hiển thị. Với mô hình máy đơn cần xử lý các pha theo từng thời kỳ (APP, CULL0, DRAW0, CULL1, DRAW1, CULL2, DRAW2…). Theo trình tự này cần nhiều thời gian để thực hiện một khung hình qua trình tự thực hiện các pha.

Để phân nhiệm ta định ra hai nhiệm vụ sau được miêu tả như ở hình 13:

- Nhiệm vụ thực hiện pha APP.
- Nhiệm vụ CULL / DRAW cho mỗi màn hình.

Với hệ thống đa xử lý, tất cả các nhiệm vụ có thể được thực hiện song song trên từng bộ xử lý riêng biệt. Hơn nữa, CULL / DRAW có thể được chia ra từng phần để có thể chạy song song trong hệ thống.

Có hai mô hình thực hiện trên hệ thống xử lý song song:

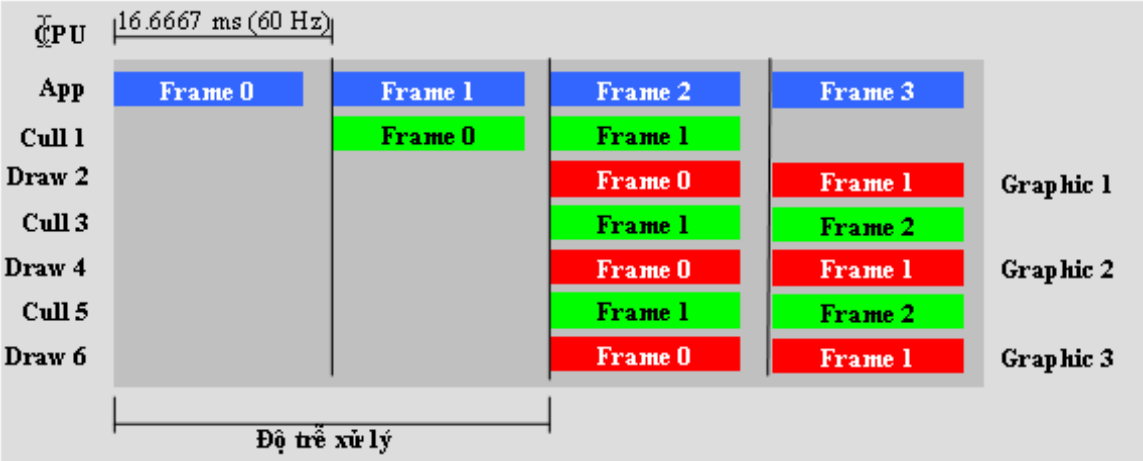
- Chia cắt một nhiệm vụ lớn thành nhiều nhiệm vụ nhỏ độc lập để thực hiện song song từng nhiệm vụ trên mỗi máy để giảm bớt thời gian xử lý.
- Thực hiện nhiệm vụ lớn song song trên các máy sao cho đồng bộ về thời gian trên hệ thống.

Với hai mô hình này ta tách pha CULL/DRAW từ pha APP, rồi sau đó tiếp tục chia pha CULL và DRAW thành nhiều nhiệm vụ chạy song song riêng biệt theo mô hình thứ nhất. Ghép CULL/DRAW thành một nhiệm vụ kép cho mỗi hệ thống hiển thị theo mô hình thứ hai.

Nhưng xuất hiện vài vấn đề ở từng giai đoạn khi chạy song song. Trước hết, những pha phải xử lý dữ liệu ra từng kỳ. Nghĩa là pha APP phải kết thúc làm việc trên dữ liệu trước pha CULL. Tương tự như vậy pha DRAW không thể bắt đầu xử lý dữ liệu khi mà pha CULL chưa làm việc xong. Tuy nhiên, APP không cần phải đợi cho đến khi cả hai pha CULL và DRAW làm việc xong mà vẫn có thể xử lý dữ liệu ở khung hình tiếp theo và hệ thống có thể vận hành theo như Hình 11.3 mô tả dưới đây. Bên cạnh đó dữ liệu dùng chung giữa hai giai đoạn phải được bảo vệ hoặc dùng bộ đệm. Bên cạnh đó dữ liệu đang ghi bởi pha ở giai đoạn này không thể đọc ở cùng pha chạy song song trên hệ thống. Điều này đòi hỏi cần có hệ thống quản lý dữ liệu lớn để xử lý dữ liệu 3D. Đây chính là mô hình của hãng SGI đưa ra và có hiệu quả trong các sản phẩm đồ họa của hãng như ở hình 14. Nhưng vài năm gần đây đã trở lên lạc hậu do một vài lý do sau:

- Vấn đề thời gian thực, khi các sản phẩm mô phỏng yêu cầu độ hiển thị khung hình là 60 Hz nghĩa là mỗi khung hình cần 16.667 mili giây để các pha thực hiện xong nhiệm vụ của nó. Vào những năm 90, SGI đã phát triển kỹ thuật đồ họa thời gian thực với những bộ xử lý có thể đạt các yêu cầu trên nhưng hệ thống máy tính có tốc độ thấp hơn so với hiện tại. Trong khi tốc độ đồ họa phụ thuộc rất nhiều vào hai pha APP và CULL.

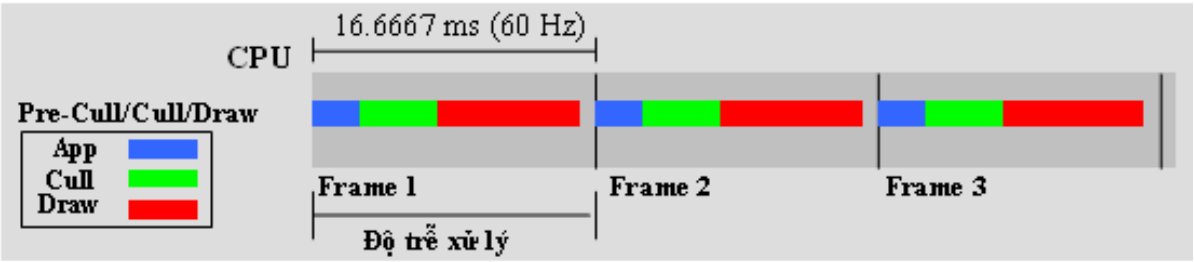
- Hơn thế nữa, hệ thống băng thông rộng phát triển giảm bớt thời gian liên thông giữa các pha ở máy chủ và các máy trạm, cùng với kỹ thuật xử lý đa luồng sẽ có những kết quả khả quan mới.
- Một vấn đề cuối cùng rất cần thiết đó là những yêu cầu đối với thiết bị mô phỏng là phải đáp ứng được các tương tác của người dùng. Vì vậy yêu cầu hình ảnh trực quan và sinh động với tốc độ hiển thị thời gian thực.



Hình 14 - Mô hình xử lý song song "Truyền thống"

11.2. Cách tiếp cận mới

Các ứng dụng đồ họa 3D chất lượng cao chạy trên phần cứng hiện thời mong muốn có số khung hình lớn hơn 60 cần phải có thời gian xử lý của pha Pre - CULL (Có thể hiểu như pha APP – theo mô hình cũ) và CULL nằm trong khoảng 1 mili-giây đến 3,5 mili-giây để thực hiện một khung hình. Xét hệ thống xử lý đơn, một màn hình hiển thị. Với yêu cầu giảm bớt thời gian trên pha Pre – CULL và CULL, sơ đồ các pha có dạng như hình 15.



Hình 15 - Bộ xử lý đơn, một màn hình hiển thị theo mô hình pha Pre – CULL và CULL

Qua sơ đồ trên ta thấy tất cả các pha đều nằm trong một khung hình, và như vậy độ trễ có thể giảm được trong mỗi khung hình. Nhưng pha DRAW chiếm quá nửa thời gian thực hiện một khung hình. Các ứng dụng đồ họa có các lợi thế từ các đồ thị khung cảnh, mà tại đây pha CULL sẽ loại bỏ các đối tượng không thuộc khung hình cần hiển thị để đưa vào các nhánh của đồ thị nhằm tối ưu hóa dữ liệu.

Xét mô hình đa xử lý với nhiều màn hình hiển thị. Cần phải tận dụng hết lợi ích của hệ thống đa xử lý bằng việc sử dụng luồng chính chạy pha Pre - CULL, và các pha CULL/DRAW ở các hệ thống con. Để làm được điều này chúng ta giả thiết hai khía cạnh về quản lý dữ liệu :

1) Dữ liệu được ghi bởi pha Pre –CULL có thuộc tính toàn cục.

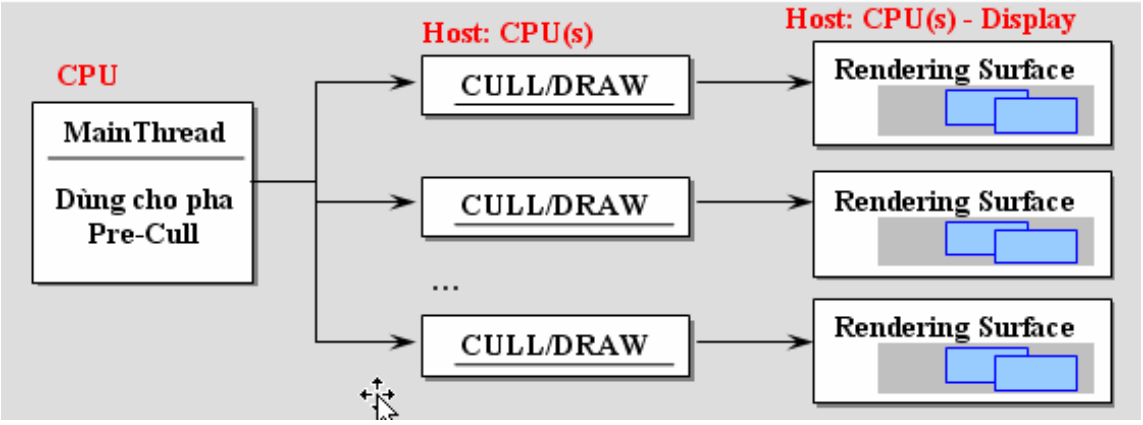
2) Dữ liệu được sinh ra ở các pha CULL mang tính cục bộ nhưng có thể tiếp cận được bởi mỗi cặp nhiệm vụ CULL/DRAW.

11.3. Ứng dụng thiết kế Mô hình xử lý đa luồng trong đồ họa

Mô hình tổng quát được thiết kế theo sơ đồ chung như trên hình 16:

Luồng Chính. Luồng thực hiện Pre - CULL. Khai báo trong hệ là một CPU đảm nhiệm nhiệm vụ đó.

CULL / DRAW. Có thể chạy như một luồng đơn, hoặc những luồng riêng biệt phụ thuộc vào mô hình xử lý được chọn từ mục trước. Điều này có thể được xác định bởi hostname trong hệ thống, và một tham số chỉ định CPU nào trên hệ thống sẽ làm việc với nó để hiển thị cảnh.

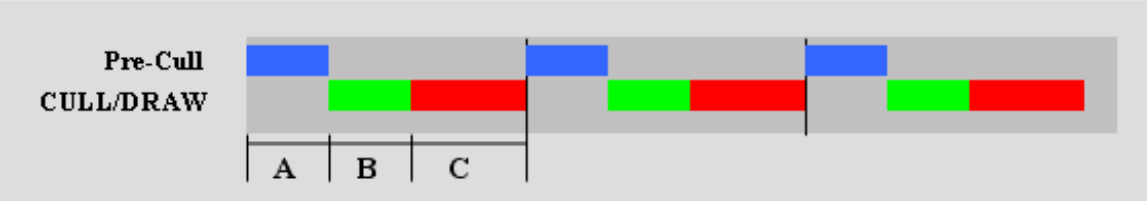


Hình 16 Mô hình xử lý song song tổng quát

Có hai mô hình trong mục trước đã đề cập để thực hiện mô hình đa nhiệm (multi-task), đa màn hình hiển thị đồ họa (multi – display). Sự khác nhau là ở chỗ có quyết định ghép hai pha CULL/DRAW thành một hay không. Ở đây đưa ra hai phương pháp mỗi phương pháp có những đặc tính riêng.

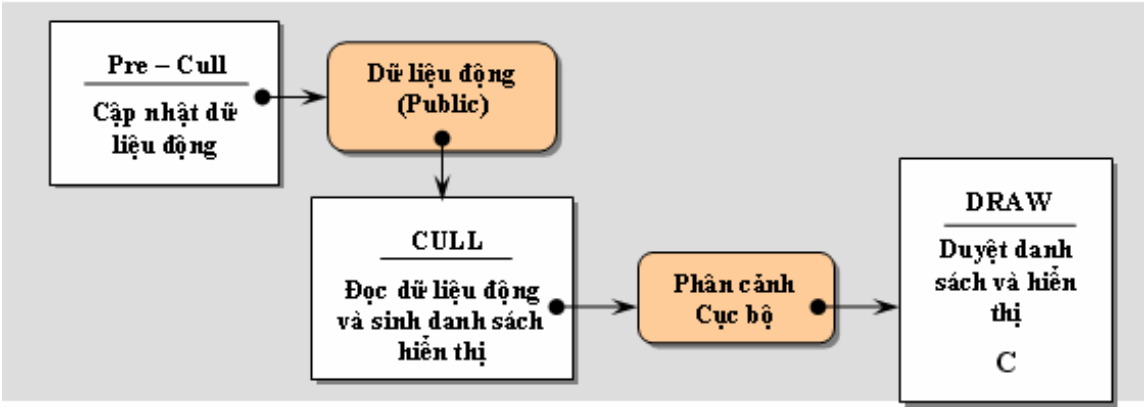
Mô hình A

Mô hình này ghép hai pha CULL/DRAW thành một nhiệm vụ kép được miêu tả theo sơ đồ như hình 17 dưới đây.



Hình 17: Ba pha hiển thị đồ họa thời gian thực theo mô hình A

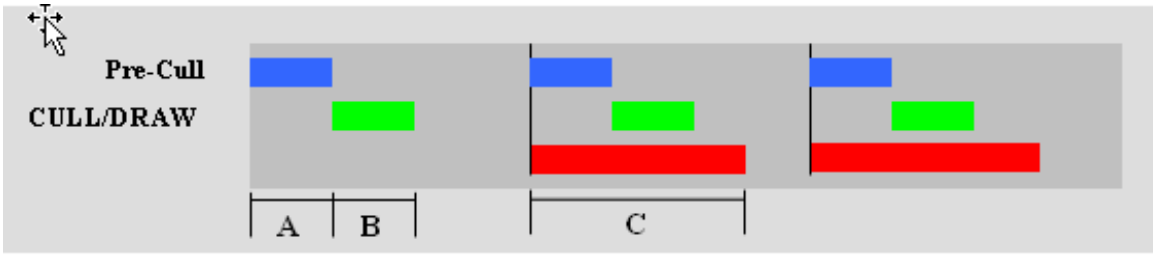
Mô hình này giả thiết một khung hình được thực hiện theo thứ tự, và dùng một luồng cho nhiệm vụ kép CULL/DRAW. Thời gian mỗi tiến trình B và C được thực hiện theo sơ đồ hình 18. Pha Pre – Cull cập nhật dữ liệu động trong đồ thị khung cảnh. Dữ liệu động này bao gồm vị trí camera, định vị những đối tượng chuyển động bên trong cảnh, số khung hình thời gian trôi qua, và đồng bộ những phương tiện quản lý dữ liệu khác. Dữ liệu này được giả thiết là dữ liệu toàn cục, được cấp phát và có thể lấy được bởi ứng dụng. Như vậy, pha CULL phải đợi cho đến khi Pre – CULL kết thúc tiến trình. Khi pha Pre – Cull thực hiện xong sẽ báo hiệu cho pha CULL thực hiện. CULL sẽ đọc dữ liệu động đã được cập nhật, và sinh dữ liệu để hiển thị, dữ liệu này mang tính cục bộ không cho ứng dụng tiếp cận nhưng có thể lấy được từ pha DRAW. Dữ liệu này được xử lý và phân ra từng kỳ. Pha DRAW sẽ duyệt qua danh sách và hiển thị cảnh đồ họa.



Hình 18. Sơ đồ miêu tả quan hệ dữ liệu giữa các pha ở mô hình A

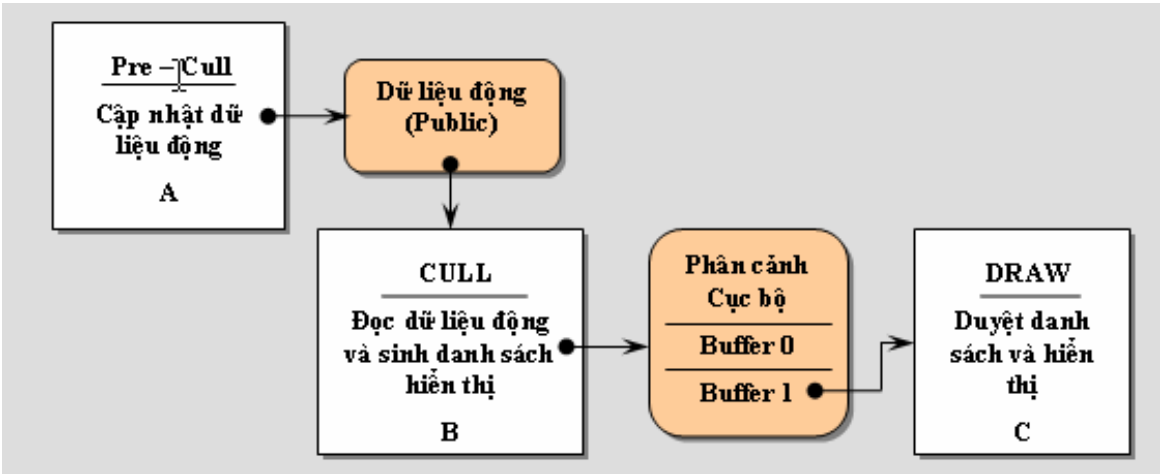
Mô hình B

Mô hình này tách riêng những luồng riêng biệt cho hai pha CULL / DRAW như hình 19



Hình 19: Ba pha hiển thị đồ họa thời gian thực theo mô hình B

Dữ liệu cho mô hình này được mô tả theo sơ đồ hình 20 sau đây.



Hình 20. Sơ đồ miêu tả quan hệ dữ liệu giữa các pha ở mô hình B

Sơ đồ này khác với sơ đồ của mô hình A là dùng một luồng cho nhiệm vụ kép CULL/DRAW . Ở đây pha DRAW sẽ không duyệt dữ liệu được cung cấp bởi pha CULL một cách trực tiếp mà sẽ qua hai bộ đệm. Dữ liệu phát sinh từ pha CULL sẽ

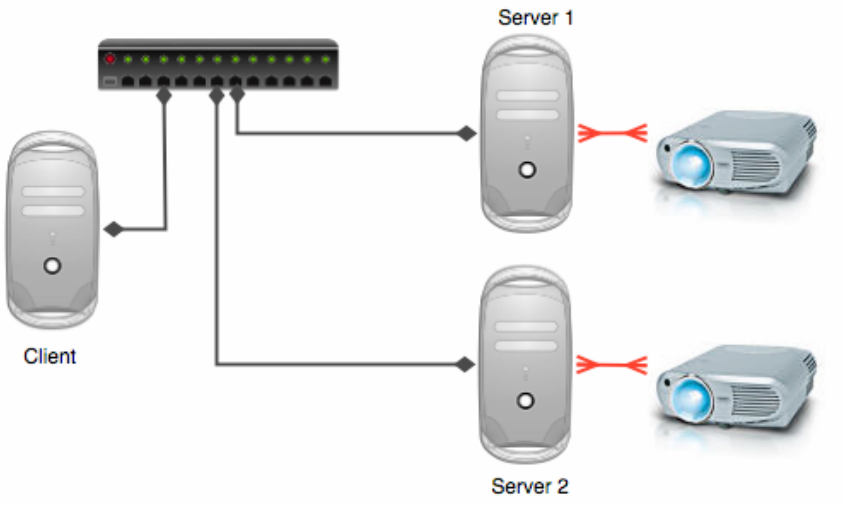
được ghi vào bộ đệm Buffer 0 trong khi pha vẽ sẽ đọc dữ liệu từ bộ đệm Buffer 1. Khi đồng bộ giữa hai pha CULL và DRAW những con trỏ chỉ tới những bộ đệm sẽ được trao đổi. Cách tiếp cận này yêu cầu khi phân cảnh ở pha CULL cần có hai bộ đệm cục bộ, và phải bổ sung việc đồng bộ giữa hai pha CULL và DRAW.

11.4. Ứng dụng thư viện OpenSG trong xây dựng sản phẩm mô phỏng

Hiện nay có nhiều thư viện đồ họa 3D đã tích hợp các thuật toán xử lý song song cảnh đồ họa 3D thời gian thực cho phép xây dựng các sản phẩm mô phỏng với dữ liệu lớn. Nổi bật hơn cả đó là thư viện OpenSG www.opensg.org. OpenSG là một hệ thống thư viện nguồn mở (LGPL) cho phép sử dụng tự do. Nó chạy được trên IRIX, Windows và Linux dựa trên OpenGL.

Yêu cầu sức mạnh tính toán trong một ứng dụng là gần như vô hạn - hoặc yêu cầu lập trình tính toán chuyên ngành hoặc yêu cầu chất lượng cao hiển thị dữ liệu. Để làm được điều đó cần phải có sức mạnh của một siêu máy tính hoặc một nhóm máy cùng xử lý theo cơ chế song song. Nhưng siêu máy tính đắt tiền hơn nhiều lần. Việc dùng nhiều máy tính theo cơ chế song song càng ngày càng được quan tâm trong xây dựng các sản phẩm.

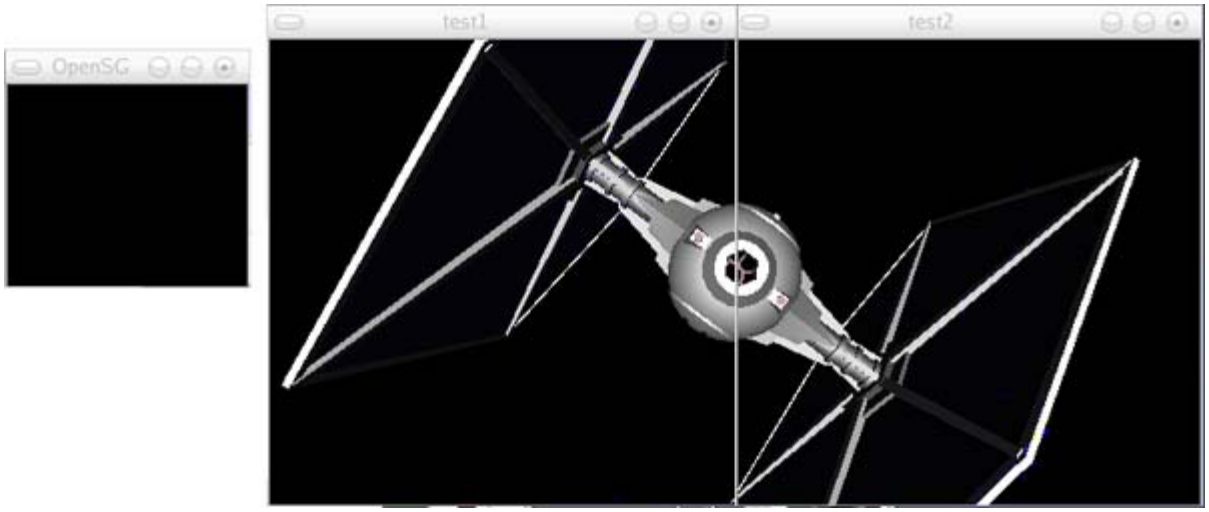
Dưới đây là mô hình ứng dụng 3 máy theo cơ chế song song.



Hình 21 – Mô hình 3 máy tính theo có chế xử lý song song hiển thị 2 màn hình

Trong sơ đồ này cần ba máy: Máy trạm chạy ứng dụng thực hiện hai pha APP và CULL (cập nhật dữ liệu, điều khiển tương tác người máy, thực hiện tính toán mô

phỏng chuyên ngành.). Hai máy chủ chỉ thực hiện pha DRAW nghĩa là chỉ hiển thị cảnh thông qua hai camera khác nhau mô phỏng hai mắt của con người . Với thư viện OpenSG ta có thể thực hiện xử lý song song trên 48 máy.



Hình 22. Một cảnh trả lại trong hai cửa sổ độc lập

Hình ảnh này cho thấy ba những cửa sổ: một nhỏ chỉ cho sự tương tác mô phỏng (sự chuyển động chuột hay bàn phím). Cả hai cửa sổ khác nhau hiển thị một cảnh như thể là được hiển thị trong một cửa sổ. Để làm được điều này cần hai chương trình khác nhau (Client & Server). Giới đây là toàn bộ chương trình được rút gọn.

Chương trình chạy trên máy trạm

```
// all needed include files
#include <OpenSG/OSGGLUT.h>
#include <OpenSG/OSGConfig.h>
#include <OpenSG/OSGSimpleGeometry.h>
#include <OpenSG/OSGGLUTWindow.h>
#include <OpenSG/OSGSimpleSceneManager.h>
#include <OpenSG/OSGMultiDisplayWindow.h>
#include <OpenSG/OSGSceneFileHandler.h>

OSG_USING_NAMESPACE
using namespace std;

SimpleSceneManager *mgr;
NodePtr scene;

int setupGLUT( int *argc, char *argv[] );

int main(int argc, char **argv)
{
#if OSG_MINOR_VERSION > 2
```

```
    ChangeList::setReadWriteDefault();
#endif
    osgInit(argc,argv);

    int winid = setupGLUT(&argc, argv);

    //this time we'll have not a GLUTWindow here, but this one
    MultiDisplayWindowPtr multiWindow = MultiDisplayWindow::create();

    //set some necessary values
    beginEditCP(multiWindow);
       // we connect via multicast
       multiWindow->setConnectionType("Multicast");
       // we want two rendering servers...
       multiWindow->getServers().push_back("Server1");
       multiWindow->getServers().push_back("Server2");
    endEditCP(multiWindow);

    //any scene here
    scene = makeTorus(.5, 2, 16, 16);

    // and the ssm as always
    mgr = new SimpleSceneManager;

    mgr->setWindow(multiWindow);
    mgr->setRoot  (scene);
    mgr->showAll();

    multiWindow->init();

    glutMainLoop();

    return 0;
}

void display(void)
{
    //redrawing as usual
    mgr->redraw();

    //the changelist should be cleared - else things
    //could be copied multiple times
    OSG::Thread::getCurrentChangeList()->clearAll();

    // to ensure a black navigation window
    glClear(GL_COLOR_BUFFER_BIT);
    glutSwapBuffers();
}

void reshape(int w, int h)
{
    glutPostRedisplay();
```

```
}

void mouse(int button, int state, int x, int y)
{
   if (state)
      mgr->mouseButtonRelease(button, x, y);
   else
      mgr->mouseButtonPress(button, x, y);
   glutPostRedisplay();
}

void motion(int x, int y)
{
   mgr->mouseMove(x, y);
   glutPostRedisplay();
}

int setupGLUT(int *argc, char *argv[])
{
   glutInit(argc, argv);
   glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);

   int winid = glutCreateWindow("OpenSG");

   glutReshapeFunc(reshape);
   glutDisplayFunc(display);
   glutMouseFunc(mouse);
   glutMotionFunc(motion);

   return winid;
}
```

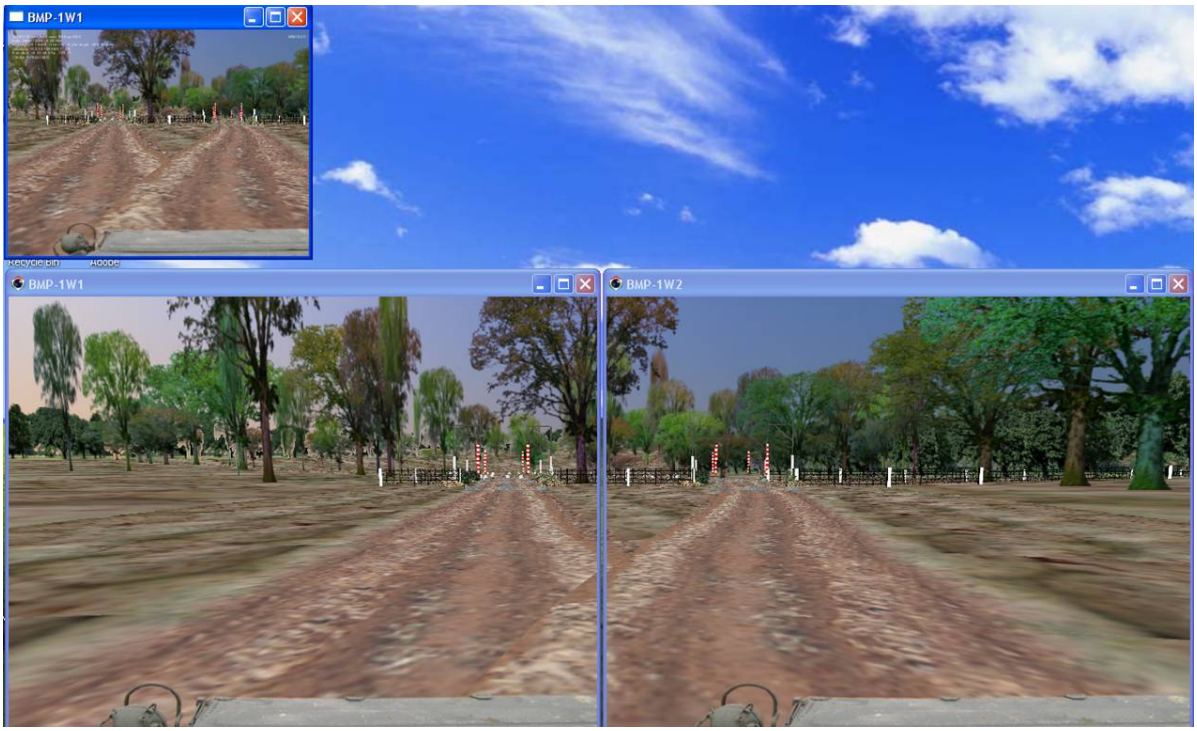
Chương trình chạy trên máy chủ

```
#include <OpenSG/OSGGLUT.h>
#include <OpenSG/OSGConfig.h>
#include <OpenSG/OSGClusterServer.h>
#include <OpenSG/OSGGLUTWindow.h>
#include <OpenSG/OSGRenderAction.h>

OSG_USING_NAMESPACE
using namespace std;

GLUTWindowPtr   window;
RenderAction    *ract;
ClusterServer  *server;

void display();
void reshape( int width, int height );

int main(int argc,char **argv)
{
   int            winid;
```

```
    // initialize Glut
    glutInit(&argc, argv);
    glutInitDisplayMode( GLUT_RGB |GLUT_DEPTH | GLUT_DOUBLE);

    if (!argv[1]){
        cout << "No name was given!" << endl;
        return -1;
    }

    // init OpenSG
#if OSG_MINOR_VERSION > 2
    ChangeList::setReadWriteDefault();
#endif
    osgInit(argc, argv);

    winid = glutCreateWindow(argv[1]);
    glutDisplayFunc(display);
    glutIdleFunc(display);
    glutReshapeFunc(reshape);
    glutSetCursor(GLUT_CURSOR_NONE);

    ract=RenderAction::create();

    window     = GLUTWindow::create();
    window->setId(winid);
    window->init();

    //create a new server that will be connected via multicast
    //argv[1] is the name of the server (at least it should be...)
    server     = new ClusterServer(window,argv[1],"Multicast","");
    server->start();

    glutMainLoop();

    return 0;
}

void display()
{
    //simply execute the render action
    server->render(ract);
    //and delete the change list
    OSG::Thread::getCurrentChangeList()->clearAll();
}

void reshape( int width, int height )
{
    window->resize( width, height );
}
```

12. Môt số kết quả nghiên cứu

Sử dụng kỹ thuật thời gian thực trong đồ họa và ứng dựng thư viene OpenSG, chúng tôi đã xây dựng thành công mô hình song song, đa màn hình hiển thị trên nhiều máy. Số khung hình đạt 60 fps bảo đảm việc tương tác qua lại của người dùng, đáp ứng được yêu cầu đạt ra của các thiết bị tập lái (Hiện được ứng dụng trong sản phẩm hệ thống tập lái xe BMP-1 do TTCN Mô phỏng thực hiện cảnh 3D được hiển thị như ở hình 23)



Hình 23. Cảnh 3 chiều trong hệ thống tập lái xe BMP-1

13. Kết luận

Với cách tiếp cận trong bài báo chúng ta có thể sử dụng lợi thế của phần cứng hiện nay để xây dựng mô hình đa nhiệm, song song, đa luồng và đa màn hình hiển thị trong đồ họa. Mô hình nêu ra ứng dụng tốt trong các sản phẩm đồ họa yêu cầu có dữ liệu lớn và đòi hỏi tương tác thời gian thực.

Tuy vậy, việc hoàn thiện mô hình cần đầu tư thêm nhiều thời gian và công sức, tập hợp được lực lượng đông đảo các giáo viên cùng tham gia nhất là đội ngũ chuyên gia chuyên ngành. Trong tương lai cần hoàn thiện thêm chức năng:

- Quản lý hỗ trợ các thiết bị thực tại ảo (Bao gồm bàn phím, chuột và joistick, Trackball).

14. Tài liệu tham khảo

[1]. *Open Scenegraph*, (Robert Osfield) 2004. www.openscenegraph.org
[2]. *Open SG*, (Dirk Reiners) 2000. www.opensg.org
[3]. Silicon Graphics Inc. SGI OpenGL Optimizer Whitepaper. http://www.sgi.com/software/optimizer/whitepaper.pdf, 1998.
[4]. *Net Juggler Guide*, (Allard, J. et al) http://netjuggler.sourceforge.net 2001
[5]. *MPI: The Complete Reference*, (Snir, M. et al) MIT Press, 1996
[6]. osgVRPN: http://mew.cx/osg/
[7]. VRPN: http://www.cs.unc.edu/Research/vrpn/
[8]. OpenThread: http://openthread.sourceforge.net 2004.
[9]. Open Producer : http://Producer.sourceforge.net 2004